Fine-grained Compatibility and Replaceability Analysis of Timed Web Service Protocols (extended version)

Julien Ponge[1,2], Boualem Benatallah[2], Fabio Casati[3], and Farouk Toumani[1]

[1] {ponge,ftoumani}@isima.fr Univ. Clermont-Ferrand 2, France
[2] boualem@cse.unsw.edu.au UNSW, Sydney, Australia
[3] casati@dit.unitn.it Univ. of Trento, Italy

Notice. This is an extended version of the paper that appeared in the proceedings of the *26th International Conference on Conceptual Modeling (ER 2007)*, held in november 2007, Auckland, New-Zealand. The original article is left intact with the extension having been added in the appendix. The original article portion is © 2007 Springer-Verlag.

Abstract. We deal with the problem of automated analysis of web service protocol compatibility and replaceability in presence of timing abstractions. We first present a timed protocol model for services and identify different levels of compatibility and replaceability that are useful to support service development and evolution. Next, we present operators that can perform such analysis. Finally, we present operators properties by showing that timed protocols form a new class of timed automata, and we briefly present our implementation.

1 Introduction

Service-oriented architectures (SOAs) and web service technologies are emerging computing paradigm for the development and integration of distributed applications [1]. They are based on the notion of *services*, which are loosely-coupled applications interfaces accessible via a programmatic API relying on open standards (e.g., XML, HTTP or SOAP). The idea behind loose coupling is that services can be made generally accessible to a community of users and clients, as opposed to being specifically developed for certain clients, as it was the case in conventional, CORBA-style integration where clients and services were often developed concurrently and by the same team. This capability comes at a price: the need of providing fairly detailed service descriptions, so that (i) at design time, developers know how to write applications that can correctly interact with the service, and (ii) at deployment or run time, it is possible to identify if a client can correctly interact with a service.

Today, service descriptions typically include the interface definition, the transport-level properties (both specified in WSDL), and *business protocol* definitions, that is, the specification of possible message exchange sequences (conversations) that are supported by the service [2]. Protocols can be specified using WS-BPEL (*Web Services Business Process Execution Language*) or any of the many other formalisms developed for this purpose (e.g., [2,3]). Providing such descriptions only solves part of the problem. To facilitate service development and interoperability there is the need for formal methods and software tools that allow the automated analysis of service descriptions to (i) identify which conversations can be carried out between two services, understand mismatches between protocols and, if possible, create adapters to allow interactions between incompatible services (called *compatibility analysis*), and (ii) manage service evolution, that is, understand if a new version of a service protocol is compatible with the intended clients (called *replaceability analysis*).

Such a need is widely recognized and many approaches have been developed, including some by the authors. In particular, in our previous work we developed a simple but expressive business protocol model based on state machines, an algebra for business protocol analysis, and a set of operators to compare and manipulate protocols and that form the basis for compatibility and replaceability analysis [4]. The operators have been implemented within *ServiceMosaic* [5], a CASE tool environment that enables the model-based design, development and management of Web services.

While previous approaches provide significant contributions to protocol analysis (and, in general, to service specification analysis), little work has been done in the context of *timed protocols*, that is, protocols that include time-related properties. This limitation is significant: time is an essential ingredient of any real-life protocol specification. There are countless examples of behaviors that involve timing issues in any kind of protocol [2], from business protocol for web services (e.g., see the RosettaNet PIPs), to interactions between traditional web-based services and users (see e-commerce web sites such as Travelocity or Amazon), to lower level protocols such as TCP. Time-related behaviors range from session timeouts to "logical" deadlines with different kinds of behaviors (e.g., seats reserved on a flight needs to be paid within n hours otherwise they are released). In [2], we have identified extensions to protocol models suitable for representing timing aspects. The extensions are based on an analysis of existing protocols so that we could identify a modeling framework that is simple but expressive. More specifically, we identified the need for representing two kinds of temporal constraints in protocol descriptions: (i) time intervals during which an operation can be invoked and (ii) deadline expirations. Such kinds of constraints can also model timing properties of languages such as WS-BPEL and RosettaNet. The introduction of time aspects adds significant complexity to the protocol analysis problem. Indeed, many formal models enabling explicit representation of time exist (e.g., timed automata, timed petri-nets), all showing extreme difficulties to handle algorithmic analysis of timed models. For example, timed automata, which are today considered as a standard modeling formalism to deal with timing constraints, suffer from undecidability of many problems such as language inclusion and complementation that are fundamental to system analysis and verification tasks [6]. Such problems have been shown to be very sensitive to several criteria (e.g., density of the time axis, type of constraints, presence of silent transitions) This paper extends our previous work in the following directions and makes the following contributions:

1. We formally define *timed protocols*, an extension of business protocols that is suitable to represent both time intervals and deadline expirations constraints.
2. We define a framework for timed protocol analysis, introducing fine-grained classes to study different degrees of compatibility and replaceability among protocols.
3. We define an algebra for protocol analysis and management by defining operators that can manipulate and analyze timed protocols and that can be used to characterize the various compatibility and replaceability classes. We see this work as being inspired, at least conceptually, by work done over the last 30 years in databases, leading to generic abstraction techniques such as relational algebras that eventually generated the widespread adoption of the relational model. We argue that an algebra for protocol analysis can bring to service-oriented computing similar benefits to what relational algebra brought to relational databases.
4. We establish a semantic-preserving mapping from timed protocols to a new class of timed automata [7] with a restricted form of ε transitions (i.e., "unobservable" or silent transitions). Based on this mapping, we reuse and extend existing results in timed automata theory to derive decidability results for our timed protocol operators. The obtained result is interesting by itself because timed protocols lead to an innovative class of timed automata that includes ε transitions that strictly increase the expressiveness of the automata (i.e., they cannot be removed without a loss of expressiveness) and despite this fact, this class still exhibits a *deterministic behavior*. Especially, the *complementation problem* is decidable for this class. To the best of our knowledge, this is the first identified class of timed automata displaying such a feature.

Due to a lack of space, proofs and additional technical details regarding this work are omitted from this paper but are given in [8] which contains them in its appendix.

2 Timed Protocol Modeling

This section introduces first informally and then formally the model of *timed business protocols* which extends *business protocols* [4] with timing-related abstractions.

2.1 Extending Business Protocols with Temporal Abstractions

We built our model upon the traditional state-machine formalism, which is commonly used to model protocols and, more generally, to model the external behaviors of systems, due to the fact that they are simple and intuitive. In the model, states represent the different phases that a service may go through during its interaction with a requester. Transitions can be associated with a message and/or a constraint. Transitions associated with a message must also indicate the message *polarity*, that denotes whether the message is incoming (plus sign) or outgoing (minus sign). They are triggered when the associated message is sent (or received, depending on the polarity). A message corresponds to the invocation of a service operation or to its reply. Hence, each state identifies a set of outgoing transitions, and therefore a set of possible messages that can be sent or received when the conversation with a client is in that state. For instance, the protocol depicted in Figure 1, inspired from the *Ford Credit web portal*, specifies that a financing service is initially in the *Start* state, and that clients begin using the service by sending a login message, upon which the service moves to the *Logged* state (transition ($login(+)$). In the figure, the initial state is indicated by an unlabeled entering arrow without source while final (accepting) states are double-circled. Furthermore, the figure shows that the sequence of message $login(+) \cdot selectVehicle(+) \cdot estimatePayment(+)$ is a conversation supported by the protocol, while the conversation $fullCredit(+) \cdot selectVehicle(+)$ is not. By defining constraints on the ordering of the messages that a web service accepts, a protocol makes explicit to clients how they can *correctly* interact with a service without generating errors due to incorrect sequencing of messages.

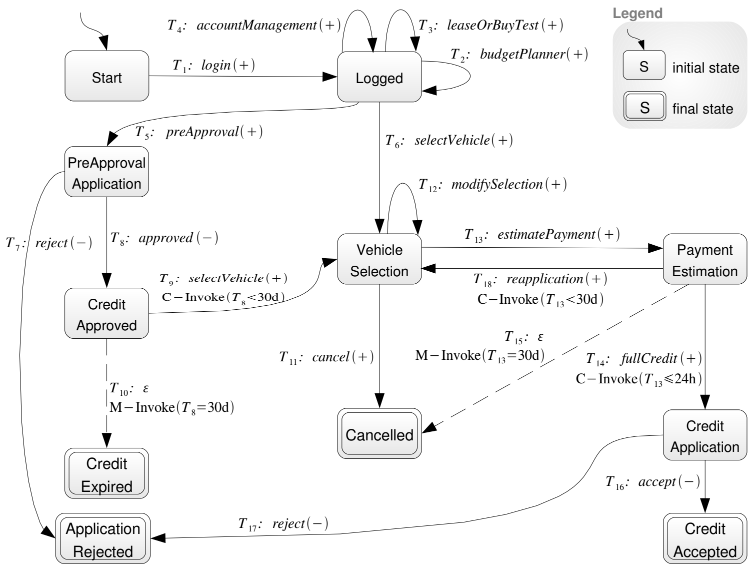


Fig. 1. A timed protocol of an online financing services.

Constraints can also be associated to transitions. In this paper we focus on timing abstractions, as we have identified two kinds of constraints that are often needed in practice:

– **C-Invoke** constraints specify a time window within which a given transition can be fired. Outside the window, the transition is disabled (exchanging the message results in an error).
– **M-Invoke** constraints specify when a transition is automatically fired.

M-Invoke constraints can only be associated with *implicit* (as opposed to explicit) transitions, which are used to model transitions that can occur without an explicit invocation by requesters. Implicit transitions are analogous to the so-called silent or ε transitions in automata theory [9]. We assume that implicit transitions are associated with an empty message noted ε.

We use the term *timed protocol* to denote a business protocol whose definition contains such temporal abstractions. Timed protocols must be deterministic, as the client always needs to be able to determine in which state the service is, else much of the purpose of the protocol specification is lost.

Continuing with the example, the financing service may need to specify that a full credit application is accepted only if it is received 24 hours after a payment estimation has been made. This behavior is specified by tagging the transition $T_{14} : \mathit{fullCredit}(+)$ with a time constraint C-Invoke$(T_{13} \leq 24h)$. This constraint indicates that transition T_{14} can only be fired within a time window $[0h, 24h]$ after the execution of the transition T_{13}. The implicit transition T_{10}, depicted in the figure using a dotted arrow, is associated with constraint M-Invoke$(T_8 = 30d)$ to specify that once a pre-approval application is approved (i.e., after transition $T_8 : \mathit{approved}(-)$ is fired which makes a service entering state $\mathit{CreditApproved}$, a client has 30 days to use the credit, after that the credit decision expires. Note that, the presence of an implicit transition at a given state affects the timing constraints of the other explicit transitions that can be fired from this state. In our example, the presence of the implicit transition T_{10} implies that transition $T_9 : \mathit{selectVehicle}(+)$ can only be executed within a time window $[0d, 30d]$ after the service has entered state $\mathit{CreditApproved}$.

