Timed transition discovery from Web service conversation logs (extended version)

Didier Devaurs[†], Kreshnik Musaraj[‡], Fabien De Marchi[‡], Mohand-Saïd Hacid[‡]

[†] University of Windsor; School of Computer Science
401 Sunset Avenue, Windsor, Ontario, Canada, N9B 3P4
ddevaurs@uwindsor.ca

[‡] Université Claude Bernard Lyon 1; LIRIS, UMR CNRS 5205
8 Boulevard Niels Bohr, 69622 Villeurbanne, France
{kreshnik.musaraj, fabien.demarchi, mohand-said.hacid}@liris.cnrs.fr

Abstract

Web service business protocols are of importance to both clients and providers, as they model the external behaviour of services. However, the business protocol is not always published together with the service interface, and this hinders automatic management. When conversation logs are available, a solution is to discover the business protocol from past service executions. This presents many difficulties. One of them is the extraction of some temporal constraints called *timed transitions*, which are not explicitly recorded in the logs. In this paper, we present our approach for discovering such transitions. We formally define a class of patterns called *proper timeouts* and show that they reveal the presence of timed transitions in the business protocol. We also present a polynomial algorithm for extracting such patterns, as well as some preliminary experiments.

Keywords: Web service, business protocol, knowledge extraction, temporal constraint

1 Introduction

Web services are undoubtedly the new well-established generation of Web technologies for application interaction and integration. Their increasing use offers many possibilities, but also raises new questions and problems which need to be quickly answered and solved. A very important ambition associated with Web services relates to loosely-coupled integration, which is already partially carried out by the fact that services use widespread standards (XML, HTTP). However, such flexibility is possible only if users know how to interact with a service. This requires to associate with services elaborate descriptions (such as WSDL, for example) enabling a good understanding of their execution semantics.

It is highlighted in [7] that descriptions like WSDL are not sufficient for a sophisticated and automatic use of Web services, because they provide information only about static properties. This is one of the main reasons which motivated authors in [7] to define a higher level model, the so-called *business protocol*, which specify the external behaviour dynamics of a service (such as allowed conversations or temporal constraints, *cf.* Section 2.1). Since it is formalized by a finite-state machine, the business protocol offers automatic reasoning mechanisms for many applications, such as correctness verification, compatibility testing, *etc.* Nevertheless, the business protocol is not often specified in real life services. Potential reasons include lack of time during implementation, or uncontrolled service evolution (making the initial model obsolete). A solution is to infer the business protocol from the *conversation logs* of a service (*cf.* Section 2.2), when available. Solving this discovery problem could also be very useful for re-engineering applications, such as implementation correctness checking, or service evolution.

Discovering service business protocols includes many technical challenges: analyzing logs for cleaning them from "noise", identifying the different conversations in the logs, defining assessable models for extracted protocols (in terms of relevance and quality), developing refining tools for an interactive extraction, *etc.* The first contribution to this problem has been proposed in [15, 16]. This work handles most of the problems mentioned above, but relates only to *untimed* business protocols. With the importance of temporal constraints in real life services (such as expiration dates), it becomes crucial to extend this discovery technique to *timed business protocols*, which contain not only explicit transitions (related to a message emission or reception), but also *timed transitions* (representing implicit state changes, due to temporal constraints). This is the aim we would like to achieve, by discovering these timed transitions.

Contribution. In this paper, we present our approach for extracting timed transitions from conversation logs. The challenging point of this problem is that no information is recorded when a timed transition is triggered. The first part of our work consists in formally defining a class of patterns called *proper timeouts*, and showing that these patterns are the best representations of timed transitions in the logs. The second part is based on a characterization of the set of proper timeouts satisfied by the logs, which leads to a polynomial extraction algorithm.

Paper organization. In Section 2, we recall the notion of timed business protocol, and stress the importance of its use and discovery. Then, we specify what conversation logs are, we define the problem more precisely, and we present an overview of our approach. Section 3 constitutes our theoretical analysis leading to the definition of proper timeouts. In Section 4, we present our method for extracting proper timeouts from logs, and the initial experiments we performed. We discuss some technical points and present some related work in Section 5. Finally, we draw some conclusions in Section 6.

2 Context, problem and approach

2.1 Timed business protocols

Business protocols [7] were proposed in order to model execution semantics of Web services. They are formalized by deterministic finite-state machines[1], where states represent the various phases in which a service can be during its execution. Transitions are triggered when the service sends or receives messages; each one is labelled by an incoming (+ sign) or an outgoing (− sign) message. A *message* corresponds to the invocation of a service operation or to its reply. A *conversation* is a sequence of messages exchanged between a given service and a customer. A *business protocol* specifies the conversations supported by a service (*i.e.* all valid sequences of message exchanges).

Some state changes are not related to the emission of explicit messages but to temporal constraints (validity period, expiration date, *etc*). The basic model was thus enhanced with timed transitions, and renamed *timed business protocol* [5, 6]. A *timed transition* (also called implicit transition) occurs automatically, after a time interval is elapsed since the transition has been enabled (*i.e.* source state of the transition has become the current state), or after some date is reached; it is labelled by the corresponding time constraint. Note that, since the model is deterministic, a state cannot have several outgoing timed transitions.

Example 1 Fig. 1 illustrates a timed business protocol describing the external behaviour of an order management service. Explicit transitions are represented by solid lines, and timed transitions by dotted lines. This protocol specifies that a customer must initially connect himself (*login* operation), before looking for products (*searchGoods*). Then, the user can add or remove products from its cart (*addToCart*, *removeFromCart*), look for other goods (*searchGoods*), or

[1]Some other proposals for high-level service models are based on Petri nets, *e.g.* [12]

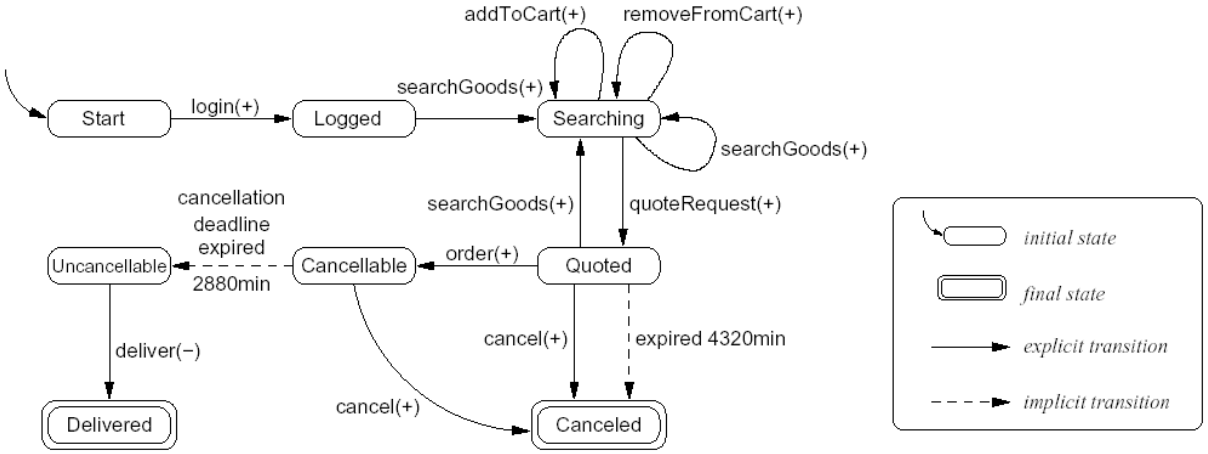


Figure 1: Example of a timed business protocol [5, 6].

ask for a quote (*quoteRequest*), which will be valid only during 3 days (*i.e.* 4320 min); during this period the goods can be ordered (*order*). If the user does not do it, the conversation finishes after 3 days (through the timed transition going out of *Quoted* state), and the order is canceled.

Using a business protocol (instead of a simple interface or code analysis) makes many problems easier to solve, because it is a *visual* model of a service behaviour. For example, it facilitates (i) the development of clients that can interact correctly with a service, (ii) the evolution of a service, when adding new functionalities, constraints or rules. As it is an *automaton*, it can easily be translated in a programming language. Thus, it enables to perform many tasks in an automatic way, such as (i) checking whether messages are sent or received according to specifications (*i.e.* service executions correctness), (ii) testing whether a service is replaceable or compatible with another one, or whether it conforms to a given standard. More generally, it provides a high-level and precise language which allows expressing and reasoning about concepts at their natural level of abstraction. As such, using business protocols can improve service management, analysis and optimization.

Despite the obvious advantages, most of real life services are not described by such a high-level model but only by their WSDL specification. It means that, even if the operations provided by a service are generally published, its valid operation sequences are not always available for potential users. Furthermore, constructing a model from an unstructured representation or from specifications of a service is quite difficult and expensive. That is why a method for extracting the business protocol of a service from its conversation logs was proposed in [15, 16]. Besides the already mentioned advantages of using business protocols, solving this discovery problem offers its own benefits for re-engineering. For example, programmers could check implementation correctness, *i.e.* they could extract the "actual" (really used) protocol of a service, and verify whether it fits the originally specified one; then they could correct the program or prepare the service evolution. Once automated this extraction process could also be applied in dynamic service discovery architectures [9] for automatic service composition and replacement.

Discovering the business protocol of a service includes many technical challenges. First, the model of the extracted protocol has to take into account the uncertainty of the result, and to evaluate its relevance by means of confidence indexes and quality criteria. This uncertainty comes mainly from the fact that service logs can contain some "noise" (*i.e.* some errors). Thus, tools are needed to analyze and clean data before treating them. Another challenging issue is the message correlation, *i.e.* the identification and separation of the different conversations, which can overlap in the logs. Finally, it is also important to propose tools that enable to correct or refine the discovered protocol, according to what the user wants or knows about it.

As already mentioned, a method for business protocol discovery is proposed in [15, 16]. This work is mature and handles most of the problems mentioned above. However, it relates only to *untimed* protocols, which were the starting point of the work realized in [7] on service protocols. With the importance of temporal aspects in real life services (such as expiration dates), it becomes crucial to extend this discovery technique to timed business protocols. This is the goal we would like to achieve, by adding to the method developed in [15, 16] an extracting "module" specialized in timed transitions discovery from conversation logs.

2.2 Conversation logs

Extracting timed transitions would be very easy if we had access to "internal" service logs (defined in the implementation code): it should be enough to make sure that some information is recorded whenever a timed transition is triggered. However it seems not realistic, because in a management platform services are generally considered as "black boxes" whose external behaviour only (published in the interface) can be observed, and not internal operations. In fact the logs we analyze are recordings of conversations taking place between services (*i.e.* messages intercepted by servers or by specific interception tools[2]); that is why they are called *conversation logs*. In the sequel we will consider these logs from the point of view of a given service, *i.e.* we will keep only the messages that it sends or receives. In the logs of this service, there will be explicit traces of all triggered transitions except timed transitions, which are not related to explicit messages.

Various ways for collecting interaction logs of a service have been described, *e.g.* in [10]. According to service implementation and to the tools used for execution management, different information can be logged. In realistic scenarios, at least the content, the sender, the receiver and the timestamp of each message are collected. However this information can be insufficient to separate conversations, which can overlap.[3] It is highlighted in [15] that providing automatically a unique identifier for each conversation (if it is not logged by the management tool) is a research topic by itself. As such, it is assumed in [15] that this information is available. We will make the same assumption. More precisely, we assume that, for each message, the name, the timestamp and the conversation identifier are available.

We now give a more formal definition of logs. Let Msg be a set of *message names*. A *message occurrence* is a couple $M = (m, t)$, where $m \in Msg$ is the message name (denoted by $M.name$ in the sequel), and $t \in \mathbb{R}_+$ is the message timestamp (denoted by $M.time$ in the sequel). Formally, a *conversation* is a sequence of message occurrences $C = \langle M_1, M_2, \ldots, M_n \rangle$, where $n \in \mathbb{N}^*$, and $M_1.time < M_2.time < \cdots < M_n.time$. A *conversation logs file* L is a multi-set of conversations, called simply *logs* L in the sequel.