2.2 Formalization

To formally define timed protocols, we first introduce the types of constraints used in this paper. Let $\mathcal{X}$ be a set of variables referring to transition identifiers, i.e., if r is a transition then $x_r \in \mathcal{X}$ is variable referring to this transition. We consider the following two kinds of time constraints defined over a set of variables $\mathcal{X}$:

– C-Invoke(c) with c defined as follows: $c ::= x \; op \; k \mid c \wedge c \mid c \vee c$ with $op \in \{=, \neq, <, >, \leq, \geq\}$, $x \in \mathcal{X}$ and $k \in \mathbb{Q}^{\geq 0}$, where $\mathbb{Q}^{\geq 0}$ denotes the set of nonnegative rational numbers.
– M-Invoke(c) is also defined with $x \in \mathcal{X}$ and $k \in \mathbb{Q}^{\geq 0}$ as above but with the restriction that $c ::= x = k \;\; [\wedge(x \neq k \mid c \wedge c)]$ (it is an equality with an optional conjunction of equalities and inequalities).

We can now introduce a formal definition of timed protocols.

Syntax.

- $\mathcal{S}$ is a finite set of states, with $s_0 \in \mathcal{S}$ being the initial state.
- $\mathcal{F} \subseteq \mathcal{S}$ is the set of final states. If $\mathcal{F} = \emptyset$, then $\mathcal{P}$ is said to be an empty protocol.
- $\mathtt{M} = \mathtt{M_e} \cup \{\varepsilon\}$ is a finite set of messages $\mathtt{M_e}$ augmented with the empty message ε. For each message $m \in \mathtt{M_e}$, we define a function Polarity$(\mathcal{P}, m)$ which will be positive $(+)$ if m is an input message in $\mathcal{P}$, and negative $(-)$ if m is an output message in $\mathcal{P}$.
- $\mathcal{X} = \{x_r \mid \exists r \in \mathcal{R}\}$ is a set of variables defined over the set of transitions $\mathcal{R}$.
- $\mathcal{C}$ is a set of time constraints defined over a set of variables $\mathcal{X}$. The absence of a constraint is interpreted as a constraint with the value of `true`.
- $\mathcal{R} \subseteq \mathcal{S}^2 \times \mathtt{M} \times \mathcal{C}$ is a finite set of transitions. Each transition (s, s', m, c) identifies a source state s, a target state s', a message m and a constraint c. We say that the message m is enabled from a state s. When $m = \varepsilon$, c must be a M-Invoke constraint. Otherwise c must be either a C-Invoke constraint or `true`.

In the sequel, we use the notation $\mathcal{R}(s, s', m, c)$ to denote the fact that $(s, s', m, c) \in \mathcal{R}$. To enforce determinism, we require that a protocol has only one initial state, and that for every state s and every two transitions (s, s_1, m_1, c_1) and (s, s_2, m_2, c_2) enabled from s, we have either $m_1 \neq m_2$ or $c_1 \wedge c_2 \equiv \mathtt{false}$. To enforce preemption of M-Invoke constraints over C-Invoke constraints, it is assumed that for each state that offers implicit transitions, the explicit transitions satisfy C-Invoke constraints as follows. They must satisfy the conjunction of all the $T_i < k_i$ constraints where $T_i = k_i$ appears in a M-Invoke constraint. Otherwise, an explicit transition could still be fired after all the implicit transitions have expired. Finally, we do not allow cycles only made of implicit transitions as the system would enter an infinite loop.

Variable interpretation. To formally define the semantics of timed protocols we introduce the notion of variable valuation. We consider as a time domain the set of non-negative reals $\mathbb{R}^{\geq 0}$. Let $\mathcal{X}$ be a set of variables with values in $\mathbb{R}^{\geq 0}$. A (variable) valuation $\mathcal{V} : \mathcal{X} \rightarrow \mathbb{R}^{\geq 0}$ is a mappings that assigns to each variable $x \in \mathcal{X}$ a time value $\mathcal{V}(x)$. We note by $\mathcal{V}_t$ the variable valuation at an instant t. At the beginning (i.e., instant $t_0 = 0$) we assume that all the variables are set to zero, i.e., $\mathcal{V}_{t_0}(x_r) = 0, \forall x_r \in \mathcal{X}$. Then, a variable valuation at a time t_j, is completely determined by a protocol execution. Consider for example an execution $\sigma = s_0 \cdot (m_0, t_0) \cdot s_1 \ldots s_{n-1}.(m_{n-1}, t_{n-1}) \cdot s_n$ of a protocol $\mathcal{P}$ and let r be a transition in $\mathcal{R}$. The valuation of a variable x_r at time t_j, with $0 < j \leq n$, is defined as follows:

$$\mathcal{V}_{t_j}(x_r) = \begin{cases} 0, \text{ if } r = (s_{j-1}, s_j, m_{j-1}, c_{j-1}) \\ \mathcal{V}_{t_{j-1}}(x_r) + t_j - t_{j-1}, \text{ otherwise} \end{cases}$$

Given a variable valuation $\mathcal{V}$ and a constraint C-Invoke(c) (respectively, M-Invoke(c)), we note by $c(\mathcal{V})$, the constraint obtained by substituting each variable x in c by its value $\mathcal{V}(x)$. A variable valuation $\mathcal{V}$ satisfies a constraint C-Invoke(c) (respectively, M-Invoke(c)) iff $c(\mathcal{V}) \equiv \mathtt{true}$. In this case, we write $\mathcal{V} \models$ C-Invoke(c) (respectively, $\mathcal{V} \models$ M-Invoke(c)).

Protocol semantics. We define the semantics of timed protocols using the notion of *timed conversation* (this is inspired from *timed words* in [7]).

Let $\mathcal{P} = (\mathcal{S}, s_0, \mathcal{F}, \mathtt{M}, \mathcal{R}, \mathcal{C})$ be a timed protocol. A correct execution (or simply, an execution) of $\mathcal{P}$ is a sequence $\sigma = s_0 \cdot (m_0, t_0) \cdot s_1 \ldots s_{n-1} \cdot (m_{n-1}, t_{n-1}) \cdot s_n$ such that: (i) $t_0 \leq t_1 \leq \ldots \leq t_n$ (i.e., the occurrence of times increase monotonically), (ii) s_0 is the initial state and s_n is a final state of $\mathcal{P}$, and (iii) $\forall j \in [1, n]$, we have: $\mathcal{R}(s_{j-1}, s_j, m_{j-1}, c_{j-1})$ and $\mathcal{V}_{j-1} \models c_{j-1}$.

As an example, the sequence $\sigma' = \mathit{Start}\cdot(\mathit{login}(+), 0)\cdot\mathit{Logged}\cdot(\mathit{preApproval}(+), 1)\cdot \mathit{PreApprovalApplication}\cdot(\mathit{approved}(-), 3)\cdot\mathit{CreditApproved}\cdot(\varepsilon, 33)\cdot\mathit{CreditExpired}$ is a correct execution of the financing service protocol depicted at figure 1. If $\sigma = s_0 \cdot (m_0, t_0) \cdot s_1 \ldots s_{n-1} \cdot (m_{n-1}, t_{n-1}) \cdot s_n$ is a correct execution of protocol $\mathcal{P}$, then the sequence $tr(\sigma) = (m_0, t_0) \ldots (m_{n-1}, t_{n-1})$ forms a timed trace which is compliant with $\mathcal{P}$. Continuing with the example, the execution σ' of the financing service protocol leads to the timed trace $tr(\sigma') = (\mathit{login}(+), 0) \cdot (\mathit{preApproval}(+), 1) \cdot (\mathit{approved}(-), 3) \cdot (\varepsilon, 33)$. During an execution σ of a protocol $\mathcal{P}$, the externally timed observable behavior of $\mathcal{P}$, hereafter called *timed conversation* of $\mathcal{P}$ and noted $\mathit{conv}(\sigma)$, is obtained by removing from the corresponding timed trace $tr(\sigma)$ all the non observable events (i.e., all the pairs (m_i, t_i) with $m_i = \varepsilon$). For example, during the previous execution σ', the observable behavior of the financing service is described by the timed conversation $\mathit{conv}(\sigma') = (\mathit{login}(+), 0) \cdot (\mathit{preApproval}(+), 1) \cdot (\mathit{approved}(-), 3)$. In the following, given a protocol $\mathcal{P}$, we denote by $Tr(\mathcal{P})$ the (possibly infinite) set of timed conversations of (or compliant with) $\mathcal{P}$.

Protocol interaction semantics. Timed conversations describe the externally observable behavior of timed protocols and, as it will be shown below, are essential to analyze the ability of two services to interact correctly. Let us consider the protocol $\mathcal{P}$ depicted on Figure 1 and its reversed protocol $\mathcal{P}'$ obtained from $\mathcal{P}$ by reversing the polarity of the messages (i.e., input messages becomes outputs and vice versa). We can observe that when $\mathcal{P}'$ interacts with $\mathcal{P}$ following a given timed conversation τ, $\mathcal{P}$ follows exactly a similar conversation but with reversed polarities on the messages. If during such an interaction the timed conversation of $\mathcal{P}'$ is $(\mathit{login}(+), 0) \cdot (\mathit{selectVehicle}(+), 1) \cdot (\mathit{estimatePayment}(+), 10) \cdot (\mathit{fullCredit}(+), 30) \cdot (\mathit{accept}(-), 100)$, then the timed conversation of $\mathcal{P}'$ will be $(\mathit{login}(-), 0) \cdot (\mathit{selectVehicle}(-), 1) \cdot (\mathit{estimatePayment}(-), 10) \cdot (\mathit{fullCredit}(-), 30) \cdot (\mathit{accept}(+), 100)$. In this case, we call the path $(\mathit{login}, 0) \cdot (\mathit{selectVehicle}, 1) \cdot (\mathit{estimatePayment}, 10) \cdot (\mathit{fullCredit}, 30) \cdot (\mathit{accept}, 100)$ a *timed interaction trace* of $\mathcal{P}$ and $\mathcal{P}'$. Please note that the polarity of the messages that appear in interaction traces is not defined, as in such traces each input message m of one protocol coincides with an output message m of the other protocol. More precisely, let $\mathcal{P}$ and $\mathcal{P}'$ be two timed protocols and let $\tau = (a_0, t_0), \ldots (a_n, t_n)$ be a sequence of events for which the messages polarities are not defined. Then τ is a timed interaction trace of $\mathcal{P}$ and $\mathcal{P}'$ if and only if there exist two timed conversation σ_1 and σ_2 such that: (i) $\sigma_1 \in Tr(\mathcal{P})$ and $\sigma_2 \in Tr(\mathcal{P}')$, and (ii) σ_1 is the reverse conversation of σ_2 (i.e., the conversation obtained from σ_2 by inverting polarity of messages), and (iii) $\tau = Unp(\sigma_1) = Unp(\sigma_2)$, where $Unp(\sigma)$ denotes the trace obtained from σ by removing the messages polarities.

3 Timed Protocol Analysis

We target two types of protocol analysis, namely *compatibility* and *replaceability* analysis. Compatibility analysis consists in checking whether two services can interact correctly based on their protocol definitions (i.e., whether a conversation can take place between the considered services), while replaceability analysis is concerned with the verification of whether two protocols can support the same set of conversations (e.g., a service can replace another in general or when interacting with specific clients). These two kinds of analysis are useful for lifecycle management of web services as, for example, to provide support for static and dynamic binding as well as in protocol evolution. For both compatibility and replaceability, we have defined several *classes* to identify different levels of compatibility and replaceability, as well as *operators* that can be applied to protocol definition to asses the level of compatibility and replaceability.

3.1 Compatibility Analysis

Compatibility analysis aims at characterizing whether two protocols (which typically depict a service provider and service requester) can interact. It also defines to which extent the compatibility is possible, as some conversations that a protocol supports may not be supported by the other protocol. More specifically, the following compatibility classes can be identified.

- *Partial compatibility* (or simply, compatibility): A timed protocol $\mathcal{P}_1$ is partially compatible with another timed protocol $\mathcal{P}_2$ if there are some executions of $\mathcal{P}_1$ that can interoperate with $\mathcal{P}_2$. In other words, partial compatibility implies that there is at least one timed conversation σ of $\mathcal{P}_1$ which is *"understood"* by $\mathcal{P}_2$ (i.e., the reversed conversation of σ is compliant with $\mathcal{P}_2$).
- *Full compatibility*: a protocol $\mathcal{P}_1$ is fully compatible with another protocol $\mathcal{P}_2$ if all the executions of $\mathcal{P}_1$ can interoperate with $\mathcal{P}_2$, i.e., any conversation that can be generated by $\mathcal{P}_1$ is understood by $\mathcal{P}_2$.

We illustrate compatibility analysis and its challenges on the examples below. Let us consider the protocols $\mathcal{P}$ and $\mathcal{P}'$ depicted at figure 2. Abstracting from the timing constraints, we can observe that $\mathcal{P}$ is fully compatible with $\mathcal{P}'$ (i.e., $a \cdot b \cdot c$ and $a \cdot b \cdot d$ are valid interaction traces of the untimed versions of $\mathcal{P}$ and $\mathcal{P}'$). However, due to the C-Invoke constraints specified on the transitions T_3 of each protocol, $\mathcal{P}$ and $\mathcal{P}'$ cannot interact correctly. Indeed, $\mathcal{P}$ supports timed conversations of the form $(a(-), 0) \cdot (b(+), t) \cdot (c(+), t')$, with $t' < t + 5$ while $\mathcal{P}'$ supports timed conversations of the form $(a(+), 0) \cdot (b(-), t) \cdot (c(-), t')$, with $t' > t + 10$. Hence, these two protocols cannot interact correctly since $\mathcal{P}'$ will always send message c too late. Therefore, to be able to interact correctly, two protocols must agree on the ordering of the messages to be exchanged as well as on the corresponding timing constraints.

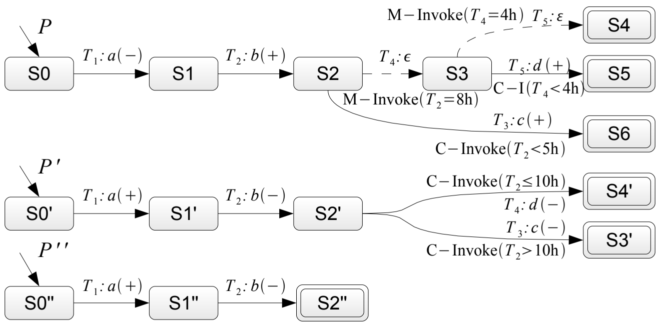


Fig. 2. Three protocols to illustrate protocols analysis.

Let us now consider the protocols $\mathcal{P}$ and $\mathcal{P}''$ of Figure 2. We can observe that when interacting according to the timed interaction trace $(a, 0) \cdot (b, t)$, $\mathcal{P}$ moves to a non-final state s_2 while $\mathcal{P}''$ moves to a final state s_2'' ending its conversation. However, due the presence of the implicit transitions T_4 and T_5, $\mathcal{P}$ is able to terminate correctly its execution by moving automatically to the final state s_4 (i.e., it waits at state s_2 for 8 hours and then moves automatically to state s_3 where it waits for 4 hours before finally moving automatically to the final state s_4). Therefore, the two protocols $\mathcal{P}$ and $\mathcal{P}''$ can interact correctly following the interaction trace $(a, 0) \cdot (b, t)$.

The next example shows that implicit transitions can influence the identification of final states and this naturally impacts compatibility analysis. We consider again protocols $\mathcal{P}$ and $\mathcal{P}'$ of figure 2. After exchanging messages a and b, the two protocols move to states s_2 and s_2' respectively. If we consider the operations that are defined explicitly at these two states, we can observe that s_2' provides an operation $d(-)$ while state s_2 does not enable any invocation of a d operation. Consequently, focusing compatibility checking only on these two states is not enough. Indeed, the presence of the implicit transition T_4 in $\mathcal{P}$ changes the service state automatically to the state s_3 after 8 hours from which $d(+)$ can be fired. Consequently, $\mathcal{P}$ and $\mathcal{P}'$ can interact correctly following timed interactions traces of the form $(a, 0) \cdot (b, t) \cdot (c, t')$, with $t + 8 < t' \leq t + 10$ (i.e., if a message d is sent between 8 and 10 hours after a message b).

3.2 Replaceability Analysis

Replaceability analysis aims at characterizing whether, and to which extent, a given service can be replaced by another one. In such a situation, the substitute service can be transparently used by clients of the original service without the need to change them beyond binding details such as the service URL. Like in the case of compatibility analysis, replaceability analysis aims at supporting flexible schemes as one cannot realistically expect to find services that are completely replaceable. We have identified the following replaceability classes.

– *Protocol equivalence w.r.t. replaceability*: two business protocols $\mathcal{P}_1$ and $\mathcal{P}_2$ are equivalently replaceable if they can be interchangeably used in any context and the change is transparent to clients.
– *Protocol subsumption w.r.t. replaceability*: a protocol $\mathcal{P}_2$ is subsumed by another protocol $\mathcal{P}_1$ w.r.t. replaceability if $\mathcal{P}_1$ supports at least all the conversations that $\mathcal{P}_2$ supports. In this case, protocol $\mathcal{P}_1$ can be transparently used instead of $\mathcal{P}_2$ but the opposite is not necessarily true.
– *Protocol replaceability w.r.t. a client protocol*: A protocol $\mathcal{P}_1$ can replace another protocol $\mathcal{P}_2$ with respect to a client protocol $\mathcal{P}_C$ if $\mathcal{P}_1$ behaves as $\mathcal{P}_2$ when interacting with a specific client protocol $\mathcal{P}_C$. This class is important in those cases where we expect the service to predominantly interact with certain types of clients.
– *Protocol replaceability w.r.t. an interaction role*: Let $\mathcal{P}_R$ be a business protocol. A protocol $\mathcal{P}_1$ can replace another protocol $\mathcal{P}_2$ with respect to a role $\mathcal{P}_R$ if $\mathcal{P}_1$ behaves as $\mathcal{P}_2$ when $\mathcal{P}_2$ behaves as $\mathcal{P}_R$. This replace-ability class allows to identify executions of a protocol $\mathcal{P}_2$ that can be replaced by protocol $\mathcal{P}_1$ even when $\mathcal{P}_1$ and $\mathcal{P}_2$ are not comparable with respect to any of the previous replace-ability classes. This class is important when we want to assess replaceability when considering only certain functionality of the service, e.g., the purchasing part of a supply chain management service.

For all of the above classes, we can distinguish between full and partial replaceability. Full replaceability is as defined above. Partial replaceability is when there is replaceability but only for some conversations and not others. For example, we have partial replaceability with respect to a client protocol when protocol $\mathcal{P}_1$ can replace another protocol $\mathcal{P}_2$ in at least some of the conversations that can occur with $\mathcal{P}_c$.

As an example, consider a protocol $\mathcal{P}_1$ obtained from $\mathcal{P}''$ of Figure 2 by reversing the messages polarities. Such a protocol can be replaced by $\mathcal{P}$ of Figure 2. Indeed, the only timed conversations supported by $\mathcal{P}_1$ are of the form $(a(-), 0) \cdot (b(-), t)$, with $t > 0$. Such conversations are also supported by $\mathcal{P}$. The opposite is however not true. Indeed, $\mathcal{P}$ may support some conversations that contain the messages c or d while $\mathcal{P}_1$ does not. However, we can observe that $\mathcal{P}_1$ can replace $\mathcal{P}$ when interacting with $\mathcal{P}''$: the only timed conversations of $\mathcal{P}$ that are understood by $\mathcal{P}''$ are of the form $(a(-), 0) \cdot (b(-), t)$, with $t > 0$. Such conversations are also supported by $\mathcal{P}_1$.

4 Protocol Operators

The discussion above concerning the compatibility and replaceability classes emphasized the need for operators to analyze and compare timed protocols. There is also a need for understanding (when two timed protocols are neither equivalent nor compatible) which conversations can take place and which ones cannot. This motivates the development of a protocol algebra that enables the manipulation and analysis of timed protocols.

We split the set of protocol operators in two categories: *manipulation* and *comparison* operators. The former category allows to compute protocols that captures

Operator name	**Symbol**	**Semantics**
Compatible Composition	$\Vert^{\mathtt{TC}}$	$\mathcal{P} = \mathcal{P}_1 \Vert^{\mathtt{TC}} \mathcal{P}_2$ is a protocol $\mathcal{P}$ such that $T \in Tr(\mathcal{P})$ iff T is an interaction trace of $\mathcal{P}_1$ and $\mathcal{P}_2$
Intersection	$\Vert^{\mathtt{TI}}$	$\mathcal{P} = \mathcal{P}_1 \Vert^{\mathtt{TI}} \mathcal{P}_2$ is a protocol $\mathcal{P}$ such that $Tr(\mathcal{P}) = Tr(\mathcal{P}_1) \cap Tr(\mathcal{P}_2)$
Difference	$\Vert^{\mathtt{TD}}$	$\mathcal{P} = \mathcal{P}_1 \Vert^{\mathtt{TD}} \mathcal{P}_2$ is a protocol $\mathcal{P}$ that satisfies the following condition: $Tr(\mathcal{P}) = Tr(\mathcal{P}_1) \setminus Tr(\mathcal{P}_2)$
Projection	$\left[\Vert^{\mathtt{TC}}\right]$	Let $\mathcal{P} = \mathcal{P}_1 \Vert^{\mathtt{TC}} \mathcal{P}_2$. $\left[\mathcal{P}_1 \Vert^{\mathtt{TC}} \mathcal{P}_2\right]_{\mathcal{P}_i}$, with $i \in \{1,2\}$, is the protocol obtained from $\mathcal{P}_1 \Vert^{\mathtt{TC}} \mathcal{P}_2$ by defining the polarity function of the messages as follows: $Polarity(\left[\mathcal{P}_1 \Vert^{\mathtt{TC}} \mathcal{P}_2\right]_{\mathcal{P}_i}, m) = Polarity(\mathcal{P}_i, m), \forall m \in \mathtt{M}$

Table 1. Protocol manipulation operators semantics.

a property regarding a pair of protocols, for example to compute a protocol that captures all of the common timed conversations of two protocols. The later category allows to compare two protocols, for example to assess if they are equivalent or not. We define these operators below.

Manipulation operators are applied to protocols and result in protocols. We describe their formal semantics in Table 1. The introduction of time does not change the definition compared to the case (untimed) business protocols of [4].