Example 2 For a service defined with respect to the business protocol illustrated by Fig. 1, we can for example obtain the following logs:
⟨ (login, 9:18), (searchGoods, 9:20), (addToCart, 9:21), (quoteRequest, 9:22), (cancel, 9:51) ⟩
⟨ (login, 11:03), (searchGoods, 11:04), (addToCart, 11:08), (quoteRequest, 11:12) ⟩

2.3 Problem statement and assumptions

Recall that we focus on timed transitions, which are local properties of the business protocol. Discovering the whole automaton is out of the scope of this paper. The problem we address can be set out as follows: Let L be a conversation logs file of a service S, whose underlying timed business protocol P is not known; extract from L the timed transitions of P.

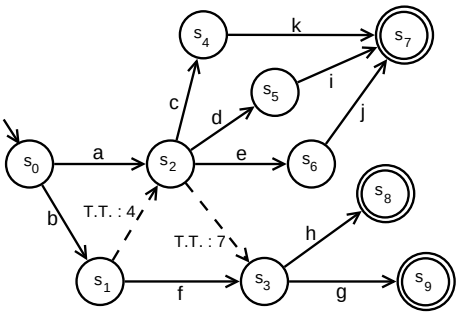
[2] We do not give any more detail about how logs are gathered. This issue is beyond the scope of this paper.

[3] Overlapping conversations result from the interactions of several instances of the service with different clients, in parallel. Isolating these conversations amounts to separating parallel executions of the service.

In order to avoid technical problems, and to focus on the central issue, the problem is simplified, by means of three restrictive working hypothesis, regarding the underlying timed business protocol. First, we assume that *transitions are uniquely labelled*, which means that there do not exist two transitions associated with the same message (*cf.* Fig. 2). This implies that each message in the logs corresponds to only one transition in the protocol. Second, we assume that *no final state has an incoming timed transition*, because our method cannot discover a timed transition that reaches a final state. Such transitions need an additional extracting technique that we plan to consider in future work. Actually, we can expect that, in real life scenarios, a notification message is sent when an expiration occurs. Finally, we assume that *there is no cycle* in the business protocol.

To make sure that all timed transitions can be found, we also assume that *logs are complete*, which means that all valid conversations are recorded in the logs (this point is discussed in Section 5). Finally, we assume that *logs are not noisy* (*i.e.* they are correctly recorded, in the right sequence), in order to propose, in a first step, a deterministic method. In future work we plan to investigate a probabilistic approach to analyze noisy logs.

Example 3 Logs L_1 (*cf.* Fig. 2) will be our running example. Note that timestamps are relative to the beginning of each conversation. Protocol P_1, which has been used for generating logs L_1, is supposed to be unknown. Note also that protocol P_1 is intentionnally simplified compared to general business protocols, which can contain much more information.



$C_1 = \langle (a,0),(c,1),(k,4)\rangle$
$C_2 = \langle (a,0),(c,3),(k,5)\rangle$
$C_3 = \langle (a,0),(d,2),(i,6)\rangle$
$C_4 = \langle (a,0),(d,5),(i,8)\rangle$
$C_5 = \langle (a,0),(e,4),(j,5)\rangle$
$C_6 = \langle (a,0),(e,6),(j,8)\rangle$
$C_7 = \langle (a,0),(h,8)\rangle$
$C_8 = \langle (a,0),(h,10)\rangle$
$C_9 = \langle (a,0),(g,15)\rangle$
$C_{10} = \langle (a,0),(g,16)\rangle$
$C_{11} = \langle (b,0),(f,1),(h,3)\rangle$
$C_{12} = \langle (b,0),(f,2),(g,4)\rangle$
$C_{13} = \langle (b,0),(f,3),(h,4)\rangle$
$C_{14} = \langle (b,0),(f,1),(g,5)\rangle$
$C_{15} = \langle (b,0),(c,6),(k,8)\rangle$
$C_{16} = \langle (b,0),(c,8),(k,9)\rangle$
$C_{17} = \langle (b,0),(d,7),(i,8)\rangle$
$C_{18} = \langle (b,0),(d,10),(i,11)\rangle$
$C_{19} = \langle (b,0),(e,8),(j,10)\rangle$
$C_{20} = \langle (b,0),(e,9),(j,10)\rangle$
$C_{21} = \langle (b,0),(h,13)\rangle$
$C_{22} = \langle (b,0),(h,15)\rangle$
$C_{23} = \langle (b,0),(g,14)\rangle$
$C_{24} = \langle (b,0),(g,15)\rangle$

Figure 2: Protocol P_1 (left) and associated logs L_1 (right).

2.4 Overall presentation of the approach

Recall that a timed transition is part of a business protocol, which is an abstract representation of a service behaviour. Since we only have logs (which are in fact recordings of service executions, in the concrete domain), we cannot work at this abstract level, and directly discover timed transitions. Furthermore, their occurrences cannot explicitly be disclosed by logs, in contrary to explicit transitions, whose labels are recorded. We need to identify some traces (*i.e.* some consequences) of a timed transition, showing that it has been triggered. Thus, our problem is twofold: (i) specify suitable patterns that could correspond to timed transitions, and (ii) extract these patterns from the logs.

We define a class of patterns called *proper timeouts*, and give a condition for a proper timeout to be *satisfied* by the logs. However, no equivalence can be found between the presence of a timed transition, and the satisfaction of a proper timeout. We present a necessary condition, which states that: if a timed transition belongs to the business protocol, then a corresponding proper timeout is satisfied by the logs. Conversely, we say that: a satisfied proper timeout reveals only the presence of a *potential* timed transition. Indeed, another scenario leads also to the satisfaction of a proper timeout, because it creates the same kind of traces in the logs. It

occurs if, in some state of the service, some messages always take longer to be sent or received than others. In both scenarios, the presence of a timed transition will be presumed, because logs by themselves are not sufficient to distinguish them (but we sketch some solutions in Section 5). Details of the theoretical analysis leading to these definitions and results, as well as examples, are presented in Section 3.

Although we cannot establish an equivalence between these patterns, proper timeouts are the best representations of timed transitions, in the logs. Due to the assumption on logs completeness, all timed transitions can be found, because each one of them involves the satisfaction of a proper timeout. That justifies the relevance of a timed transition discovery method based on proper timeout mining.

A basic "generate and test" approach for proper timeouts extraction would be intractable, because it would suffer from combinatorial explosion. Instead we propose a simple characterization of the set of proper timeouts satisfied by the logs (presented in Section 4.1). It is based on a specific (and totally ordered) partition of the occurring episodes, which represent pairs of messages that occur consecutively in a conversation. Once this partition is constructed, the set of proper timeouts is exactly given by its pairs of consecutive elements. Finally, we propose an incremental algorithm for constructing such a partition (presented in Section 4.2). This construction is achievable in polynomial time with respect to the input size (*i.e.* the number of episodes) using only two simple operations (add and merge).

3 Associating patterns with timed transitions

3.1 Episodes

Let Msg be the set of message names, and L the conversation logs file. To reason about consecutive messages, we introduce the notion of episode occurrence.

Definition 1 An **episode** is a sequence of two message names $\alpha = \langle m, m' \rangle$, with $m, m' \in Msg$.

Definition 2 Given an episode $\alpha = \langle m, m' \rangle$, an **occurrence** of α is a sequence of message occurrences $\langle M, M' \rangle$ such that there exists a conversation $C \in L$ satisfying:

$$\begin{cases} M, M' \in C \\ M.name = m \text{ and } M'.name = m' \\ M.time < M'.time \\ \nexists\, M'' \in C \text{ such that } M.time < M''.time < M'.time \end{cases}$$

If such a sequence exists, we say that α occurs in conversation C.

Given an episode α, we denote by $Occ(\alpha)$ its set of occurrences. We say that α occurs in logs L if α occurs in at least one conversation C of L, *i.e.* if $Occ(\alpha) \neq \phi$. We denote by Ep the set of episodes that occur in logs L.

Proposition 1 *Consider the set $P_m = \{\alpha \in Ep | \exists m' \in Msg,\ \alpha = \langle m, m' \rangle\}$, for each $m \in Msg$. Then, $\{P_m \mid m \in Msg\}$ is a partition of Ep.*

This means that one can partition the episodes, such that each part contains all the episodes whose first element is a given message m. It will enable to decompose our discovery task. Instead of analyzing all the episodes as a whole, we will treat each element of this partition separately.

Example 4 Consider logs L_1 (*cf.* Fig. 2). Here, $Ep = P_a \cup P_b \cup P_c \cup P_d \cup P_e \cup P_f$, where $P_b = \{\langle b, c \rangle, \langle b, d \rangle, \langle b, e \rangle, \langle b, f \rangle, \langle b, g \rangle, \langle b, h \rangle\}$ (note that for example $\langle b, i \rangle \notin P_b$), *etc.* Moreover, $Occ(\langle b, h \rangle) = \{ \langle (b, 0), (h, 13) \rangle , \langle (b, 0), (h, 15) \rangle \}$ (but $\langle (b, 0), (h, 3) \rangle \notin Occ(\langle b, h \rangle)$), *etc.*

We define now the concept of occurrence duration. Intuitively, the occurrence duration of an episode occurrence is the difference between the message timestamps in this occurrence. From this, we define the minimal (respect. maximal) occurrence duration of an episode as the smallest (respect. greatest) occurrence duration of all its occurrences. The occurrence duration interval of an episode is the interval which includes all its occurrence durations.

Definition 3 Consider an episode $\alpha \in Ep$.
The duration of an occurrence $\langle M, M' \rangle$ of α is $M'.time - M.time$.
The minimal occurrence duration of α is $d_{min}(\alpha) = min\{M'.time - M.time | \langle M, M' \rangle \in Occ(\alpha)\}$.
Its maximal occurrence duration is $d_{max}(\alpha) = max\{M'.time - M.time \mid \langle M, M' \rangle \in Occ(\alpha)\}$.
The **occurrence duration interval** of α is $[d_{min}(\alpha); d_{max}(\alpha)]$.

In the same way, we define the minimal (respect. maximal) occurrence duration of a set of episodes as the minimum (respect. maximum) of all the minimal (respect. maximal) occurrence durations of these episodes. The occurrence duration interval of a set of episodes is the interval which includes all the occurrence durations of these episodes.

Definition 4 Consider a set of episodes $A \subseteq Ep$ $(A \neq \phi)$.
The minimal occurrence duration of A is $D_{min}(A) = min\{d_{min}(\alpha) \mid \alpha \in A\}$.
The maximal occurrence duration of A is $D_{max}(A) = max\{d_{max}(\alpha) \mid \alpha \in A\}$.
The **occurrence duration interval** of A is $[D_{min}(A); D_{max}(A)]$.

Example 5 Consider logs L_1 (*cf.* Fig. 2). The occurrence duration interval of episode $\langle c, k \rangle$ is $[1; 3]$. The occurrence duration intervals of sets $\{\langle a, c \rangle, \langle a, d \rangle\}$ and $\{\langle a, h \rangle\}$ are $[1; 5]$ and $[8; 10]$ respectively. They are disjoint and satisfy a precedence relation on the time scale. Obviously, this is due to the timed transition between states s_2 and s_3, triggered automatically at time 7 (after arrival in state s_2) if none of the messages c, d or e is emitted before.

Since it is the consequence of the presence of a timed transition, the precedence relation presented in this example will be useful in the sequel. If the reasoning is reversed, finding that such a relation is satisfied by the data could lead to the discovery of a timed transition. Thus, we formalize this relation.

3.2 Order relation on sets of episodes

Definition 5 Consider $A, B \subseteq Ep$ $(A, B \neq \phi)$. We say that A **precedes** B, which is denoted by $A \prec B$, if $D_{max}(A) < D_{min}(B)$. We say that A and B are not comparable, which is denoted by $A \parallel B$, if $A \not\prec B$ and $B \not\prec A$.

Intuitively, the expression $A \prec B$ means that the occurrence duration interval of A precedes that of B on the time scale, and that they are disjoint. We say also that B follows A.

Property 1 *Relation $\prec$ is a strict order relation on $\mathcal{P}(Ep) \setminus \{\phi\}$.*

Proof
– Irreflexivity:
Consider $A \subseteq Ep$ $(A \neq \phi)$.
$D_{min}(A) \leq D_{max}(A) \Rightarrow D_{max}(A) \not< D_{min}(A) \Rightarrow A \not\prec A$.
Therefore, $\forall A \subseteq Ep$ $(A \neq \phi)$, $A \not\prec A$.
– Asymmetry:
Consider $A, B \subset Ep$ such that $A \prec B$.
As $A \prec B$, we have: $D_{max}(A) < D_{min}(B)$.