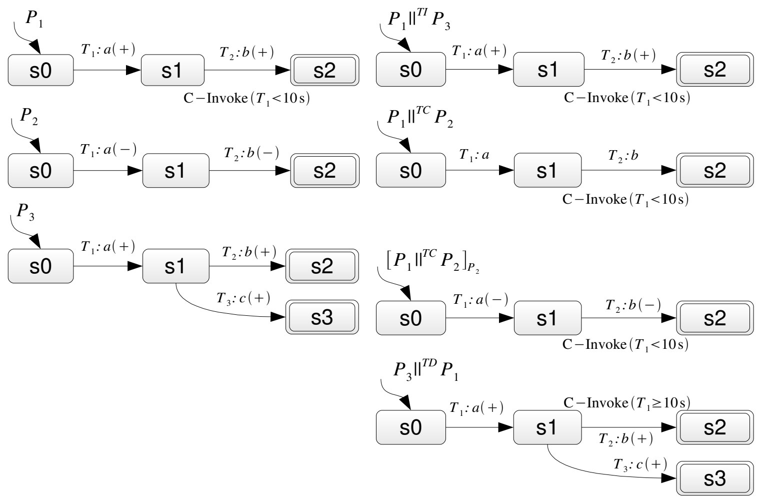


Fig. 3. Three timed protocols $\mathcal{P}_1, \mathcal{P}_2$ and $\mathcal{P}_3$ and some resulting protocols when using protocol manipulation operators.

To illustrate these operators, Figure 3 shows three simple timed protocols $\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{P}_3$ as well as the results when applying operators on them. For example, the protocol $\mathcal{P}_1 \Vert^{\mathtt{TI}} \mathcal{P}_3$ captures the timed conversations that are commonly supported by both $\mathcal{P}_1$ and $\mathcal{P}_3$: $\mathcal{P}_1$ does not support receiving a message c, hence it does not appear in $\mathcal{P}_1 \parallel^{\mathtt{TI}} \mathcal{P}_3$. Similarly $\mathcal{P}_1$ can only receive a b message within the 10 seconds that follow the reception of a a message. Another example is the protocol $\mathcal{P}_3 \parallel^{\mathtt{TD}} \mathcal{P}_1$ that captures all the conversations that $\mathcal{P}_3$ supports, but that $\mathcal{P}_1$ doesn't. This is why the C-Invoke constraint of T_2 in $\mathcal{P}_3 \parallel^{\mathtt{TD}} \mathcal{P}_1$ is the negation of the one of T_2 in $\mathcal{P}_1$ as $\mathcal{P}_3$ does not carry a C-Invoke constraint on its transition T_2. Similarly, $\mathcal{P}_3$ supports receiving c messages while $\mathcal{P}_1$ does not.

We define two comparison operators, namely *subsumption* and *equivalence*. They enable to compare timed protocols w.r.t. their timed conversations. The subsumption, noted $\sqsubseteq$, assesses whether one protocol supports all of the timed conversations of another protocol (i.e., $\mathcal{P} \sqsubseteq \mathcal{P}'$ iff $Tr(\mathcal{P}) \subseteq Tr(\mathcal{P}')$). The equivalence, noted $\equiv$, checks whether two protocols support exactly the same set of conversations (i.e., $\mathcal{P} \equiv \mathcal{P}'$ iff $Tr(\mathcal{P}) = Tr(\mathcal{P}')$.

Class	**Characterization**
Partial compatibility of $\mathcal{P}_1$ and $\mathcal{P}_2$	$\mathcal{P}_1 \parallel^{\mathtt{TC}} \mathcal{P}_2$ is not *empty*
Full compatibility of $\mathcal{P}_1$ and $\mathcal{P}_2$	$\left[\mathcal{P}_1 \parallel^{\mathtt{TC}} \mathcal{P}_2\right]_{\mathcal{P}_1} \equiv \mathcal{P}_1$
Replaceability of $\mathcal{P}_1$ by $\mathcal{P}_2$	$\mathcal{P}_2 \sqsubseteq \mathcal{P}_1$
Equivalence of $\mathcal{P}_1$ and $\mathcal{P}_2$ w.r.t. replaceability	$\mathcal{P}_1 \equiv \mathcal{P}_2$
Replaceability of $\mathcal{P}_2$ by $\mathcal{P}_1$ w.r.t. a client protocol $\mathcal{P}_C$	$\left[\mathcal{P}_C \parallel^{\mathtt{TC}} \mathcal{P}_2\right]_{\mathcal{P}_2} \sqsubseteq \mathcal{P}_1$ or equivalently $\mathcal{P}_C \parallel^{\mathtt{TC}} (\mathcal{P}_2 \parallel^{\mathtt{TD}} \mathcal{P}_1)$ is *empty*
Replaceability of $\mathcal{P}_2$ by $\mathcal{P}_1$ w.r.t. a role $\mathcal{P}_R$	$(\mathcal{P}_R \parallel^{\mathtt{TI}} \mathcal{P}_2) \sqsubseteq \mathcal{P}_1$

Table 2. Characterization of the compatibility and replaceability classes.

The characterization of the protocol compatibility and replaceability analysis classes using the protocol manipulation and comparison operators is given in Table 2. The introduction of time does not change the characterization that had been defined for (untimed) business protocols in [4].

5 Protocol Operators Properties

This section investigates the decidability and complexity properties underlying our protocol operators. We show that there is a semantic-preserving mapping from protocols into a new class of timed automata [7] with ε-transitions (i.e., ε-transitions). We illustrate such a mapping on an example and then we discuss how existing results in timed automata theory can be reused/extended to deal with our specific problems. More technical details can be found in [8].

5.1 Mapping Protocols into Timed Automata

Briefly, a timed automaton [7] is a finite automaton augmented with a finite set of real-valued clocks. Clock constraints can be associated with transitions and can also be reset to zero simultaneously with any transition. Figure 4 shows a timed protocol and its corresponding timed automaton. The obtained automaton uses two clock variables, x_1 and x_2, to implement the timing constraints described in the corresponding timed protocol. For example, the constraint C-Invoke($T_1 < 5h$) of transition T_1 is captured in the timed automaton by the constraint $x_1 < 5$ associated with the arc $b(+)$ between states s_1 and s_2. Indeed, this constraint is defined over variable x_1 which is reset to zero when the automaton switches from state s_0 to s_1 on symbol $a(-)$. Then, while the automaton is at state s_1, the value of variable x_1 shows the time elapsed since the occurrence of the last transition $s_0 \cdot a(-) \cdot s_1$. The transition from state s_1 to s_2 on symbol $b(+)$ is enabled only if the value of variable x_1 is less than 5. Thus, the timing constraint expressed by this automaton is that the symbol $b(+)$ must occur less than 5 units of time after the occurrence of the symbol $a(-)$ (and this is exactly what the constraint C-Invoke($T_1 < 5h$) on transition T_1 prescribes). Also, note that the implicit transition T_4 in the timed protocol is described using an ε-transition between states s_2 and s_4 in the corresponding timed automaton. The associated M-Invoke($T_2 = 10h$) constraint is modeled in the timed automaton using two clock constraints $x = 10$, associated with the ε transition, and $x < 10$ associated with the remaining transition that is enabled from state s_2 on symbol $c(+)$. In the remainder, we assume that timed protocols have been *normalized* by making explicit all the temporal constraints as described above.

Timed automata are in general more expressive than timed protocols and hence not any timed automaton can be mapped into a protocol. However, the restricted class of timed automata that are obtained by a mapping from a timed protocol can be translated back into timed protocols without loss of semantics. We call this class *PTA* for *Protocol Timed Automata*. The procedure that we propose translates a protocol into a timed automaton, as briefly explained below. To do that, and to make sure that the mapping is effectively bijective, we give three conditions that identify timed automata that can be mapped back into timed protocols.

Let $\mathcal{P} = (\mathcal{S}, s_0, \mathcal{F}, \mathtt{M}, \mathcal{X}, \mathcal{C}, \mathcal{R})$ be a timed protocol. An associated timed automaton $A_{\mathcal{P}} = (L, L^0, L^f, X_{\mathcal{P}}, E)$ over alphabet $\Sigma_{\mathcal{P}} = \mathtt{M}$ is built as follows: $L = \mathcal{S}$, $L^0 = \{s_0\}$, $L^f = \mathcal{F}$, $X_{\mathcal{P}} = \mathcal{X}$ and $\forall r = (s, s', m, c) \in \mathcal{R}$, a new switch $(s, a, \varphi, \lambda, s')$ is added to E such that: $a = m$, $\varphi = \alpha$ if $c =$ C-Invoke(α) or $c =$ M-Invoke(α), and $\lambda = \{x_r\}$. Figure 4 depicts a timed protocol and its associated timed automaton.

Let $A = (L, L^0, L^f, X, E)$ be a timed automaton verifying the following conditions:

($\mathbf{C_1}$) $\forall e = (l, a, \varphi, \lambda, l') \in E$, $|\lambda| = 1$ (i.e., exactly one clock is reset), and the clock in λ is only reset on e: for every two distinct switches $(l_1, a_1, \varphi_1, \lambda_1, l'_1)$ and $(l_2, a_2, \varphi_2, \lambda_2, l'_2)$ of E, we have $\lambda_1 \cap \lambda_2 = \emptyset$,

($\mathbf{C_2}$) A is deterministic, i.e., for every two switches $(l, a, \varphi_1, \lambda_1, l'_1)$ and $(l, a, \varphi_2, \lambda_2, l'_2)$ from E recognizing the same event a from the same location l, then $\varphi_1 \wedge \varphi_2 \equiv$ `false`,

($\mathbf{C_3}$) The allowed guards of the ε-transitions are conjunctions of atomic equality and inequality constraints such that each guard has at least 1 equality constraint,

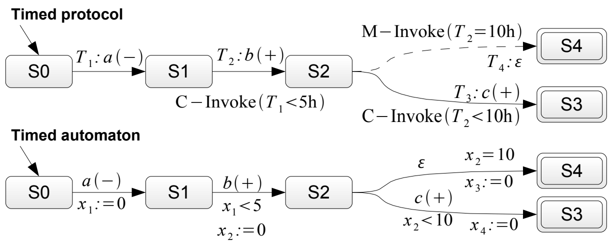


Fig. 4. A timed protocol and its associated timed automaton.

(C_4) Given the set $\{(l, \varepsilon, \varphi_1, \lambda_1, l'_1), \cdots, (l, \varepsilon, \varphi_n, \lambda_1, l'_n)\}$ of ε-transitions starting from a location l, the guard φ_j of each switch $(l, m, \varphi_j, \lambda_j, l'_j)$ (with $m \in \Sigma \cup \{\varepsilon\}$) satisfies $\bigwedge_{i \neq j} (x_i < k_i)$ such that $(x_i = k_i)$ appears in φ_k with $k \in \{1, \cdots, n\}$.

Condition (C_1) enforces that every switch resets only one clock. Indeed, a timed automaton switch is allowed to reset an arbitrary number of clocks, while in the case of the mappings of timed protocols we need to reset only one clock: the one that is associated with the transition. This defines a bijection between the set of clocks and the set of switches. Condition (C_2) ensures determinism the guards of 2 switches that recognize the same event from the same location must be disjoint. Condition (C_3) enforces the definition of guards on the ε-labeled switches. Finally, condition (C_4) enforces the semantics of the M-Invoke constraints (determinism and preemption).

Every timed automaton that verifies the conditions (C_1), (C_2), (C_3) and (C_4) above can be mapped into a timed protocol, and hence is a *timed protocol automaton*. This mapping can be performed by reversing the procedure described above. The following theorem says that a timed protocol and its associated timed automaton are semantically equivalent.

Theorem 1. *Let $\mathcal{P}$ be a timed protocol and $A_{\mathcal{P}}$ its associated protocol timed automata. Then: $Tr(\mathcal{P}) = \mathcal{L}(A_{\mathcal{P}})$, where $\mathcal{L}(A_{\mathcal{P}})$ denotes the timed language recognized by the automaton $A_{\mathcal{P}}$.*

The proof derives from the definition of both the PTA class and the mapping from timed protocols to PTA.

5.2 Closure Property of Protocol Manipulation Operators

Through the aforementioned mapping, we derive results regarding intersection and compatible composition operators. Indeed, it is well known that timed automata are closed under intersection [6]. Such a property is established by extending the classical automata product construction to timed automata. In [8] we extend the product construction to show that the closure property also holds for the PTA class (e.g., intersection of two timed protocol automata is a timed protocol automaton). This leads to an algorithm to compute timed intersection or composition of protocols.

The situation regarding the difference and subsumption operators is however more complex. The main problem lies in the undecidability of the complementation in timed automata with ε-transitions. Since the difference and the language inclusion problem (or subsumption) depend on the complementation (e.g., $A \backslash B \equiv A \sqcap \overline{B}$ and $L(A) \subseteq L(B)$ iff $L(A) \cap L(\overline{B}) = \emptyset$), the decidability of these operators requires a proper investigation of the characteristics of protocol timed automata. The main difficulty lies in the presence of the ε-transitions, which unlike in the case of classical (untimed) automata, strictly increase the expressiveness level of timed automata. [9] investigates the expressive power of ε-transitions and identifies cases where ε-transitions can be removed without a loss of expressiveness (e.g., case of ε-transitions that do not reset clocks). Unfortunately, this result is of no use in our case as the ε-transitions that we deal with do not belong to the identified cases. Indeed, in the PTA class, ε-transitions strictly increase the expressiveness of protocol timed automata as they reset clocks, and hence they cannot be removed [10]. However, we have shown that the class PTA is closed under complementation, which allows claiming that timed protocols are closed under difference. Moreover, since PTA are closed under intersection, and given that the emptiness checking problem is decidable for timed automata [6], the protocol subsumption and equivalence problems are decidable. The proof of closure under complementation is based on the observation that although PTA automata contain ε-transitions with clocks resets, they still exhibit a *deterministic behavior* which ensures that at each step of an execution, all clock values are solely determined by the input word. Therefore, closure under complementation can be proved by extending the usual construction to PTA. The main result of this section is given below.

Theorem 2. *Timed protocols are closed under intersection, compatible composition and difference.*

Performing a subsumption or equivalence test between two protocols is thus decidable, as described by the theorem hereafter. We also give a complexity result which is derived from existing work on the *timed language inclusion* problem for timed automata [7].

Theorem 3. *The subsumption and equivalence operators on timed protocols are decidable, and their decision problems are* PSPACE-Complete.

With the two theorems above, we have proved that our full set of operators can be implemented by reusing the already-known constructs on timed automata [7]. Those results directly come from the novel class of timed automata that we have identified. This makes it possible to conduct automated analysis for all of the compatibility and replaceability classes on timed protocols.

6 Implementation and Discussion

We have developed a prototype as part of the larger *ServiceMosaic* project [5]. Briefly, *ServiceMosaic* (see http://servicemosaic.isima.fr/) is a CASE-toolset model-driven prototype platform for modeling, analyzing, and managing web service models including business protocols, orchestration, and adapters. The *ServiceMosaic* projects are developed for the Java$^{\text{TM}}$ platform version 5. We created libraries that provide the functionalities of our contributions, then we integrate them into the Eclipse platform as plug-ins. Regarding the work presented in this paper, we have designed a model for timed protocols and implemented the operators (the subsumption and equivalence operators rely on the *UPPAAL* model checker). We have also created a graphical editor for protocols as well as component that can extract the protocols of the services that are used in a BPEL orchestration. In our experimentations, we have also worked on protocols (manually) extracted from RosettaNet PIPs.

The approach that we have described in this paper can be used in several practical contexts. We briefly outline one of them where we have used our prototype to facilitate service composition development [8]. Given a BPEL orchestration, we have used it to check if the selected services where fully or partially compatible with the BPEL process behavior. By identifying which conversations can or cannot be carried out, we have been able to support the development of protocol adapters in a similar fashion as in [11] which tackles adaptation in the case of untimed business protocols.

We now provide a brief outlook of related work. Several ongoing efforts in the area of Web services recognize the importance of high level modeling and analysis of services protocols (e.g., [3,4,12,13]). Similar approaches for protocols compatibility and replaceability exist in the area of component-based systems [14,15]. In terms of protocol description, the existing models do not explicitly take timing constraints into account. In terms of protocols analysis, mechanisms have been proposed to verify protocols compatibility and replaceability. However, the verifications are still *"black or white"* whereas our approach targets a fine-grained analysis for the cases where *partial* results are desirable. Standardization efforts recognize the need for supporting the explicit description of web services functional and non-functional properties [16]. Of most interest in the case of making explicit business protocols are the Business Process Execution Language for Web Services (BPEL), the Web Services Conversation Language (WSCL) and the Web Service Choreography Interface (WSCI). Documents complying to those specifications can be derived from protocols and vice-versa as our approach is complementary to them.

In our work, we used a states machine-based model for describing protocols. However, the formal foundations could have been also based on another model such as Petri nets. In this case, the protocol operators would have to be ported to this formalism to be able to perform compatibility and replaceability analysis. In fact, timed protocols can be viewed as a syntactic variant of timed automata. In this paper we have also significantly extended our initial work on service protocols [17] by proposing: (i) a model for service business protocols that supports rich timing constraints, (ii) a set of fine-grained protocol compatibility and replaceability classes, and (iii) a set of operators with formal foundations that can be combined for performing those types of analysis. The results we have achieved is a framework and a tool that can support development and binding of services with timing properties. We believe that this is a significant contribution as the number of available services increases and as the need of automated support for service lifecycle management becomes a necessity. Interestingly, this work has also lead to the discovery of an innovative class of timed automata. In future work, we aim at extending the approach for analyzing web services compositions in presence of timing abstractions.

A Theoretical study

A.1 Characterization of Protocol Timed Automata

Lemma 1. *The mapping from a timed protocol $\mathcal{P}$ to a timed automaton A yields a protocol timed automaton. Also, the inverse mapping of a A into $\mathcal{P}$ yields a timed protocol.*

Proof. The mapping of the states, transitions and messages of $\mathcal{P}$ to locations, switches and alphabet in A is straightforward. We show that the conditions $(C_1), (C_2), (C_3)$ and (C_4) of protocol timed automata are satisfied by construction.

- (C_1): $\forall r = (s, s', m, c) \in \mathcal{P}$, the mapping generates a switch $(s, m, \varphi_c, \lambda, s')$ such that $\lambda = \{x_r\}$ is the reset on the unique clock which is associated to the switch. Hence, two switches in A cannot reset the same clock. Also, the variables $\mathcal{X}$ in $\mathcal{P}$ are mapped to clocks X in A and the constraints are preserved (e.g., C-Invoke($T_1 < 3$) in $\mathcal{P}$ becomes $x_{T_1} < 3$ in A).
- $(C_2), (C_3)$: they are satisfied by definition of timed protocols and M-Invoke constraints.
- (C_4): this condition can be satisfied after making all C-Invoke constraints explicit: $\forall (s, s'_1, \varepsilon, c_1), (s, s'_2, m, c_2) \in \mathcal{P}$,

$$c_2 \models \text{C-Invoke}\,(\neg c_1 \ \texttt{and}\ ((T_r < k_r)\ \texttt{and}\ \cdots))$$

  where $(x_r = k_r)$ appears in c_1. This is then mapped in A as the following constraint which satisfies (C_4):

$$\neg\varphi_{c_1} \wedge ((x_{T_r} < k_r) \wedge \cdots)$$

We can also show that the conditions $(C_1), (C_2), (C_3)$ and (C_4) ensure that the inverse mapping from A to $\mathcal{P}$ preserves the definition of timed protocols.

- (C_1) ensures that each clock of A is associated to a unique transition variable in $\mathcal{P}$. This also preserves the constraints expressions.
- (C_2) ensures determinism in $\mathcal{P}$.
- (C_3) ensures that the M-Invoke constraints can be correctly mapped from the guards of the ε-transitions of A.
- (C_4) ensures that the M-Invoke constraints preempt the explicit transitions when their condition become satisfied. It also enforces them to be fired, as the C-Invoke constraints become disabled once there is no more implicit transition whose M-Invoke constraint can be satisfied.

Lemma 2. *Let $\mathcal{P}$ be a timed protocol, and let A be its associated protocol timed automaton: $Tr(\mathcal{P}) = \mathcal{L}(A)$.*

Proof. We start by showing that $Tr(\mathcal{P}) \subseteq \mathcal{L}(A)$. To do that, we first consider a timed trace $\sigma = (a_0, t_0) \cdots (a_n, t_n) \in \mathcal{P}$. We propose the following induction regarding the execution of σ over A: we show that each symbol a_i $(0 \leq i \leq n)$ can be *recognized* by A until a_n which leads to σ to be *accepted* by A as it reaches a final location. The property used in the induction is that *the mapping of $\mathcal{P}$ into A is correct*, which we discuss after the induction.

(a_0) is recognized by A at time t_0, else the mapping is incorrect. Let $0 < k < n$ such that the prefix $(a_0, t_0) \cdots (a_k, t_k)$ of σ has been recognized by A. Then a_{k+1} is also recognized by A at time t_{k+1}, else the mapping is incorrect. Finally, a_n is recognized at time t_n by A, else again, the mapping would be incorrect.

The mapping of $\mathcal{P}$ into A is incorrect if any of the following cases is true.

1. Either of the conditions $(C_1), (C_2), (C_3)$ and (C_4) is violated. This is impossible by Lemma 1.
2. Some states or transitions in $\mathcal{P}$ have not been mapped to locations and switches in A. This is impossible by construction.
3. Similarly as the case above, initial and final states in $\mathcal{P}$ have not been mapped as initial and final locations in A, which is also impossible by construction.
4. Given $s \xrightarrow{a,c} s' \in \mathcal{P}$ and the corresponding switch $s \xrightarrow{a,\varphi_c} s' \in A$, there exists a timed word σ' such that when c is satisfied, φ_c is not. This case cannot happen as the constraints mapping does not change them.

As a consequence, the induction is correct: $\sigma \in \mathcal{L}(A)$, hence $Tr(\mathcal{P}) \subseteq \mathcal{L}(A)$.