Since $D_{min}(A) \leq D_{max}(A)$ and $D_{min}(B) \leq D_{max}(B)$, we have: $D_{min}(A) < D_{max}(B)$.
Thus, $D_{max}(B) \not< D_{min}(A)$, *i.e.* $B \not\prec A$.
Therefore, $\forall\, A, B \subset Ep$, $A \prec B \Rightarrow B \not\prec A$.
– *Transitivity*:
Consider $A, B, C \subset Ep$ such that $A \prec B$ and $B \prec C$.
We have: $A \prec B \Rightarrow D_{max}(A) < D_{min}(B)$ and $B \prec C \Rightarrow D_{max}(B) < D_{min}(C)$.
Since $D_{min}(B) \leq D_{max}(B)$, we have: $D_{max}(A) < D_{min}(C)$, *i.e.* $A \prec C$.
Therefore, $\forall\, A, B, C \subset Ep$, we have: $(A \prec B$ and $B \prec C) \Rightarrow A \prec C$.

□

Example 6 Consider logs L_1 (presented in Fig. 2). We have:
· $\{\langle a, c\rangle\} \prec \{\langle a, g\rangle\}$, for $D_{max}(\{\langle a, c\rangle\}) = 3 < 15 = D_{min}(\{\langle a, g\rangle\})$
· $\{\langle a, c\rangle, \langle a, d\rangle\} \prec \{\langle a, g\rangle\}$, for $D_{max}(\{\langle a, c\rangle, \langle a, d\rangle\}) = 5 < 15 = D_{min}(\{\langle a, g\rangle\})$
· $\{\langle a, c\rangle, \langle a, d\rangle\} \prec \{\langle a, g\rangle, \langle a, h\rangle\}$, for $D_{max}(\{\langle a, c\rangle, \langle a, d\rangle\}) = 5 < 8 = D_{min}(\{\langle a, g\rangle, \langle a, h\rangle\})$
· $\{\langle a, c\rangle, \langle a, d\rangle, \langle a, e\rangle\} \prec \{\langle a, g\rangle, \langle a, h\rangle\}$ for $D_{max}(\{\langle a, c\rangle, \langle a, d\rangle, \langle a, e\rangle\}) < D_{min}(\{\langle a, g\rangle, \langle a, h\rangle\})$
· *etc*

Proposition 2 *Consider $m \in Msg$, and $A, B \subset P_m$ $(A, B \neq \phi)$. If there exists a timed transition, in the business protocol, between the state from which the transitions corresponding to the elements of A are going out, and the one from which the transitions corresponding to the elements of B are going out, then $A \prec B$.*

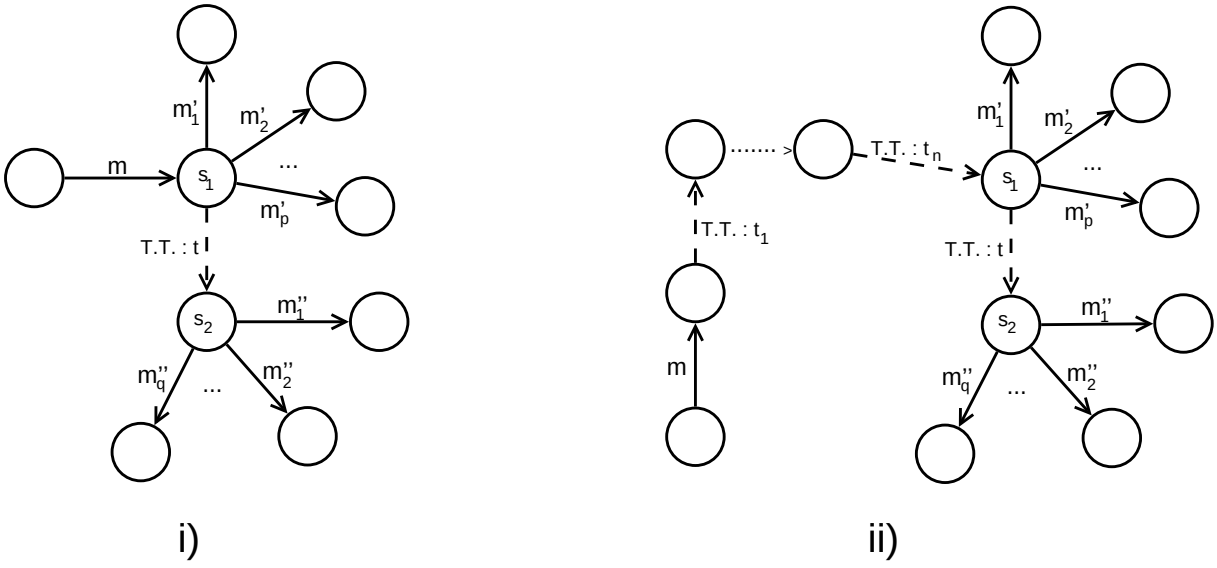


Figure 3: Various configurations associated with a timed transition having as time constraint t.

Proof Consider $m \in Msg$, and $A, B \subset P_m$ $(A, B \neq \phi)$ such that $A = \{\langle m, m'_1\rangle, \ldots, \langle m, m'_p\rangle\}$ and $B = \{\langle m, m''_1\rangle, \langle m, m''_2\rangle, \ldots, \langle m, m''_q\rangle\}$. Since transitions of the business protocol are uniquely labelled, $A \cap B = \phi$.
Assume there exists a timed transition, in the business protocol, between the state (denoted by s_1) from which the transitions labelled by m'_1, m'_2, …, m'_p are going out, and the one (denoted by s_2) from which the transitions labelled by m''_1, m''_2, …, m''_q are going out. With this transition is associated some time constraint t. During a conversation, once state s_1 is reached, if none of the messages m'_1, m'_2, …, m'_p is emitted before time t, the service reaches automatically state s_2. In other words, in a point of view which is global to all executions, we observe that: once state s_1 is reached, messages m'_1, m'_2, …, m'_p can be emitted only before time t, although messages m''_1, m''_2, …, m''_q can be emitted only after.

- If the transition labelled by m come into state s_1 (*cf.* Fig. 3 *i*)):
  We have: $\forall\ 1 \leq i \leq p$, $d_{max}(\langle m, m'_i\rangle) < t$, and $\forall\ 1 \leq j \leq q$, $d_{min}(\langle m, m''_j\rangle) > t$.
  As $D_{max}(A) = max_{1\leq i\leq p}\{d_{max}(\langle m, m'_i\rangle)\}$ and $D_{min}(B) = min_{1\leq j\leq q}\{d_{min}(\langle m, m''_j\rangle)\}$, we have: $D_{max}(A) < t < D_{min}(B)$.
  Therefore, $A \prec B$.
- Else (*cf.* Fig. 3 *ii*)):
  The state in which the transition labelled by m is coming, is linked to s_1 by a sequence of n ($n \geq 1$) timed transitions associated with time constraints t_1, t_2, ..., t_n.
  Then we have: $D_{max}(A) < t + T < D_{min}(B)$, where $T = t_1 + t_2 + \cdots + t_n$.
  Therefore, $A \prec B$.

Remark. We can prove that, if there exists a sequence of m timed transitions ($m \geq 1$) between the states from which the transitions corresponding to the elements of A and B respectively are going out, then $A \prec B$. In this case, if t_1, t_2, ..., t_m are the time constraints associated with the different timed transitions, the proof is similar, with $t = t_1 + t_2 + \cdots + t_m$.

□

This proposition is a necessary condition for the existence of a timed transition. Our problem would be solved if this condition was also sufficient (we would have an object equivalent to a timed transition); but that is not the case.

Remark. The converse of Proposition 2 does not hold.

Counter-example. We have $\{\langle a, c\rangle\} \prec \{\langle a, e\rangle\}$ (for $D_{max}(\{\langle a, c\rangle\}) = 3 < 4 = D_{min}(\{\langle a, e\rangle\})$) in logs L_1, whereas the transitions labelled by c and e are going out of the same state (*cf.* Fig. 2).

Proposition 2 and the assumption on logs completeness guarantee that the set of expressions $A \prec B$ verified by the logs encompasses all timed transitions. However, these expressions can also give false information, about non-existent transitions. This arises from the fact that the relation $\prec$ does not take into account all the information induced by the presence of a timed transition. That is why we define in the sequel a richer relation on sets of episodes.

3.3 Timeouts

Definition 6 A **timeout** is a triplet $T(m, A, B)$, where $m \in Msg$, and $A, B \subset P_m$ ($A, B \neq \phi$). We say that logs L satisfy the timeout $T(m, A, B)$, which is denoted by $L \vDash T(m, A, B)$, if:

$$\begin{cases} A \prec B \\ \forall\, \alpha \in P_m,\ \{\alpha\} \nparallel A \text{ or } \{\alpha\} \nparallel B \end{cases}$$

If $A = \{\langle m, m'_1\rangle, \langle m, m'_2\rangle, \ldots, \langle m, m'_p\rangle\}$, $B = \{\langle m, m''_1\rangle, \ldots, \langle m, m''_q\rangle\}$, and $L \vDash T(m, A, B)$, by abuse of notation, we write: $L \vDash T(m, \{m'_1, m'_2, \ldots, m'_p\}, \{m''_1, m''_2, \ldots, m''_q\})$.

Intuitively, the assertion $L \vDash T(m, A, B)$ means that, according to logs L, (i) occurrence durations of all the episodes in A are strictly less than occurrence durations of all the episodes in B, and that (ii) there is no episode having one occurrence whose duration belongs to the occurrence duration interval of A, and another occurrence whose duration belongs to the occurrence duration interval of B. With this timeout can be associated an expiry time, greater than the maximum occurrence duration of all the episodes in A (*i.e.* $D_{max}(A)$), and less than the minimal occurrence duration of all the episodes in B (*i.e.* $D_{min}(B)$). The possible values for such an expiry time are all real numbers in the interval $]D_{max}(A); D_{min}(B)[$, called the expiry interval of $T(m, A, B)$.

Example 7 $T(b, \{c, d, e\}, \{g, h\})$ is a timeout satisfied by logs L_1 (*cf.* Fig. 2). Indeed, the sets $\{\langle b, c\rangle, \langle b, d\rangle, \langle b, e\rangle\}$ and $\{\langle b, g\rangle, \langle b, h\rangle\}$ satisfy the conditions of Definition 6 for P_b, *i.e.*
$\cdot\ \{\langle b, c\rangle, \langle b, d\rangle, \langle b, e\rangle\} \prec \{\langle b, g\rangle, \langle b, h\rangle\}$ (for $D_{max}(\{\langle b, c\rangle, \langle b, d\rangle, \langle b, e\rangle\}) < D_{min}(\{\langle b, g\rangle, \langle b, h\rangle\})$)
$\cdot\ \forall\, \alpha \in P_b$, $\{\alpha\} \nparallel \{\langle b, c\rangle, \langle b, d\rangle, \langle b, e\rangle\}$ or $\{\alpha\} \nparallel \{\langle b, g\rangle, \langle b, h\rangle\}$ (*e.g.* $\{\langle b, f\rangle\} \prec \{\langle b, g\rangle, \langle b, h\rangle\}$)
We also verify that $L_1 \vDash T(a, \{c, d, e\}, \{g\})$, $L_1 \vDash T(a, \{c, d\}, \{g\})$, $L_1 \vDash T(b, \{f\}, \{c, d, e\})$, $L_1 \vDash T(b, \{f\}, \{g, h\})$, $L_1 \vDash T(b, \{f\}, \{c, d, e, g, h\})$, *etc.*
On the other hand, $L_1 \nvDash T(a, \{c\}, \{e\})$, for $\{\langle a, d\rangle\} \parallel \{\langle a, c\rangle\}$ and $\{\langle a, d\rangle\} \parallel \{\langle a, e\rangle\}$.

Proposition 3 *Consider $m \in Msg$, and $A, B \subset P_m$ $(A, B \neq \phi)$. If there exists a timed transition, in the business protocol, between the state from which the transitions corresponding to the elements of A are going out, and the one from which the transitions corresponding to the elements of B are going out, then $L \vDash T(m, A, B)$.*

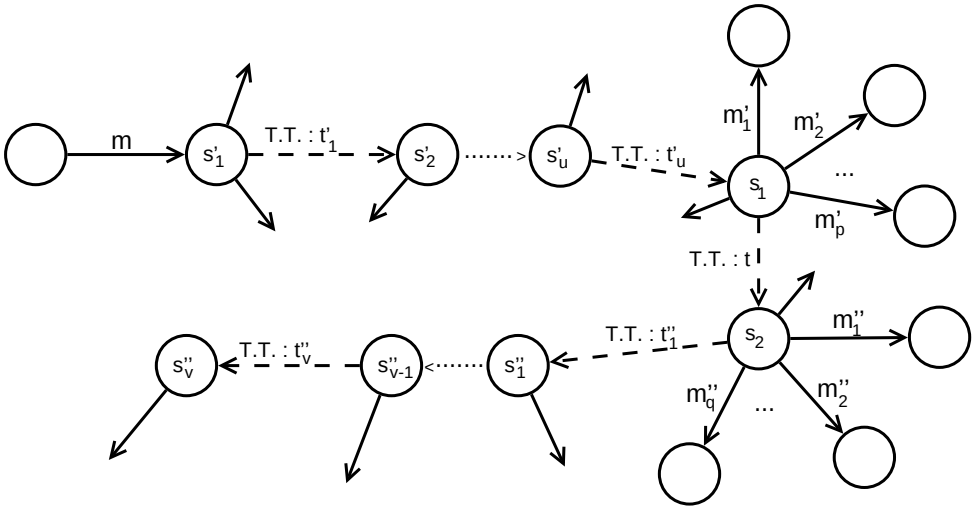


Figure 4: General configuration associated with A and B.

Proof Consider $m \in Msg$, and $A, B \subset P_m$ $(A, B \neq \phi)$ such that $A = \{\langle m, m'_1\rangle, \ldots, \langle m, m'_p\rangle\}$ and $B = \{\langle m, m''_1\rangle, \ldots, \langle m, m''_q\rangle\}$. Assume there exists a timed transition, in the business protocol, between the state from which the transitions labelled by m'_1, m'_2, …, m'_p are going out, and the one from which the transitions labelled by m''_1, m''_2, …, m''_q are going out. This configuration is illustrated by Fig. 4 (where u and v can equal zero). According to Proposition 2, we already know that $A \prec B$.
Consider $\alpha = \langle m, m'\rangle \in P_m$. m' is associated with a transition going out of, either state s_1, or state s_2, or one of the states s'_1, s'_2, …, s'_u, or one of the states s''_1, s''_2, …, s''_v.

- If it is s_1, or one of the states s'_1, s'_2, …, s'_u, we have:
  $\forall\ 1 \leq j \leq q$, $d_{max}(\alpha) < t + \sum_{k=1}^{u} t'_k < d_{min}(\langle m, m''_j\rangle)$.
  Thus: $D_{max}(\{\alpha\}) = d_{max}(\alpha) < min\{d_{min}(\langle m, m''_j\rangle) \mid 1 \leq j \leq q\} = D_{min}(B)$.
  Then, $\{\alpha\} \prec B$, and therefore $\{\alpha\} \nparallel B$.
- If it is s_2, or one of the states s''_1, s''_2, …, s''_v, we have:
  $\forall\ 1 \leq i \leq p$, $d_{min}(\alpha) > t + \sum_{k=1}^{u} t'_k > d_{max}(\langle m, m'_i\rangle)$.
  Thus: $D_{min}(\{\alpha\}) = d_{min}(\alpha) > max\{d_{max}(\langle m, m'_i\rangle) \mid 1 \leq i \leq p\} = D_{max}(A)$.
  Then, $A \prec \{\alpha\}$, and therefore $\{\alpha\} \nparallel A$.
- Conclusion: $\forall\, \alpha \in P_m$, $\{\alpha\} \nparallel A$ or $\{\alpha\} \nparallel B$.

Since $A \prec B$, we have: $L \vDash T(m, A, B)$.

□

Remark. The converse of Proposition 3 does not hold.

Counter-example. The timeout $T(b, \{f\}, \{g, h\})$ is satisfied by logs L_1, although there is no timed transition between states s_1 and s_3 of protocol P_1 (*cf.* Fig. 2). However, a chain composed of two timed transitions connects these states.

Proposition 3 and the assumption on logs completeness guarantee that each timed transition of the business protocol can be found via some timeout satisfied by the logs. However, a timeout is satisfied between two sets of episodes, if a timed transition is present between the states corresponding to those sets of episodes, as well as if it is a chain of timed transitions. Thus, we define a restricted class of timeouts, in order to avoid such ambiguity. Another problem related to timeouts is that they are much more numerous than timed transitions are (*cf.* Ex. 8).

Example 8 Consider protocol P_1 and logs L_1 (*cf.* Fig. 2). The timed transition between states s_1 and s_2 involves the satisfaction of $T(b, \{f\}, \{c, d, e\})$, but also of $T(b, \{f\}, \{c, d, e, g, h\})$. The one between s_2 and s_3 leads to the satisfaction of $T(a, \{c, d, e\}, \{g\})$, but also of $T(a, \{c, d\}, \{g\})$.

This example illustrates that several forms of "redundancy" can appear. Nevertheless, we would like to give to the user the minimal information needed to find the timed transitions. It is in this sense that we define *proper timeouts*.

3.4 Proper timeouts

Definition 7 A **proper timeout** is a triplet $PT(m, A, B)$, where $m \in Msg$, and $A, B \subset P_m$ $(A, B \neq \phi)$. We say that logs L satisfy the proper timeout $PT(m, A, B)$, which is denoted by $L \vDash PT(m, A, B)$, if:

$$\begin{cases} A \prec B \\ \forall\, \alpha \in P_m \setminus (A \cup B),\ \{\alpha\} \nparallel A \cup B \\ \forall\, X, Y \subset A\ (X, Y \neq \phi) \text{ such that } X \cup Y = A, \text{ we have } X \not\prec Y \\ \forall\, X, Y \subset B\ (X, Y \neq \phi) \text{ such that } X \cup Y = B, \text{ we have } X \not\prec Y \end{cases}$$

If $A = \{\langle m, m'_1\rangle, \langle m, m'_2\rangle, \ldots, \langle m, m'_p\rangle\}$, $B = \{\langle m, m''_1\rangle, \ldots, \langle m, m''_q\rangle\}$, and $L \vDash PT(m, A, B)$, by abuse of notation, we write: $L \vDash PT(m, \{m'_1, m'_2, \ldots, m'_p\}, \{m''_1, m''_2, \ldots, m''_q\})$.

Intuitively, the assertion $L \vDash PT(m, A, B)$ means that, according to logs L, (i) occurrence durations of all the episodes in A are strictly less than occurrence durations of all the episodes in B, that (ii) there is no episode, except those of A and B, having occurrences whose durations belong to the occurrence duration interval of $A \cup B$ (interval including the occurrence duration intervals of A and B), and that (iii) there is no partition of A or B composed of two subsets ordered by the relation $\prec$. Obviously, a proper timeout is a timeout. This result is formalized by the following property, whose converse does not hold (*cf.* Ex. 9).

Property 2 *Consider $m \in Msg$, and $A, B \subset P_m$ $(A, B \neq \phi)$. If $L \vDash PT(m, A, B)$, then $L \vDash T(m, A, B)$.*

Proof Consider $m \in Msg$, and $A, B \subset P_m$ $(A, B \neq \phi)$.
Assume that: $L \vDash PT(m, A, B)$. Then we have: $A \prec B$.
Consider $\alpha \in P_m$.

- If $\alpha \in P_m \setminus (A \cup B)$, by assumption, we have: $\{\alpha\} \nparallel A \cup B$, *i.e.* $\{\alpha\} \prec A \cup B$ or $A \cup B \prec \{\alpha\}$.
  Thus: $d_{max}(\alpha) < D_{min}(A \cup B)$ or $d_{min}(\alpha) > D_{max}(A \cup B)$.
  Then, $d_{max}(\alpha) < D_{min}(A)$ or $d_{min}(\alpha) > D_{max}(A)$,
  for $A \subset A \cup B \Rightarrow [D_{min}(A \cup B) \leq D_{min}(A)$ and $D_{max}(A \cup B) \geq D_{max}(A)]$.
  Therefore, $\{\alpha\} \prec A$ or $A \prec \{\alpha\}$, *i.e.* $\{\alpha\} \nparallel A$.

- If $\alpha \in A$, we have: $d_{max}(\alpha) \leq D_{max}(A)$.
  As $D_{max}(A) < D_{min}(B)$ (for $A \prec B$), we have: $d_{max}(\alpha) < D_{min}(B)$.
  Then, $\{\alpha\} \prec B$, and therefore $\{\alpha\} \nparallel B$.
- If $\alpha \in B$, we have: $d_{min}(\alpha) \geq D_{min}(B)$.
  As $D_{min}(B) > D_{max}(A)$ (for $A \prec B$), we have: $d_{min}(\alpha) > D_{max}(A)$.
  Then, $A \prec \{\alpha\}$, and therefore $\{\alpha\} \nparallel A$.
- Conclusion: $\forall\, \alpha \in P_m$, $\{\alpha\} \nparallel A$ or $\{\alpha\} \nparallel B$

Since $A \prec B$, we have: $L \vDash T(m, A, B)$.

□

Example 9 $PT(b, \{c, d, e\}, \{g, h\})$ is a proper timeout satisfied by logs L_1 (*cf.* Fig. 2), because $\{\langle b, c\rangle, \langle b, d\rangle, \langle b, e\rangle\}$ and $\{\langle b, g\rangle, \langle b, h\rangle\}$ satisfy the conditions of Definition 7, for P_b, *i.e.*
$\cdot\ \{\langle b, c\rangle, \langle b, d\rangle, \langle b, e\rangle\} \prec \{\langle b, g\rangle, \langle b, h\rangle\}$
$\cdot\ \{\langle b, f\rangle\} \nparallel \{\langle b, c\rangle, \langle b, d\rangle, \langle b, e\rangle, \langle b, g\rangle, \langle b, h\rangle\}$
$\cdot\ \forall\, X, Y \subset \{\langle b, c\rangle, \langle b, d\rangle, \langle b, e\rangle\}$ $(X, Y \neq \phi)$ such that $X \cup Y = \{\langle b, c\rangle, \langle b, d\rangle, \langle b, e\rangle\}$, we have $X \not\prec Y$
$\cdot\ \{\langle b, g\rangle\} \parallel \{\langle b, h\rangle\}$
We also verify that, for example, $L_1 \vDash PT(a, \{c, d, e\}, \{h\})$. On the other hand, we have:
$\cdot\ L_1 \nvDash PT(b, \{f\}, \{g, h\})$, for $\{\langle b, c\rangle\} \parallel \{\langle b, f\rangle, \langle b, g\rangle, \langle b, h\rangle\}$
$\cdot\ L_1 \nvDash PT(b, \{f\}, \{c, d, e, g, h\})$, for $\{\langle b, c\rangle, \langle b, d\rangle, \langle b, e\rangle\} \prec \{\langle b, g\rangle, \langle b, h\rangle\}$
$\cdot\ L_1 \nvDash PT(a, \{c, d\}, \{g\})$, for $\{\langle a, e\rangle\} \parallel \{\langle a, c\rangle, \langle a, d\rangle, \langle a, g\rangle\}$

Proposition 4 *Consider $m \in Msg$, and $A, B \subset P_m$ $(A, B \neq \phi)$. If there exists a timed transition, in the business protocol, between two states s_1 and s_2 such that the sets of transitions going out of s_1 and s_2 respectively are in bijection with A and B, then there exist $A' \subseteq A$ and $B' \subseteq B$ $(A', B' \neq \phi)$ such that $L \vDash PT(m, A', B')$.*

Proof Consider $m \in Msg$, and $A, B \subset P_m$ $(A, B \neq \phi)$. Assume there exists a timed transition, in the business protocol, between two states s_1 and s_2 such that the sets of transitions going out of s_1 and s_2 are in bijection with A and B respectively.
Consider $\{A_1, A_2, \ldots, A_x\}$ the partition (which can be reduced to one element) of A such that:
$-$ $A_1 \prec A_2 \prec \cdots \prec A_x$ (knowing that $\prec$ is an order relation), and
$-$ $\forall\, 1 \leq k \leq x$, $\forall\, X, Y \subset A_k$ $(X, Y \neq \phi)$ such that $X \cup Y = A_k$, we have $X \not\prec Y$.
Consider $\{B_1, B_2, \ldots, B_y\}$ the partition (which can be reduced to one element) of B such that:
$-$ $B_1 \prec B_2 \prec \cdots \prec B_y$, and
$-$ $\forall\, 1 \leq k \leq y$, $\forall\, X, Y \subset B_k$ $(X, Y \neq \phi)$ such that $X \cup Y = B_k$, we have $X \not\prec Y$.
Denote by m'_1, m'_2, …, m'_p the elements of A_x, and by m''_1, m''_2, …, m''_q the elements of B_1.
This configuration is illustrated by Fig. 4 (where u and v can equal zero).
According to Proposition 3, we already know that $L \vDash T(m, A_x, B_1)$, and thus that $A_x \prec B_1$.
Consider $\alpha = \langle m, m'\rangle \in P_m \setminus (A_x \cup B_1)$. m' is associated with a transition going out of, either state s_1, or state s_2, or one of the states s'_1, s'_2, …, s'_u, or one of the states s''_1, s''_2, …, s''_v.