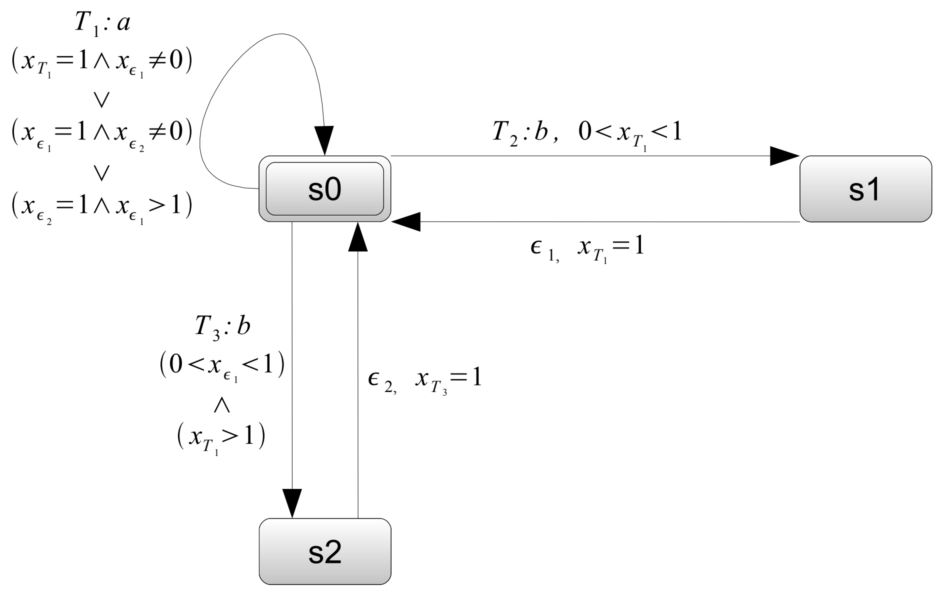
It can be shown in a similar fashion that given $w \in \mathcal{L}(A)$, $w \in Tr(\mathcal{P})$, hence $\mathcal{L}(A) \subseteq Tr(\mathcal{P})$ and consequently $Tr(\mathcal{P}) = \mathcal{L}(A)$.

Theorem 4. *ε-transitions strictly increase the expressiveness of protocol timed automata.*

Proof. We need to show that ε-transitions in protocol timed automata cannot always be removed, i.e., there are protocol timed automata for which there doesn't exist equivalent automata without ε-transitions. To do that, we exhibit the protocol timed automaton A depicted on Figure 5 and use the notions of *precise time* and *precise actions* that were introduced in the Theorem 24 of [9] as a tool to identify timed languages that can only be recognized by timed automata featuring ε-transitions. The proof is actually the same as the one of Corollary 29 in [9].

It is easy to check that A is a protocol timed automaton verifying the conditions $(C_1), (C_2), (C_3)$ and (C_4) above. A presents 2 ε-transitions lying on directed cycles, hence we don't know if they can be removed using the techniques presented in Section 8 in [9].

Let us now suppose that $\mathcal{L}(A)$ can be recognized by a timed automaton A' without any ε-transition. Note that A' is necessarily free of diagonal constraints (e.g., constraints of the form $x - y \# c$) by definition of protocol timed automata. Also, A' can be rendered disjunction-free without any loss of generality (see [9] for



$T_1: a$
$(x_{T_1}=1 \wedge x_{\epsilon_1} \neq 0)$
$\vee$
$(x_{\epsilon_1}=1 \wedge x_{\epsilon_2} \neq 0)$
$\vee$
$(x_{\epsilon_2}=1 \wedge x_{\epsilon_1} > 1)$
s0
$T_2: b,\ 0 < x_{T_1} < 1$
s1
$\epsilon_1,\ x_{T_1}=1$
$T_3: b$
$(0 < x_{\epsilon_1} < 1)$
$\wedge$
$(x_{T_1} > 1)$
$\epsilon_2,\ x_{T_3}=1$
s2

Fig. 5. A protocol timed automaton A that cannot be expressed equivalently without ε-transitions.

techniques and discussion). In order to leverage the Theorem 24 of [9], we define $C_{\max} = 1$ (no constant in the guards of A are larger than 1). Let also $\delta > 0$. A can recognize words of the form

$$(b, \delta_1) \cdot (b, \delta_2) \cdots (b, \delta_{d-1}) \cdot (a, d) \cdot (a, d+1) \cdots$$

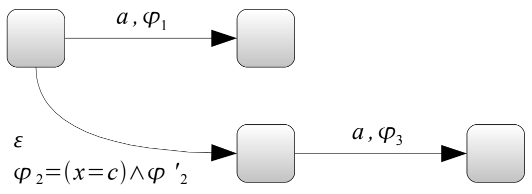
where $d \in \mathbb{N}$, $d \geq C_{\max}$ and $\delta_i \in (i-1, i) \setminus \delta\mathbb{N}$ for all $0 < i < d$. Let P a path of A' that accepts such a timed word. Given that the a-labeled events occur at integer times, their occurrences should be *precise* in P. Also, $d \geq C_{\max}$, hence from Theorem 24 of [9], there exist some occurrence of b that should be precise in P which is not possible as $\delta_i \notin \delta\mathbb{N}$ for any $0 < i < d$. Consequently, $\mathcal{L}(A)$ cannot be recognized by a timed automaton without ε-transitions.

Lemma 3. *Protocol timed automata are deterministic: given a protocol timed automaton A and a timed word $w \in \mathcal{L}(A)$, w has exactly one run over A.*

Proof. There can be more than one run of w over A if given a location $l \in A$ and an input symbol a at time t, at least two switches can recognize a (i.e., their guards can be both satisfied by the clock valuations at time t). If l does not exhibit any ε-transition, then this is not possible by condition (C_2). The same condition also ensures that if l exhibits more than one ε-transition, then their guards are also disjoint.

Let us now consider the case where l exhibits a ε-transition, an a-labeled transition and the state that is reached through the ε-transition also exhibits an

a-labeled transition:



By definition, $\varphi_1 \models (x < c)$ and $\varphi_3 \models (x \geq c)$, hence $\varphi_1 \wedge \varphi_3 \models \texttt{false}$ for any clock valuation. Moreover, given a ε-transition e from a location l, every non-ε-transition is disabled when the guard of e becomes satisfied (condition (C_4)).

Consequently, w has exactly one run over A.

A.2 Characterization of Protocol Operators

Theorem 5. *The class of protocol timed automata is closed under intersection and parallel composition.*

Proof. Intersection and parallel composition are computed from the product of the states by synchronizing on the switches labels. For instance two switches $(l_1, a, \varphi_1, \lambda_1, l'_1)$ and $(l_2, a, \varphi_2, \lambda_2, l'_2)$ each in a distinct timed automaton can be synchronized to form a switch $((l_1, l_2), a, \varphi_1 \wedge \varphi_2, \lambda_1 \cup \lambda_2, (l'_1, l'_2))$ in the intersection. Parallel composition only varies on the synchronization function: in the case of intersection we can synchronize $a(+)$ and $a(+)$ while in parallel composition $a(+)$ and $a(-)$ would be synchronized. ε-labeled switches have to be handled differently as they must not be synchronized: when a location offers such a switch, it must be reproduced in the intersection or parallel composition automaton. This could potentially introduce indeterminism on protocol timed automata, hence we provide a modified construction that exploits the specificities of the model to handle ε-labeled switches. We introduce two variations: the first one is on the guards and clocks while the second one handles ε-labeled switches.

When performing the synchronization of two switches on timed automata, their clock reset sets are merged: $\lambda_1 \cup \lambda_2$. In protocol timed automata, a unique clock is reset on every switch. We take advantage of this by creating a new clock on every switch of the intersection or parallel composition, and rewrite the constraints. For example, a constraint $\varphi = (x_{e_1} < 3)$ is rewritten as $\varphi' = (x_{(e_1, e'_1)} < 3)$ if e'_1 can be synchronized with e_1.

Consider two protocol timed automata A and A'. Given two locations $l \in A$ and $l' \in A'$ such that a synchronization on a non-ε event is possible, we consider each pair $(l, \varepsilon, \varphi_i, \lambda_i, l_i) \in A$ and $(l', \varepsilon, \varphi'_j, \lambda'_j, l'_j) \in A'$ of their ε-labeled switches. Note that in the case where only either l or l' offers ε-labeled switches, they can be simply reported in the intersection or parallel composition. Depending on clock resets, three cases are possible:

1. φ_i becomes satisfied before φ_j, or
2. φ_j becomes satisfied before φ_i, or

3. both φ_i and φ_j become satisfied at the same time.

Thus, we add to the intersection or parallel composition of A and A' the following switches:

$$e_1 = \big((l,l'),\varepsilon,\varphi_i \wedge \neg\varphi_j,\lambda_{e_1},(l_i,l')\big)$$
$$e_2 = \big((l,l'),\varepsilon,\neg\varphi_i \wedge \varphi_j,\lambda_{e_2},(l,l'_j)\big)$$
$$e_3 = \big((l,l'),\varepsilon,\varphi_i \wedge \varphi_j,\lambda_{e_3},(l_i,l'_j)\big)$$

where λ_{e_1}, λ_{e_2} and λ_{e_3} each reset a new single clock x_{e_1}, x_{e_2} or x_{e_3}. Note that if one automaton (e.g., A) had a constraint on such a ε-labeled switch (e.g., $x_{e_k} < 3$ with $\lambda_i = \{x_{e_k}\}$ from $(l,\varepsilon,\varphi_i,\lambda_i,l_i)$ above), then the constraint in the intersection or parallel composition is rewritten as a disjunction (e.g., $x_{e_1} < 3 \vee x_{e_3} < 3$). Finally, we enforce condition (C_4) on the non-ε-labeled switches from (l,l').

We can check that this construction satisfies the definition of a protocol timed automaton, hence the closure property under intersection and parallel composition.

- (C_1): this comes from the modifications on the clocks assignment and guard constraints rewriting.
- (C_2): by definition of the synchronization on non-ε-labeled switches, and by the way ε-labeled switches are added to the intersection of parallel composition.
- (C_3) and (C_4): by construction.

Corollary 1. *Timed protocols are closed under intersection ($\|^{\mathtt{TI}}$) and parallel composition ($\|^{\mathtt{TC}}$).*

Theorem 6. *The class of protocol timed automata is closed under complementation.*

Proof. We compute the complement of a protocol timed automaton using the following procedure which is derived from the one for deterministic timed automata as given in [7], with the difference lying in the presence of ε-transitions. Then, we show that this construction satisfies the definition of a timed automaton.

Given a protocol timed automaton A, we denote by A^* its *complete* automaton which is build as follows.

1. A location q is added to A^* whose role is to act as a *rejection* location: given any timed word w defined over $\overline{\mathcal{L}(A)}$, the execution of w over A^* goes to the location q as soon as an input symbol yields to a word which is not in $\mathcal{L}(A)$. Hence, any timed word w defined over the alphabet of A has a (unique) execution over A^*.
2. For each location l of A (this includes q) where $\nexists l'$ such that $l \xrightarrow{\varepsilon} l'$, and for each symbol a, a transition $l \xrightarrow{a,g} q$ is added. The guard g is defined as the negation of the disjunctions of the guards of the other a-labeled transitions from l.
3. For each location l of A where $\exists l'$ such that $l \xrightarrow{\varepsilon,\varphi} l'$, we add transitions $l \xrightarrow{a,g} q$ as in the case where there is no ε-transition, but g must also satisfy $\neg\varphi \wedge ((x_i < k_i) \wedge (x_{i+1} < k_{i+1}) \wedge \cdots)$ where each $x_i < k_i$ comes from every $x_i = k_i$ constraint that appears in φ.