- If it is one of the states s'_1, s'_2, …, s'_u, we have:
  $\forall\, 1 \leq i \leq p$, $d_{max}(\alpha) < \sum_{k=1}^{u} t'_k < d_{min}(\langle m, m'_i\rangle)$; and
  $\forall\, 1 \leq j \leq q$, $d_{max}(\alpha) < \sum_{k=1}^{u} t'_k < d_{min}(\langle m, m''_j\rangle)$.
  Thus: $d_{max}(\alpha) < min(\{d_{min}(\langle m, m'_i\rangle) \mid 1 \leq i \leq p\} \cup \{d_{min}(\langle m, m''_j\rangle) \mid 1 \leq j \leq q\})$,
  i.e. $D_{max}(\{\alpha\}) < D_{min}(A_x \cup B_1)$.
  Then, $\{\alpha\} \prec A_x \cup B_1$, and therefore $\{\alpha\} \nparallel A_x \cup B_1$.
- If it is one of the states s''_1, s''_2, …, s''_v, we have:
  $\forall\, 1 \leq i \leq p$, $d_{min}(\alpha) > t + t''_1 + \sum_{k=1}^{u} t'_k > d_{max}(\langle m, m'_i\rangle)$; and
  $\forall\, 1 \leq j \leq q$, $d_{min}(\alpha) > t + t''_1 + \sum_{k=1}^{u} t'_k > d_{max}(\langle m, m''_j\rangle)$.
  Thus: $d_{min}(\alpha) > max(\{d_{max}(\langle m, m'_i\rangle) \mid 1 \leq i \leq p\} \cup \{d_{max}(\langle m, m''_j\rangle) \mid 1 \leq j \leq q\})$,
  i.e. $D_{min}(\{\alpha\}) > D_{max}(A_x \cup B_1)$.
  Then, $A_x \cup B_1 \prec \{\alpha\}$, and therefore $\{\alpha\} \nparallel A_x \cup B_1$.

- If it is state s_1 (which implies that $x \geq 2$):
  Since $\alpha \in P_m \setminus A_x$, we have: $\alpha \in A_l$, with $1 \leq l \leq x-1$.
  Thus: $d_{max}(\alpha) \leq D_{max}(A_l) < D_{min}(A_x)$ (for $A_l \prec A_x$, by transitivity).
  Furthermore, $\forall\, 1 \leq j \leq q$, $d_{max}(\alpha) < t + \sum_{k=1}^{u} t'_k < d_{min}(\langle m, m''_j \rangle)$.
  Then, $d_{max}(\alpha) < D_{min}(B_1)$, and therefore $D_{max}(\{\alpha\}) = d_{max}(\alpha) < D_{min}(A_x \cup B_1)$.
  Accordingly, $\{\alpha\} \prec A_x \cup B_1$, and thus $\{\alpha\} \nparallel A_x \cup B_1$.
- If it is state s_2 (which implies that $y \geq 2$):
  Since $\alpha \in P_m \setminus B_1$, we have: $\alpha \in B_h$, with $2 \leq h \leq y$.
  Thus: $d_{min}(\alpha) \geq D_{min}(B_h) > D_{max}(B_1)$ (for $B_1 \prec B_h$, by transitivity).
  Furthermore, $\forall\, 1 \leq i \leq p$, $d_{min}(\alpha) > t + \sum_{k=1}^{u} t'_k > d_{max}(\langle m, m'_i \rangle)$.
  Then, $d_{min}(\alpha) > D_{max}(A_x)$, and therefore $D_{min}(\{\alpha\}) = d_{min}(\alpha) > D_{max}(A_x \cup B_1)$.
  Accordingly, $A_x \cup B_1 \prec \{\alpha\}$, and thus $\{\alpha\} \nparallel A_x \cup B_1$.
- Conclusion: $\forall\, \alpha \in P_m \setminus (A_x \cup B_1)$, $\{\alpha\} \nparallel A_x \cup B_1$.

By assumption, we have also:

- $\forall\, X, Y \subset A_x$ $(X, Y \neq \phi)$ such that $X \cup Y = A_x$, we have $X \not\prec Y$;
- $\forall\, X, Y \subset B_1$ $(X, Y \neq \phi)$ such that $X \cup Y = B_1$, we have $X \not\prec Y$.

Therefore, $L \vDash PT(m, A_x, B_1)$, with $A_x \subseteq A$ and $B_1 \subseteq B$ $(A_x, B_1 \neq \phi)$.

□

Remark. The converse of Proposition 4 does not hold.

Counter-example. We have $L_1 \vDash PT(a, \{h\}, \{g\})$, although the transitions labelled by h and g are going out of the same state (*cf.* Fig. 2). The satisfaction of this proper timeout is explained by the fact that, in logs L_1, after message a has been emitted, message g always takes longer to be emitted than message h.

According to Proposition 4 and the assumption on logs completeness, since each timed transition involves the satisfaction by the logs of a proper timeout, we can find all of them. However, we can discover more proper timeouts than there are timed transitions, if some messages always take longer to be sent or received than the messages associated with all other transitions of the same state. The following theorem states that this is the only possible alternative.

Theorem 1 *Consider $m \in Msg$, and $A, B \subset P_m$ $(A, B \neq \phi)$. If $L \vDash PT(m, A, B)$, then there exist in the business protocol:*

- *either two states s_1 and s_2, such that s_2 is linked to s_1 by a timed transition, that A corresponds to a subset of the transitions going out of s_1, and that B corresponds to a subset of the transitions going out of s_2,*
- *or one state s, such that $A \cup B$ corresponds to a subset of the transitions going out of s, and that the messages in B always take longer to be emitted than those in A.*

Proof Consider $m \in Msg$, and $A, B \subset P_m$ $(A, B \neq \phi)$. Assume that $L \vDash PT(m, A, B)$, *i.e.* :

$$\begin{cases} A \prec B & (1) \\ \forall\, \alpha \in P_m \setminus (A \cup B),\ \{\alpha\} \nparallel A \cup B & (2) \\ \forall\, X, Y \subset A\ (X, Y \neq \phi) \text{ such that } X \cup Y = A,\ \text{we have } X \not\prec Y & (3) \\ \forall\, X, Y \subset B\ (X, Y \neq \phi) \text{ such that } X \cup Y = B,\ \text{we have } X \not\prec Y & (4) \end{cases}$$

Denote by s_m the state of the business protocol in which the transition labelled by m is coming. Since $A, B \subset P_m$, the elements in $A \cup B$ correspond to transitions which are going out of, either s_m, or a state linked to s_m by a timed transition, or by a sequence of timed transitions.

- If all these transitions are going out of the same state (s_m or another):
  Since $A \prec B$, we know that the messages associated with the elements of B always take longer to be emitted than those associated with elements of A.

- If these transitions are going out of two different states s_1 and s_2:
  s_1 and s_2 are linked by a sequence of p timed transitions, where $p \geq 1$ (if $p = 1$, it is a single timed transition). Without loss of generality, we can assume that s_1 is at the "beginning" of the sequence, and s_2 at the "end". Denote by E_1 and E_2 the sets of episodes from P_m corresponding to the transitions going out of s_1 and s_2 respectively.
  We have: $A \cup B \subseteq E_1 \cup E_2$, $(A \cup B) \cap E_1 \neq \phi$, and $(A \cup B) \cap E_2 \neq \phi$.
  Assume that $E_1 \cap A = \phi$.
    We have $A \subseteq E_2$, and therefore $E_1 \cap B \neq \phi$.
    Assume that $E_2 \cap B \neq \phi$.
      We have $(E_1 \cap B) \cup (E_2 \cap B) = B$.
      Since s_2 is linked to s_1 by a sequence of timed transitions, we have: $E_1 \cap B \prec E_2 \cap B$, which contradicts (4).
    Then, $E_2 \cap B = \phi$, and thus $B \subseteq E_1$.
    As $A \subseteq E_2$, and s_2 is at the end of the timed transitions sequence, we have: $B \prec A$, which contradicts (1).
  Therefore, $E_1 \cap A \neq \phi$.
  Assume that $E_2 \cap A \neq \phi$.
    We have $(E_1 \cap A) \cup (E_2 \cap A) = A$.
    Since s_2 is linked to s_1 by a sequence of timed transitions, we have: $E_1 \cap A \prec E_2 \cap A$, which contradicts (3).
  Therefore, $E_2 \cap A = \phi$.
  Then, $A \subseteq E_1$, and thus $E_2 \cap B \neq \phi$.
  As we cannot have $E_1 \cap B \neq \phi$, we have: $B \subseteq E_2$.
  Thus, A and B correspond to two sets of transitions going out of s_1 and s_2 respectively.

  Assume that $p \geq 2$ (*i.e.* the sequence is composed of several transitions).
  There exists a state s_3 "between" s_1 and s_2.
  Denote by E_3 the set of episodes from P_m associated with the transitions going out of s_3.
  Consider $\alpha \in E_3$.
  We have: $d_{min}(\alpha) < D_{max}(B) = D_{max}(A \cup B)$ (for $A \prec B$), and thus $A \cup B \not\prec \{\alpha\}$.
  Furthermore, $d_{max}(\alpha) > D_{min}(A) = D_{min}(A \cup B)$ (for $A \prec B$), and thus $\{\alpha\} \not\prec A \cup B$.
  Therefore, $\{\alpha\} \parallel A \cup B$ with $\alpha \in P_m \setminus (A \cup B)$, which contradicts (2).
  Consequently, $p = 1$, *i.e.* s_2 is linked to s_1 by a single timed transition.

- If these transitions are going out of n states s_1, s_2, ..., s_n (with $n \geq 3$):
  Denote by E_1, E_2, ..., E_n the sets of episodes from P_m which correspond to the transitions going out of s_1, s_2, ..., s_n respectively.
  We have: $A \cup B \subseteq \bigcup_{i=1}^{n} E_i$, and $\forall\ 1 \leq i \leq n$, $(A \cup B) \cap E_i \neq \phi$.
  Assume that: $\exists\ i \in [\![1; n]\!]$ such that $A \subseteq E_i$.
    If $I = [\![1; n]\!] \setminus \{i\}$, we have: $card(I) \geq 2$, and $\forall\ k \in I$, $B \cap E_k \neq \phi$.
    Therefore, $\exists\ j \in I$ such that $B \cap E_j \prec \bigcup_{k \in I \setminus \{j\}} (B \cap E_k)$, which contradicts (4).
  Thus, $\nexists\ i \in [\![1; n]\!]$ such that $A \subseteq E_i$.
  If $J = \{k \in [\![1; n]\!] \mid A \cap E_k \neq \phi\}$, we have: $card(J) \geq 2$.
  Therefore, $\exists\ j \in J$ such that $A \cap E_j \prec \bigcup_{k \in J \setminus \{j\}} (A \cap E_k)$, which contradicts (3).
  Thus, we cannot have $n \geq 3$.

□

Although we cannot ensure a total correspondence between these objects, proper timeouts are the best possible representations of timed transitions, in the logs, for practical purposes. Theorem 1 guarantees that, if we discover a proper timeout in the logs, then there is a timed transition in the business protocol, or some messages take longer to be emitted than others,

knowing that the logs by themselves are not sufficient to detect such eventuality. Thus, we will say that: a satisfied proper timeout reveals the presence of a *potential* timed transition. That justifies the relevance of the development of a timed transition discovery method based on the research of the proper timeouts satisfied by the logs.

4 Extracting the proper timeouts

The "naive" discovery method consists in generating all possible proper timeouts, and testing for each one of them if the conditions of Definition 7 are satisfied. However, this method is doubly exponential because (i) the number of candidates is exponential, and (ii) for each candidate, if $A \prec B$, we must check that all subsets of A and B are not comparable with respect to $\prec$. Instead, we propose a nice characterization of the set of proper timeouts satisfied by the logs, which will enable us to construct this set in a very simple way.

4.1 Characterization of the satisfied proper timeouts

This characterization is formalized by the following theorem. It states that: for a set P_m of episodes whose first message is m, if we build a partition Π of P_m such that (i) Π is totally ordered by the relation $\prec$, and (ii) for each element of Π, there does not exist a sub-partition of this element composed of two subsets ordered by the relation $\prec$; then (a) there is a proper timeout "between" each pair of consecutive elements in Π, and (b) these are the only proper timeouts satisfied by the logs and related to message m .

Theorem 2 *Consider $m \in Msg$, $i_m \in \mathbb{N}^*$, and $\{P_m^{(1)}, P_m^{(2)}, \ldots, P_m^{(i_m)}\}$ a partition of P_m. The following assertions are equivalent:*

i) $\begin{cases} P_m^{(1)} \prec P_m^{(2)} \prec \cdots \prec P_m^{(i_m)}, \text{ and} \\ \forall\, 1 \leq i \leq i_m,\ \forall\, X, Y \subset P_m^{(i)}\ (X, Y \neq \phi) \text{ such that } X \cup Y = P_m^{(i)},\ \text{we have } X \not\prec Y \end{cases}$

ii) $\begin{cases} \forall\, 1 \leq i < i_m,\ L \vDash PT(m, P_m^{(i)}, P_m^{(i+1)}), \text{ and} \\ \forall\, A, B \subset P_m\ (A, B \neq \phi) \text{ such that } L \vDash PT(m, A, B), \text{ we have:} \\ \qquad \exists\, i \in [\![1; i_m - 1]\!] \text{ such that } A = P_m^{(i)} \text{ and } B = P_m^{(i+1)} \end{cases}$

Proof Consider $m \in Msg$, $i_m \in \mathbb{N}^*$, and $\{P_m^{(1)}, P_m^{(2)}, \ldots, P_m^{(i_m)}\}$ a partition of P_m.
i) $\Rightarrow$ *ii)*: Assume that

$$\begin{cases} P_m^{(1)} \prec P_m^{(2)} \prec \cdots \prec P_m^{(i_m)}, \text{ and} \\ \forall\, 1 \leq i \leq i_m, \forall\, X, Y \subset P_m^{(i)}\ (X, Y \neq \phi) \text{ such that } X \cup Y = P_m^{(i)},\ \text{we have } X \not\prec Y \end{cases}$$

Consider $i \in [\![1; i_m - 1]\!]$.

- $P_m^{(i)} \prec P_m^{(i+1)}$.
- $\forall\, X, Y \subset P_m^{(i)}\ (X, Y \neq \phi)$ such that $X \cup Y = P_m^{(i)}$, we have $X \not\prec Y$.
- $\forall\, X, Y \subset P_m^{(i+1)}\ (X, Y \neq \phi)$ such that $X \cup Y = P_m^{(i+1)}$, we have $X \not\prec Y$.
- Consider $\alpha \in P_m \setminus (P_m^{(i)} \cup P_m^{(i+1)})$.
  Since $\{P_m^{(j)} \mid 1 \leq j \leq i_m\}$ is a partition of P_m, $\exists\, j \in [\![1; i_m]\!] \setminus \{i, i+1\}$ such that $\alpha \in P_m^{(j)}$.
  - If $j < i$, then $P_m^{(j)} \prec P_m^{(i)}$ and $P_m^{(j)} \prec P_m^{(i+1)}$.
    As $\{\alpha\} \subseteq P_m^{(j)}$, we have: $\{\alpha\} \prec P_m^{(i)}$ and $\{\alpha\} \prec P_m^{(i+1)}$.
    Thus, $\{\alpha\} \prec P_m^{(i)} \cup P_m^{(i+1)}$, and thereby $\{\alpha\} \not\parallel P_m^{(i)} \cup P_m^{(i+1)}$.
  - If $j > i + 1$, then $P_m^{(i)} \prec P_m^{(j)}$ and $P_m^{(i+1)} \prec P_m^{(j)}$.
    As $\{\alpha\} \subseteq P_m^{(j)}$, we have: $P_m^{(i)} \prec \{\alpha\}$ and $P_m^{(i+1)} \prec \{\alpha\}$.
    Thus, $P_m^{(i)} \cup P_m^{(i+1)} \prec \{\alpha\}$, and thereby $\{\alpha\} \not\parallel P_m^{(i)} \cup P_m^{(i+1)}$.
  - Conclusion: $\forall\, \alpha \in P_m \setminus (P_m^{(i)} \cup P_m^{(i+1)})$, $\{\alpha\} \not\parallel P_m^{(i)} \cup P_m^{(i+1)}$.

- Therefore, $L \vDash PT(m, P_m^{(i)}, P_m^{(i+1)})$.

Consequently, $\forall\, 1 \leq i < i_m,\ L \vDash PT(m, P_m^{(i)}, P_m^{(i+1)})$.

Consider $A, B \subset P_m (A, B \neq \phi)$ such that $L \vDash PT(m, A, B)$. We have:

$$\begin{cases} A \prec B & (1) \\ \forall\, \alpha \in P_m \setminus (A \cup B),\ \{\alpha\} \nVdash A \cup B & (2) \\ \forall\, X, Y \subset A\ (X, Y \neq \phi) \text{ such that } X \cup Y = A, \text{ we have } X \not\prec Y & (3) \\ \forall\, X, Y \subset B\ (X, Y \neq \phi) \text{ such that } X \cup Y = B, \text{ we have } X \not\prec Y & (4) \end{cases}$$

- Since $A \subset P_m$ and $\{P_m^{(j)} \mid 1 \leq j \leq i_m\}$ is a partition of P_m, we know that: $\exists\, i \in [\![1; i_m]\!]$ such that $A \cap P_m^{(i)} \neq \phi$.
  We can define $I_A = \{i \in [\![1; i_m]\!] \mid A \cap P_m^{(i)} \neq \phi\}$, $i_A = max(I_A)$ and $I'_A = I_A \setminus \{i_A\}$.
  Assume that $I'_A \neq \phi$.
  We have: $A = \bigcup_{i=1}^{i_m} (A \cap P_m^{(i)}) = \bigcup_{i \in I_A} (A \cap P_m^{(i)}) = (A \cap P_m^{(i_A)}) \cup (\bigcup_{i \in I'_A} (A \cap P_m^{(i)}))$, with $A \cap P_m^{(i_A)} \subset A$ and $\bigcup_{i \in I'_A} (A \cap P_m^{(i)}) \subset A$.
  However, $\forall\, i \in I'_A$, $i < i_A$, and thus: $\forall\, i \in I'_A$, $P_m^{(i)} \prec P_m^{(i_A)}$.
  Since $\forall\, i \in I_A$, $A \cap P_m^{(i)} \subseteq P_m^{(i)}$, we have: $\forall\, i \in I'_A$, $A \cap P_m^{(i)} \prec A \cap P_m^{(i_A)}$.
  Therefore $\bigcup_{i \in I'_A} (A \cap P_m^{(i)}) \prec A \cap P_m^{(i_A)}$, which contradicts (3).
  Thereby, $I'_A = \phi$.
  Consequently, $\exists!\, i \in [\![1; i_m]\!]$ such that $A \cap P_m^{(i)} \neq \phi$, *i.e.* $\exists\, i \in [\![1; i_m]\!]$ such that $A \subseteq P_m^{(i)}$.
- In the same way (and according to (4)), we prove that: $\exists\, j \in [\![1; i_m]\!]$ such that $B \subseteq P_m^{(j)}$.
- Assume that: $A \subsetneq P_m^{(i)}$.
  Consider $\alpha \in P_m^{(i)} \setminus A \subset P_m \setminus A$.
  - If $\alpha \in B$, since $A \prec B$, we have: $A \prec \{\alpha\}$.
  - If $\alpha \notin B$, $\alpha \in P_m \setminus (A \cup B)$ and, according to (2), $\{\alpha\} \nVdash A \cup B$.
    Thus, $\{\alpha\} \nVdash A$, *i.e.* $\{\alpha\} \prec A$ or $A \prec \{\alpha\}$.
  - Conclusion: $\forall\, \alpha \in P_m^{(i)} \setminus A$, we have: $\{\alpha\} \prec A$ or $A \prec \{\alpha\}$.

  Define $X = \{\alpha \in P_m^{(i)} \setminus A \mid \{\alpha\} \prec A\} \subset P_m^{(i)}$ and $Y = \{\alpha \in P_m^{(i)} \setminus A \mid A \prec \{\alpha\}\} \subset P_m^{(i)}$.
  We have: $P_m^{(i)} \setminus A = X \cup Y$, and thus $P_m^{(i)} = X \cup Y \cup A$.
  As $A \neq P_m^{(i)}$, we have: $X \neq \phi$ or $Y \neq \phi$.
  - If $X \neq \phi$ and $Y = \phi$:
    Since $\forall\, \alpha \in X$, $\{\alpha\} \prec A$, we have: $X \prec A$.
    Therefore, $P_m^{(i)} = X \cup A$ with $X \prec A$, which contradicts i).
  - If $X = \phi$ and $Y \neq \phi$:
    Since $\forall\, \alpha \in Y$, $A \prec \{\alpha\}$, we have: $A \prec Y$.
    Therefore, $P_m^{(i)} = Y \cup A$ with $A \prec Y$, which contradicts i).
  - If $X \neq \phi$ and $Y \neq \phi$:
    We have: $X \prec A$ and $A \prec Y$.
    Therefore, $X \prec (A \cup Y)$ with $P_m^{(i)} = X \cup (Y \cup A)$, which contradicts i).

  Consequently, $A = P_m^{(i)}$.
- In the same way, we prove that: $B = P_m^{(j)}$.
- Assume that: $j < i$.
  Then we have: $P_m^{(j)} \prec P_m^{(i)}$, *i.e.* $B \prec A$, which contradicts (1).
  Therefore $j \geq i$.
- Assume that: $j > i + 1$.
  Then we have: $P_m^{(i)} \prec P_m^{(i+1)} \prec P_m^{(j)}$, *i.e.* $A \prec P_m^{(i+1)} \prec B$.
  Consider $\alpha \in P_m^{(i+1)} \subset P_m \setminus (A \cup B)$.

We have $A \prec \{\alpha\} \prec B$, and thus: $\{\alpha\} \not\prec A$ and $B \not\prec \{\alpha\}$.
Thereby, $\{\alpha\} \not\prec A \cup B$ and $A \cup B \not\prec \{\alpha\}$.
Thus, $\{\alpha\} \parallel A \cup B$ with $\alpha \in P_m \setminus (A \cup B)$, which contradicts (2).
Therefore, $j \in \{i, i+1\}$.

- If $j = i$, then $A = B$, which is impossible, for $A \prec B$.
  Therefore, $j = i + 1$.
- Conclusion: $\exists\, i \in [\![1; i_m - 1]\!]$ such that $A = P_m^{(i)}$ and $B = P_m^{(i+1)}$

Consequently, $\forall\, A, B \subset P_m$ $(A, B \neq \phi)$ such that $L \vDash PT(m, A, B)$, we have:
$\exists\, i \in [\![1; i_m - 1]\!]$ such that $A = P_m^{(i)}$ and $B = P_m^{(i+1)}$.

ii) $\Rightarrow$ *i)*:
$\forall\, i \in [\![1; i_m - 1]\!]$, since $L \vDash PT(m, P_m^{(i)}, P_m^{(i+1)})$, we have:

- $P_m^{(i)} \prec P_m^{(i+1)}$
- $\forall\, X, Y \subset P_m^{(i)}$ $(X, Y \neq \phi)$ such that $X \cup Y = P_m^{(i)}$, we have $X \not\prec Y$
- $\forall\, X, Y \subset P_m^{(i+1)}$ $(X, Y \neq \phi)$ such that $X \cup Y = P_m^{(i+1)}$, we have $X \not\prec Y$

Therefore, $P_m^{(1)} \prec P_m^{(2)} \prec \cdots \prec P_m^{(i_m)}$, and
$\forall\, 1 \leq i \leq i_m$, $\forall\, X, Y \subset P_m^{(i)}$ $(X, Y \neq \phi)$ such that $X \cup Y = P_m^{(i)}$, we have $X \not\prec Y$.