As in [7], the complement $\overline{A}$ of A is deduced from A^* by inverting the final and the normal locations due to the fact that every timed word $w \in \mathcal{L}(A)$ has a unique run over A.

We can check that this construction satisfies the definition of a protocol timed automaton, hence the closure property under complementation.

- (C_1): A already satisfies this, and the switches to q that are added in A^* can also be assigned a clock each.
- (C_2): the switches that originate from A already satisfy this, and the ones that have been added in A^* satisfy this requirement by construction.
- (C_3) and (C_4): from A and by construction.

Corollary 2. *Timed protocols are closed under difference ($\|^{\mathtt{TD}}$).*

Proof. Protocol timed automata are closed under complementation and intersection, hence timed protocols are closed under difference.

Corollary 3. *The subsumption ($\sqsubseteq$) and equivalence ($\equiv$) decision problems for protocol timed automata are* PSPACE-COMPLETE*.*

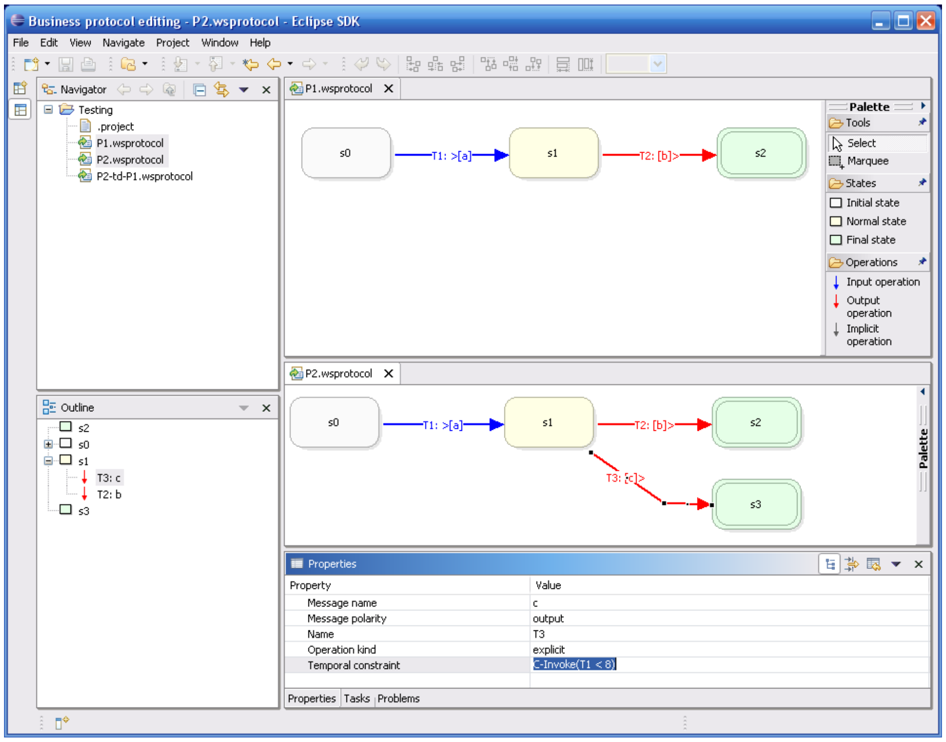
Proof. The subsumption and equivalence reduce to checking wether the language that is recognized by a protocol timed automaton is empty. This can be done using a timed automata *emptiness checking* technique such as the construction of the *regions automata* or the *zones automata* which are known to be PSPACE-COMPLETE [6, 7, 18].

B Protocol Analysis at Work

We know show how the prototype (see Figure 6) can be used to facilitate service development. We assume that a developer is defining a BPEL process, related to the handling of a purchase order, and that the process invokes several services during its execution. The tool will assist the developer in checking if the selected services have a protocol which is fully or partially compatible with the defined BPEL process, will identify which conversations can and cannot be carried out, and will also tackle the case of non compatibility by supporting the development of protocol adapters.

B.1 BPEL Process Outline

Consider the BPEL process depicted on Figure 7. It orchestrates four web services to process a purchase order. For the sake of clarity, we have removed the *assign* BPEL instructions from the process diagram, normally required to prepare and reuse the messages exchanged with the involved web services. The first part of the process handles the payment options. If the customer asks for a loan, then the process will make an offer using the *accounting* web service. The customer can then accept or reject it. The asynchronous *pick* BPEL construction also defines an alarm that will be fired after 72 hours to discard the process instance if the



Property	Value
Message name	c
Message polarity	output
Name	T3
Operation kind	explicit
Temporal constraint	C-Invoke(T1 < 8)

Fig. 6. Screenshot of the ServiceMosaic protocol development and analysis prototype.

customer does not reply in time to the loan offer. The second part checks for the ordered goods availability with the warehouse web service. If some goods are not available, they will be ordered. In order to match quality of service requirements, the purchase is canceled if the warehouse does not manage to purchase the missing goods within 48 hours. The third an last part of the process handles the payment and prepares the goods delivery. Finally, the customer is notified that the purchase has successfully completed.

B.2 Business Protocols Extraction

Based on this BPEL process definition, we extract the timed protocols that the process supports when interacting with its partner services. To do this, we use the multi-party protocol BPEL extractor that we developed, and we then obtain the protocol governing the interaction of the process with each of the partner services by filtering the multi-party protocol based on each service partner link. The resulting protocols are shown in Figure 8. Figure 9 shows instead the protocol of the warehouse service we are planning to use as one of the services invoked by our process.

B.3 Protocol Analysis

We next apply the protocol analysis operators to assess compatibility between the protocols supported by our process and the protocols of the services we plan to use. For this, we assume that either the protocol or BPEL definition (from which we extract the protocol) of these services is available. Figure 10) shows the results of this analysis for the warehouse service. In particular, the compatible composition operator $P_5 \parallel^{\mathtt{TI}} P_3$ gives the set of the conversations that can occur between protocols P_3 and P_5. Ideally, we would want this set to be equal to the conversations supported by P_3, which means that P_5 is fully compatible with P_3.

However, in our example, we do not have such luck. In fact we see that the conversations supported by the compatible composition are a subset of those supported by P_3. The Figure further shows the conversations that are supported by the process but not by our partner service P_5 (which is empty in case of full compatibility), as well as the conversations that the partner supports but that the process does not support. The first of these two combined protocol is obtained by computing the inverse P_3' of P_3 and then the difference $P_3^{-1} \parallel^{\mathtt{TD}} P_5$. The latter is instead computed as $P_5 \parallel^{\mathtt{TD}} P_3^{-1}$. As we will examine later, all these combined protocols will become helpful in examining if and which changes need to be made to the process.

In particular, while the first combined protocol of Figure 10 (compatible composition) tells us what we can do, the second one denotes what our process is prevented from doing when using this partner (hence we call these *prevented interactions*), while the third one denotes conversations that the partner would support, but we are not leveraging due to how we implemented the process. We call these *neglected interactions*.

It is interesting to note that no compatibility problem would have been spotted in the case of business protocols without timing constraints [4]. Indeed, the untimed version of P_5 would have supported all of the conversations of the untimed version of P_3.

B.4 Managing Partial Replaceability Scenarios

By looking at the three combined protocols, the developer can assess if the selected service is a good fit or not, and how to handle situations of partial replaceability or of no replaceability. In general, this depends on the specific business purpose of the process. For example, the service I am planning to invoke may not support a *cancelPO* operation, but I may be willing to take the risk and use it anyways even if cancellations are not allowed, for example because it offers cheaper rates. Or, conversely, the selected service supports several forms of payments (accessed via different protocols) but my process can only support one of them, and we may be fine with it as for example our company only issues payments via credit card and not via bank transfers.

Alternatively, we can modify the process definition to adapt it to the service we are using, either to i) ensure that our process does not generate conversations our partner cannot understand, or to ii) leverage conversations supported by our selected services (e.g., extend our process to support bank transfers). As another example, in our process, we can remove the *onAlarm 48h* handler of the second *pick* complex activity, so that the process will wait for the *purchaseResponse* message to arrive, thereby removing the problematic temporal constraints in the extracted expected warehouse protocol. However, the process may find itself being put on hold indefinitely if a problem occurs on the warehouse service and it does not send a *purchaseResponse* message back.

Another solution is to generate a protocol adapter [11] to reconcile the differences. It can be done with the ServiceMosaic tools using an aspect-oriented framework [19] where adapters are plugged through *advices* written in BPEL. The adapter is be developed as follows. The *pointcut* is triggered when a *purchaseRequest* message is received. The advice is a BPEL process where an alarm starts counting from the reception of the *purchaseRequest* message. If the service does not send a *purchaseResponse* withing the next 48 hours, then the adapter drops it when the warehouse service sends it afterwards. The BPEL engine will have already woken up the process instance by then, and taken action by replying to the client partner link with a *cancelPO* message.

Finally, it should be noted that for most BPEL engines, a message is simply dropped when it cannot be dispatched to any process instance for which it is waiting. An exception is then usually raised and logged inside the BPEL engine. In this example the adapter would be useful for diminishing the number of internally-thrown exceptions (raising exceptions has a significant performance cost). The choice of developing this adapter should be balanced in light of its development cost compared to the (limited) benefits, as BPEL engines can provide a form of "implicit" adapter in very specific mismatches cases such as this one.