□

Example 10 The set of proper timeouts satisfied by logs L_1 (*cf.* Fig. 2) and related to message b is exactly $\{\, PT(b, \{f\}, \{c, d, e\})\,,\ PT(b, \{c, d, e\}, \{g, h\})\,\}$. Indeed, the partition $\{\, \{\langle b, f\rangle\}\,,\ \{\langle b, c\rangle, \langle b, d\rangle, \langle b, e\rangle\}\,,\ \{\langle b, g\rangle, \langle b, h\rangle\}\,\}$ of P_b satisfies the conditions of Theorem 2 *i)*
$\cdot\ \{\langle b, f\rangle\} \prec \{\langle b, c\rangle, \langle b, d\rangle, \langle b, e\rangle\} \prec \{\langle b, g\rangle, \langle b, h\rangle\}$
$\cdot\ \forall\, X, Y \subset \{\langle b, c\rangle, \langle b, d\rangle, \langle b, e\rangle\}$ $(X, Y \neq \phi)$ such that $X \cup Y = \{\langle b, c\rangle, \langle b, d\rangle, \langle b, e\rangle\}$, we have $X \not\prec Y$
$\cdot\ \{\langle b, g\rangle\} \parallel \{\langle b, h\rangle\}$

This example illustrates that, given $m \in Msg$ and the partition[4] Π of P_m satisfying Theorem 2 *i)*, the set of proper timeouts satisfied by the logs and related to message m is exactly given by the set of pairs of consecutive elements in Π. In the sequel we present an algorithm for constructing such a partition in an incremental way.

4.2 Algorithm and experiments

The construction process is expressed by Algorithm 1. The input is composed of a given message name m, the set P_m of episodes whose first message is m, and the occurrence duration intervals of all episodes in P_m (derived from the logs by the global algorithm presented in the sequel). The output is the partition Π of P_m satisfying Theorem 2 *i)*. Π is constructed incrementally, starting with an empty partition, and inserting one by one the elements of P_m in such a way that Theorem 2 *i)* is satisfied at each step. This means that the sets composing Π are strictly ordered by $\prec$. In order to describe the general step of the algorithm, let us now consider that Π is already partly constructed. Let α be an episode of P_m not yet considered. A single pass is made over the partition in order to determine (i) whether the occurrence duration intervals of some elements of Π overlap the one of α, and (ii) between which sets of Π α is situated according to $\prec$ (which is done by comparing their occurrence duration intervals). If there is no overlap, a new set containing α is created and inserted into the partition in compliance with $\prec$. If the overlap takes place with only one element of Π, α is simply inserted in this set. If the overlap occurs between α and several elements of Π, they are necessarily consecutive according to $\prec$; as such they are merged and α is inserted into the resulting set. Let us highlight that once a set

[4]The considered set of proper timeouts being unique, this partition is unique too.

is created in the partition, it is never split. In fact, the algorithm is based on three simple set operations: creation, insertion and merge. As for each episode $\alpha \in P_m$ only one pass is made over the partition, the complexity of this algorithm is $\mathcal{O}(|P_m|^2)$.

Algorithm 1 *partition*P_m

Require: m, P_m, and $\forall\, \alpha \in P_m$, the Occurrence Duration Interval of α (denoted by $ODI(\alpha)$)
Ensure: Π the partition of P_m satisfying Theorem 2 i)

```
1: Π = ∅
2: while P_m ≠ ∅ do
3:    Choose episode α ∈ P_m; Remove α from P_m
4:    Overlap = ∅; B_inf = ∅; B_sup = ∅
5:          // pass made over the partition
6:    for all set S ∈ Π do
7:       if ODI(S) ∩ ODI(α) ≠ ∅ then
8:          Insert S into Overlap
9:       else if S ≺ {α} then
10:         B_inf = S
11:      else
12:         B_sup = S; goto line 15
13:      end if
14:   end for
15:         // update of the partition
16:   if Overlap = ∅ then
17:      Insert {α} into Π between B_inf and B_sup
18:   else if |Overlap| = 1 then
19:      Insert α into the set S of Π that belongs to Overlap
20:   else
21:      Merge all the sets of Π that belong to Overlap and insert α into the resulting set
22:   end if
23: end while
```

Proof *Correctness of Algorithm 1.*
The termination of the *partition*P_m algorithm is guaranteed by the finiteness of P_m. Its soundness and completeness are proven by showing that, at each step of the algorithm (*i.e.* for each new element inserted in the partition), the property stated in Theorem 2 i) is satisfied. This is obvious in the first step. In the general step, let us assume that the property was satisfied in the previous step. Then, different scenarios have to be analyzed:

- *Overlap* is empty. If $\{\alpha\}$ has to be inserted at the beginning of the partition: we find that B_{inf} is empty and that B_{sup} is the first element of Π (and $\{\alpha\}$ is inserted before B_{sup}). If it has to be at the end of the partition: we find that B_{inf} is the last element of Π and that B_{sup} is empty (and $\{\alpha\}$ is inserted after B_{inf}). Otherwise: we find that B_{inf} and B_{sup} are two consecutive elements of Π such that $B_{inf} \prec \{\alpha\} \prec B_{sup}$ (and $\{\alpha\}$ is inserted between B_{inf} and B_{sup}). Thus $\{\alpha\}$ is inserted in accordance with the order relationship between partition elements, which ensures that the first part of the property is still satisfied. Furthermore, this insertion does not change the fact that the sets of the partition cannot be decomposed. Thus the second part of the property is also still satisfied.

- *Overlap* contains only one element. Since α is inserted into this set, the order of the partition elements is preserved. Since the occurrence duration intervals of this set and of α are not disjoined, this insertion does not enable to decompose this set. The others sets of the partition remain undecomposable.

- *Overlap* contains several elements. Let us denote them by $E_1, E_2, \ldots, E_k$, and assume without loss of generality that $E_1 \prec E_2 \prec \cdots \prec E_k$. Let us denote $E_1 \cup E_2 \cup \cdots \cup E_k \cup \{\alpha\}$

by E. We have $B_{inf} \prec \{\alpha\} \prec B_{sup}$, and $\forall 1 \leq i \leq k\,, ODI(E_i) \cap ODI(\alpha) \neq \emptyset$. Thus, in the partition, we have $\cdots \prec B_{inf} \prec E_1 \prec E_2 \prec \cdots \prec E_k \prec B_{sup} \prec \ldots$. Therefore, $\cdots \prec B_{inf} \prec E \prec B_{sup} \prec \ldots$. Furthermore, since $\forall 1 \leq i \leq k\,, ODI(E_i) \cap ODI(\alpha) \neq \emptyset$, E cannot be decomposed. The other sets of the partition remain undecomposable.

□

Example 11 Given logs L_1 (c.f. Fig. 2), algorithm *partition*P_m constructs the partition Π of $P_a = \{\langle a, c\rangle, \langle a, h\rangle, \langle a, e\rangle, \langle a, g\rangle, \langle a, d\rangle\}$, by including the episodes one by one (the final result being independent of the insertions order):

- $\langle a, c\rangle$: in the first stage, the first set is created; $\Pi = \{\{\langle a, c\rangle\}\}$
- $\langle a, h\rangle$: since $\{\langle a, c\rangle\} \prec \{\langle a, h\rangle\}$ (i.e. there is no overlap), $\langle a, h\rangle$ is inserted in a new element of the partition after $\{\langle a, c\rangle\}$; $\Pi = \{\{\langle a, c\rangle\}, \{\langle a, h\rangle\}\}$
- $\langle a, e\rangle$: as $\{\langle a, c\rangle\} \prec \{\langle a, e\rangle\}$ and $\{\langle a, e\rangle\} \prec \{\langle a, h\rangle\}$, $\langle a, e\rangle$ is inserted in a new part between $\{\langle a, c\rangle\}$ and $\{\langle a, h\rangle\}$; $\Pi = \{\{\langle a, c\rangle\}, \{\langle a, e\rangle\}, \{\langle a, h\rangle\}\}$
- $\langle a, g\rangle$: since $\{\langle a, h\rangle\} \prec \{\langle a, g\rangle\}$, $\langle a, g\rangle$ is inserted in a new set of the partition after $\{\langle a, h\rangle\}$; $\Pi = \{\{\langle a, c\rangle\}, \{\langle a, e\rangle\}, \{\langle a, h\rangle\}, \{\langle a, g\rangle\}\}$
- $\langle a, d\rangle$: as $\{\langle a, d\rangle\} \parallel \{\langle a, c\rangle\}$ and $\{\langle a, d\rangle\} \parallel \{\langle a, e\rangle\}$ (i.e. they overlap), $\{\langle a, c\rangle\}$ and $\{\langle a, e\rangle\}$ are merged; $\Pi = \{\{\langle a, c\rangle, \langle a, e\rangle, \langle a, d\rangle\}, \{\langle a, h\rangle\}, \{\langle a, g\rangle\}\}$.

The global method for extracting all the proper timeouts satisfied by the logs is divided in two steps. The first one is a preprocessing of the data, performed in order to compute the set of message names *Msg*, the set of episodes *Ep*, and the occurrence duration intervals of all episodes in *Ep*. A single pass is made over the logs, during which the occurrence duration of each sequence of two consecutive messages is calculated.[5] The second step consists in constructing the partition $\{P_m \,|\, m \in Msg\}$ of *Ep*, and running algorithm *partition*P_m for each $m \in Msg$. The logs' size being far greater than the number of episodes, the first step will be the most costly. Thus the complexity of the global algorithm is $\mathcal{O}(|L|)$.

Example 12 The proper timeouts satisfied by logs L_1 (c.f. Fig. 2), and extracted by the global method, are listed in Table 1.

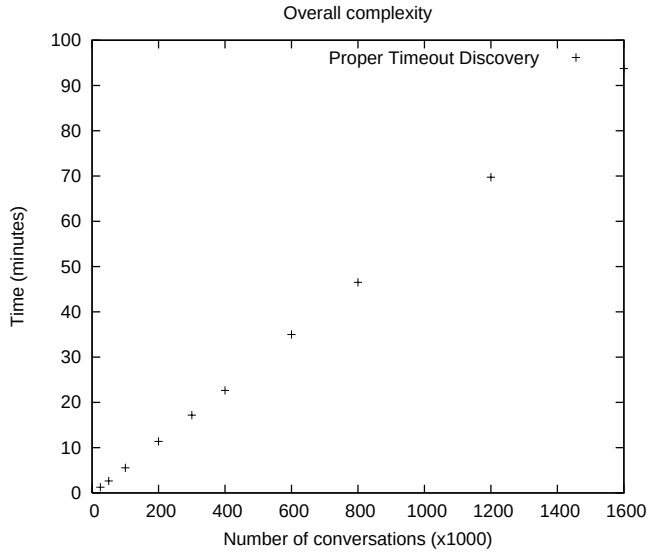
Table 1: Proper timeouts satisfied by logs L_1.

	proper timeout	expiry interval
P_a	$PT(a, \{c, d, e\}, \{h\})$	$]6; 8[$
	$PT(a, \{h\}, \{g\})$	$]10; 15[$
P_b	$PT(b, \{f\}, \{c, d, e\})$	$]3; 6[$
	$PT(b, \{c, d, e\}, \{g, h\})$	$]10; 13[$

Experiments. We implemented our discovery process to test its scalability. Tests were performed on a computer with an Intel Pentium 4, 2.8 GHz processor, 2 GB of RAM, running Microsoft Windows XP Professional Edition SP2. In order to easily have a big amount of data, we implemented a log generator which creates conversation logs from a given business protocol by mimicking the behavior of a service. The generator was designed using the Matlab

[5] In other words, we use an analysis window of length 2 over the data.

Simulink tool from MathWorks. A protocol is modelled as a finite-state machine (FSM) and timed transitions are materialized by logic constraints on the FSM transitions from one state to another. This generator can be of a more general interest in the field of WS simulation since it features a human-friendly user interface for designing protocols to be run, as well as the possibility to introduce execution constraints and a random-based noise factor which should give more realistic logs. The implementation of the discovery algorithm is written in Java (version $1.5.0-06$) using the Eclipse development environment and the $java\ -Xms512m\ -Xmx1024m$ instruction for defining the heap size of the JVM. The maximum time for each test was set to 19 hours. Results of our experiments are presented in Fig. 5. They confirm the complexity results we established formally: (i) partition complexity is quadratic wrt the number of episodes, (ii) partition time is negligible compared to preprocessing time, and (iii) global discovery (i.e. log preprocessing and partition) complexity is linear wrt the logs' size. Considering the logs' size, running times are reasonable enough to affirm that our method is scalable. The final test will be to run our algorithm on real-life data in further experiments.



Number of episodes	Running time of partition process (in seconds)
2	0.71
4	12.5
6	27.3
8	49.4

Figure 5: Running times of overall discovery method (*left*) and partition process (*right*).

5 Discussion and state of the art

As illustrated previously, elements of the partition $\{P_m \,|\, m \in Msg\}$ of Ep are treated separately. Thus, with each part P_m will be associated a set of discovered proper timeouts. This process can seem redundant, in a sense that, if two transitions labelled for example by messages a and b enter in the same state, from which a timed transition is going out, then two different proper timeouts will be satisfied by the logs (one for P_a and another for P_b), and interpreted as representing two different potential timed transitions. In fact, it is possible to realize that there is only one, by using cross-checking between all sets of proper timeouts (*cf.* Ex. 13), which can be done automatically.

This cross-checking can also enable to reject some proper timeouts which cannot represent timed transitions (*cf.* Ex. 13). Indeed, as already explained, a proper timeout is satisfied if, in some state of the service, some messages take longer to be sent or received than others. However, cross-checking can be inefficient, if there is no contradicting proper timeout, to reject a "fake" timed transition. In such a case, only a domain expert having some knowledge about the service can make a decision. More generally, the verification of the discovered proper timeouts by an expert could always increase the confidence about the result.

Example 13 Consider the proper timeouts satisfied by logs L_1 (*cf.* Table 1). $PT(a, \{h\}, \{g\})$ is rejected because it cannot represent a timed transition. Indeed, $L_1 \vDash PT(b, \{c, d, e\}, \{g, h\})$ implies that $\{\langle b, h\rangle\} \parallel \{\langle b, g\rangle\}$; thus, h and g label two transitions going out of the same state (and g takes longer to be emitted than h, only after a has been emitted). Regarding the "redundancy" problem, we can see that $PT(a, \{c, d, e\}, \{h\})$ and $PT(b, \{c, d, e\}, \{g, h\})$ represent the same timed transition, because there is a correspondence between the involved sets of messages. Finally, the result consists only on two timed transitions: one corresponding to $PT(b, \{f\}, \{c, d, e\})$, and the other corresponding to $PT(a, \{c, d, e\}, \{h\})$ and $PT(b, \{c, d, e\}, \{g, h\})$.

Recall that we assumed to have complete logs. First, it can be seen as a very strong assumption, but it is not. In fact, we ask only for *valid* conversations (*i.e.* the ones which finish in a final state), and not for all paths of the business protocol, to be separately represented in the logs. Furthermore, services generally do not allow a lot of operations, and complex services are mainly constructed by composing more simple ones. As such, for a given service, the set of valid conversations is quite small. Second, logs completeness is mainly a theoretical assumption, made in order to prove that our algorithm can discover all timed transitions in this case. In fact, we do not ask for logs to be complete in real life scenarios. Thus, some timed transitions can be missed. In a re-engineering point of view, this can mean that some operations are useless, and justify an evolution of the service.

Related work.

The business protocol discovery problem addressed in [15, 16] can be considered as a particular case of a more general issue: the extraction of a model from its instances. Literature related to model discovery is extensive, for example in grammatical inference [17, 3], in workflow mining [8, 1, 20, 13, 19], or in Web services interaction mining [10].

In grammatical inference, the problem consists in finding a grammar generating a language, given a set of words that belong to this language, and a set of words that do not belong to it; using both positive and negative examples allows producing a correct model. The business protocol discovery problem is different in the sense that only (noisy and incomplete) positive instances are available.

In Web services interaction mining, the goal is to discover from exchanged logs a workflow modelling the interactions that take place between several services. Despite the similar context, this differs from business protocol discovery in its knowledge extraction level; in [15, 16] cross-services protocols are not considered.

Workflow mining (or software process discovery) is very similar to business protocol discovery; actually the work in [8] strongly inspired the discovery method presented in [15, 16]. Main differences between all techniques lie in the choice of the model (automaton [15, 16, 8], Petri net [20] or directed graph [1, 13, 19]), in the fact that noise is considered [15, 16, 8, 1, 13, 19] or not [20], and that the extraction process allows user driven refinement [15, 16] or not [8, 1, 20, 13, 19]. It is worth to notice that none of these approaches consider the extraction of temporal constraints from the data.

At the same time the work exposed in [15, 16] was achieved, similar techniques [9] were developed, though with quite different concerns. In [9] the concepts of *interaction* and *interaction protocol* are used equivalently to *conversation* and *business protocol*. The goal is to solve an interoperability problem between a client and several Web services that expose the same interface but may have different interaction protocols. The proposed solution consists in automatically extracting approximated interaction protocols from recorded interactions (between these services and previous users); the client can then verify which services are compatible with its requirements and adapt its own sequences of interactions. This method is presented as a simple extension of a service discovery architecture. As in [15, 16] the model discovery algorithm is borrowed from the workflow mining area. The main advantage of the approach exposed in [9]

is that it is fully automatic, which is essential in a service discovery architecture. However, this can be an important drawback in the service management and re-engineering area, where it is essential to allow a user driven refinement of the extracted model [15, 16]. Two other limitations of [9] are that (i) the modules performing the protocol discovery are supposed to be associated by service brokers with all new published services (which raises many questions about a possible implementation), and that (ii) noise in mined interactions is not taken into account, contrary to [15, 16]. Finally, it is worth to note that our work could be an extension of [9] too, as possible temporal constraints of the interaction protocol are not considered.

Since we are not interested in discovering the whole business protocol but only some particular transitions, our work is more related to the area of pattern mining. Among the numerous kinds of patterns that have been studied in the literature, we can mention for example sequential patterns [2, 18, 14], workflow patterns [11], or service interaction patterns [4].

Sequential patterns [2, 18] are basically event subsequences; in [14], they are named *episodes* and defined as directed acyclic graphs of events. They are extracted from a sequence database [2, 18] or from event sequences [14], using levelwise [2, 14] or pattern-growth [18] methods. As in many data mining problems, only frequent patterns are sought, contrary to our approach. It is also worth to notice that, even if they provide some information about the order between events, sequential patterns do not contain explicit temporal constraints, contrary to proper timeouts.

In the area of workflow mining, not all the literature relates to model discovery. In [11] workflow patterns are defined as substructures of the directed graph representing the workflow, and are extracted from execution logs. Contrary to our approach, the model is supposed to be known for the generation of the patterns, and only the frequent ones are considered. In fact, the method presented in [11] does not aim to discover any knowledge about the workflow, but to perform diagnostic and termination prediction. As sequential patterns, workflow patterns do not contain explicit temporal constraints.

In the Web services field, service interaction patterns [4] are defined as abstracted forms of representative scenarios. They are not extracted from execution data, but empirically derived from the literature, standardization activities, and real-scale use cases. They are mainly used to benchmark Web services functionalities related to choreography and orchestration.

6 Conclusion

In the context of the discovery of the timed business protocol of a Web service from its conversation logs, we have focused on extracting timed transitions. Our contribution is based on a formal framework leading to the definition of proper timeouts. We have shown that proper timeouts are the best representations of timed transitions in conversation logs. We have given a simple characterization of the set of proper timeouts satisfied by the logs. We have also proposed a polynomial algorithm for extracting these patterns.

As future work, our first objective is to broaden our method: we plan to deal with more general business protocols, with cycles, and where transitions are not necessarily uniquely labelled, or having timed transitions entering in a final state. It would be also relevant to analyze noisy logs and to propose a probabilistic method. Another interesting prospect would be to make use of our technique to discover the entire business protocol. In fact, the result we propose to a user contains not only proper timeouts but also some local knowledge about transitions. We would like to investigate whether gathering this local information could lead to a coherent global knowledge about the protocol.

As already mentioned, this work is an extension of the one proposed in [15, 16], and combining both approaches is an exciting prospect. In fact, both take part in *ServiceMosaic*[6] international project, which aims at developing a platform for modeling, analysing and managing Web ser-

[6] http://servicemosaic.isima.fr

vices. The main goals of this project are (i) defining models enabling description, orchestration and composition of services, (ii) specifying an algebra for high-level analysis, and (iii) creating service management and development tools.

Acknowledgments

The authors would like to thank Boualem Benatallah and Hamid Motahari, whose comments and discussions considerably improved the quality of the paper.

References

[1] Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. Mining process models from workflow logs. In *EDBT '98*, pages 469–483, Valencia, Spain, Mar 1998. Springer.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *ICDE '95*, pages 3–14, Taipei, Taiwan, Mar 1995. IEEE Computer Society.

[3] Dana Angluin and Carl H. Smith. Inductive inference: Theory and methods. *ACM Computing Surveys*, 15(3):237–269, Sep 1983.

[4] Alistair Barros, Marlon Dumas, and Arthur H. M. ter Hofstede. Service interaction patterns. In *Business Process Management 2005*, pages 302–318, Nancy, France, Sep 2005. Springer.

[5] Boualem Benatallah, Fabio Casati, Julien Ponge, and Farouk Toumani. Compatibility and replaceability analysis for timed web service protocols. In *BDA '05*, Saint-Malo, France, Oct 2005.

[6] Boualem Benatallah, Fabio Casati, Julien Ponge, and Farouk Toumani. On temporal abstractions of web services protocols. In *CAiSE '05 Short Paper Proceedings*, pages 39–44, Porto, Portugal, June 2005. Springer.

[7] Boualem Benatallah, Fabio Casati, and Farouk Toumani. Analysis and management of web service protocols. In *Conceptual Modeling - ER '04*, pages 524–541, Shanghai, China, Nov 2004. Springer.

[8] Jonathan E. Cook and Alexander L. Wolf. Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, July 1998.

[9] Giovanni Denaro, Mauro Pezzé, Davide Tosi, and Daniela Schilling. Towards self-adaptive service-oriented architectures. In *TAV-WEB '06*, pages 10–16, Portland, Maine, USA, Jul 2006. ACM.

[10] Schahram Dustdar, Robert Gombotz, and Karim Baïna. Web services interaction mining. Technical Report TUV-1841-2004-16, Technical University of Vienna, Vienna, Austria, Sep 2004.

[11] Gianluigi Greco, Antonella Guzzo, Giuseppe Manco, and Domenico Saccà. Mining and reasoning on workflows. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):519–534, Apr 2005.

[12] Rachid Hamadi and Boualem Benatallah. A petri net-based model for web service composition. In *ADC '03*, pages 191–200, Adelaide, Australia, Feb 2003. Australian Computer Society, Inc.

[13] San-Yih Hwang and Wan-Shiou Yang. On the discovery of process models from their instances. *Decision Support Systems*, 34(1):41–57, Dec 2002.

[14] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, Sep 1997.

[15] Hamid R. Motahari-Nezhad, Régis Saint-Paul, Boualem Benatallah, and Fabio Casati. Protocol discovery from imperfect service interaction logs. In *ICDE '07*, pages 1405–1409, Istanbul, Turkey, Apr 2007. IEEE.

[16] Hamid R. Motahari-Nezhad, Régis Saint-Paul, Boualem Benatallah, Fabio Casati, Julien Ponge, and Farouk Toumani. Servicemosaic: Interactive analysis and manipulations of service conversations. In *ICDE '07*, Istanbul, Turkey, Apr 2007. IEEE. Demonstration.

[17] Rajesh Parekh and Vasant Honavar. Grammar inference, automata induction, and language acquisition. In *Handbook of natural language processing*, pages 727–764. Marcel Dekker, Inc., New York, USA, 2000.

[18] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Jianyong Wang, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1424–1440, Nov 2004.

[19] Ricardo Silva, Jiji Zhang, and James G. Shanahan. Probabilistic workflow mining. In *KDD '05*, pages 275–284, Chicago, Illinois, USA, Aug 2005. ACM.

[20] Wil van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, Sep 2004.