References

1. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services: Concepts, Architectures, and Applications. Springer Verlag (2004)
2. Boualem Benatallah, Fabio Casati, Farouk Toumani: Web services conversation modeling: The Cornerstone for E-Business Automation. IEEE Internet Computing **8**(1) (January 2004)
3. Bultan, T., Fu, X., Hull, R., Su, J.: Conversation specification: a new approach to design and analysis of e-service composition. In: WWW 2003, ACM (May 2003) 403–410
4. Farouk Toumani, Boualem Benatallah, Fabio Casati: Analysis and Management of Web Services Protocols. DKE, Special issue from ER'04 (2004)
5. Benatallah, B., Casati, F., Toumani, F., Ponge, J., Nezhad, H.R.M.: Service mosaic: A model-driven framework for web services life-cycle management. IEEE Internet Computing **10**(4) (2006) 55–63
6. Rajeev Alur, P. Madhusudan: Decision problems for timed automata: A survey. In: 4th Intl. School on Formal Methods for Computer, Communication, and Software Systems. (2004)
7. Rajeev Alur, David L. Dill: A theory of timed automata. Theoretical Computer Science (126) (1994) 183–235

8. Ponge, J., Benatallah, B., Casati, F., Toumani, F.: Fine-grained Compatibility and Replaceability Analysis of Timed Web Service Protocols (extended version). http://www.isima.fr/~ponge/publications/tr/er07-extended.pdf (2007)
9. Beatrice Berard, Volker Diekert, Paul Gastin, Antoine Petit: Characterization of the expressive power of silent transitions in timed automata. Technical report, LIAFA Jussieu (1999)
10. Volker Diekert, Paul Gastin, Antoine Petit: Removing ε-transitions in timed automata. In: STACS'97
11. Boualem Benatallah, Fabio Casati, Daniela Grigori, Hamid R. Motahari Nezhad, Farouk Toumani: Developing Adapters for Web Services Integration. In: Proceedings of CAiSE 2005, Porto, Portugal. LNCS, Springer-Verlag (June 2005)
12. Bordeaux, L., Salaun, G., Berardi, D., Marcella, M.: When are two Web Services Compatible? In: VLDB TES'04, Toronto, Canada. (2004)
13. Beyer, D., Chakrabarti, A., Henzinger, T.A.: Web service interfaces. In: WWW'05, New York, NY, USA, ACM Press (2005) 148–159
14. Yellin, D., Storm, R.: Protocol Specifications and Component Adaptors. ACM Trans. Program. Lang. Syst. **19**(2) (March 1997) 292–333
15. Canal, C., Fuentes, L., Pimentel, E., Troya, J.M., Vallecillo, A.: Adding roles to corba objects. IEEE Trans. Softw. Eng. **29**(3) (2003) 242–260
16. Nezhad, H.R.M., Benatallah, B., Casati, F., Toumani, F.: Web services interoperability specifications. Computer **39**(5) (2006) 24–32
17. Boualem Benatallah, Fabio Casati, Julien Ponge, Farouk Toumani: Compatibility and replaceability analysis for timed web service protocols. In: BDA. (October 2005)
18. Rajeev Alur: Timed Automata. NATO-ASI 1998 Summer School on Verification of Digital and Hybrid Systems (1998)
19. Kongdenfha, W., Saint-Paul, R., Benatallah, B., Casati, F.: An aspect-oriented framework for service adaptation. In: ICSOC. (2006) 15–26

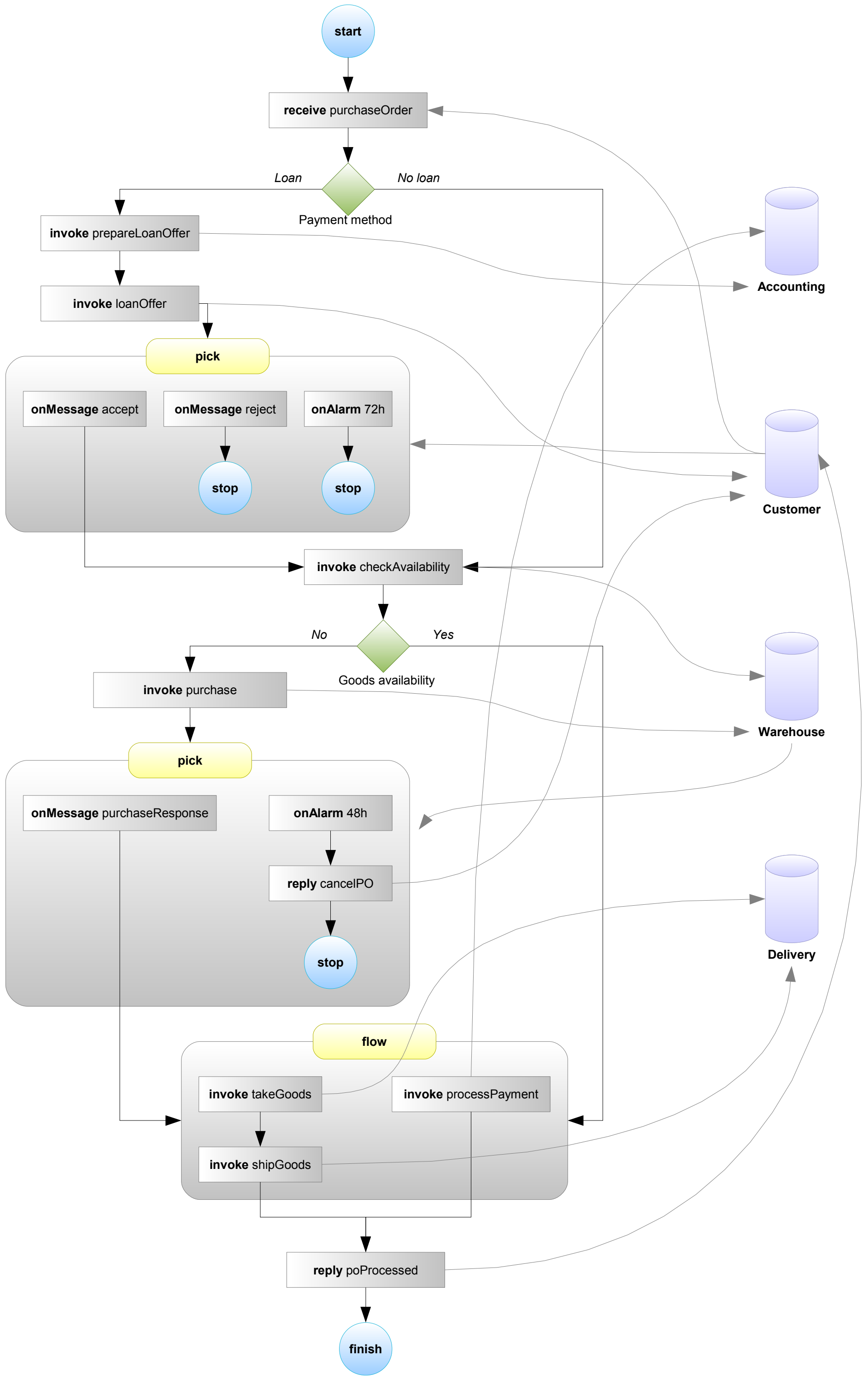


Fig. 7. A BPEL process that handles purchase order.

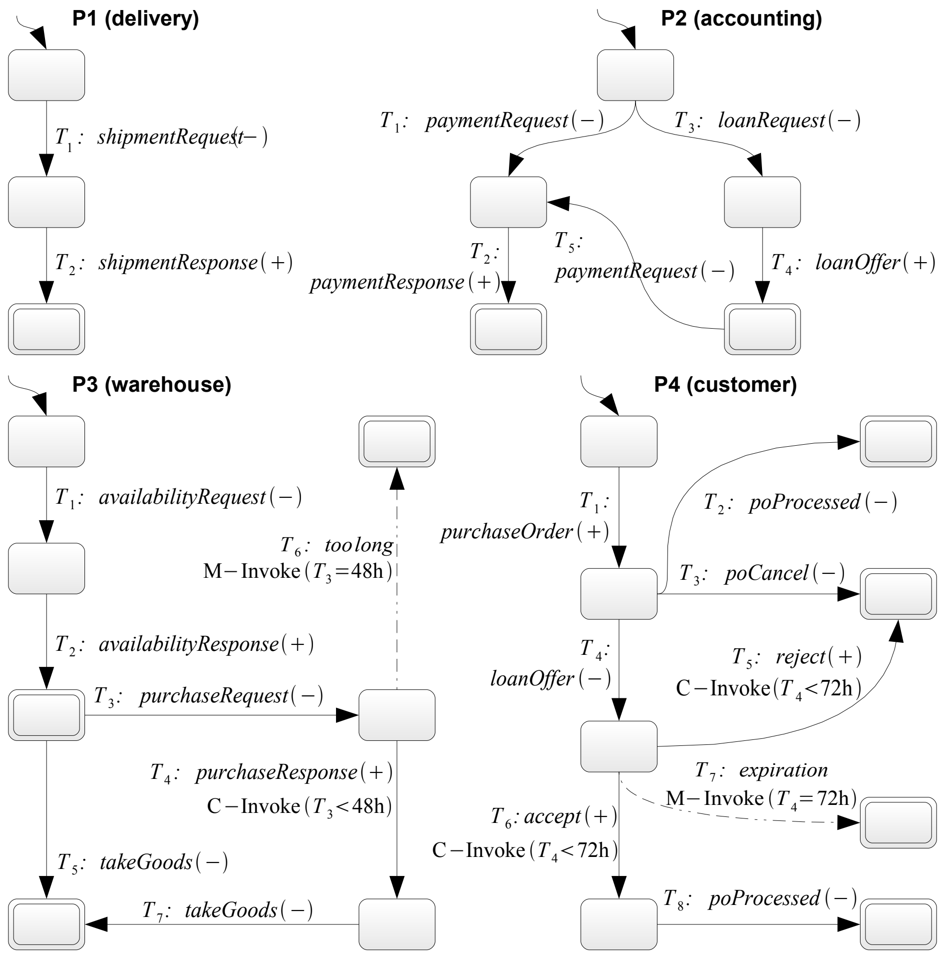


Fig. 8. Timed protocols extracted from the BPEL process of Figure 7.

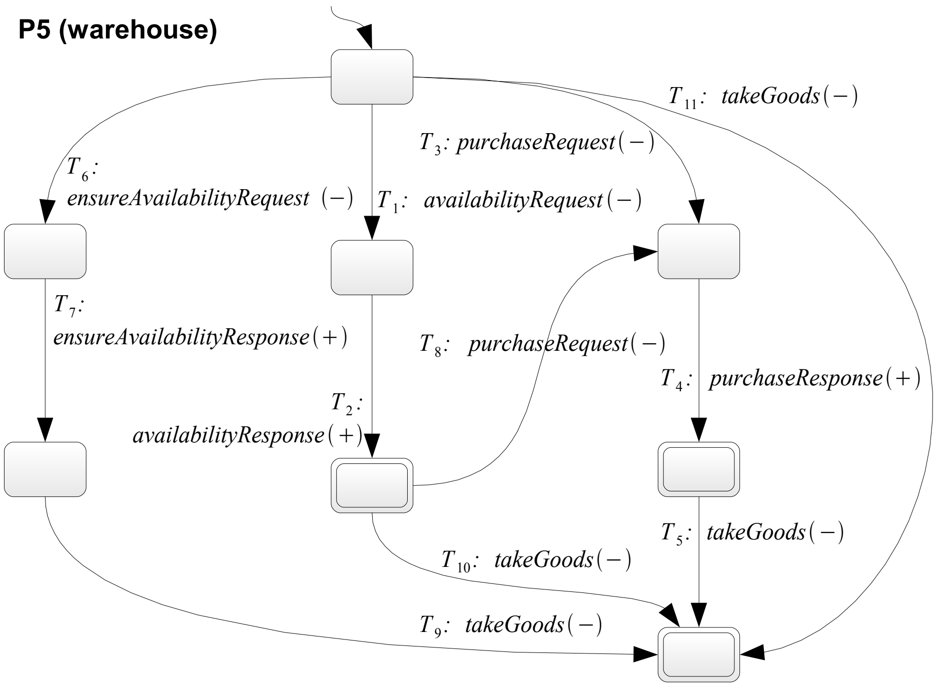

P5 (warehouse)
T_{11}: takeGoods(−)
T_3: purchaseRequest(−)
T_6: ensureAvailabilityRequest (−)
T_1: availabilityRequest(−)
T_7: ensureAvailabilityResponse(+)
T_8: purchaseRequest(−)
T_4: purchaseResponse(+)
T_2: availabilityResponse(+)
T_5: takeGoods(−)
T_{10}: takeGoods(−)
T_9: takeGoods(−)

Fig. 9. The complete warehouse service protocol.

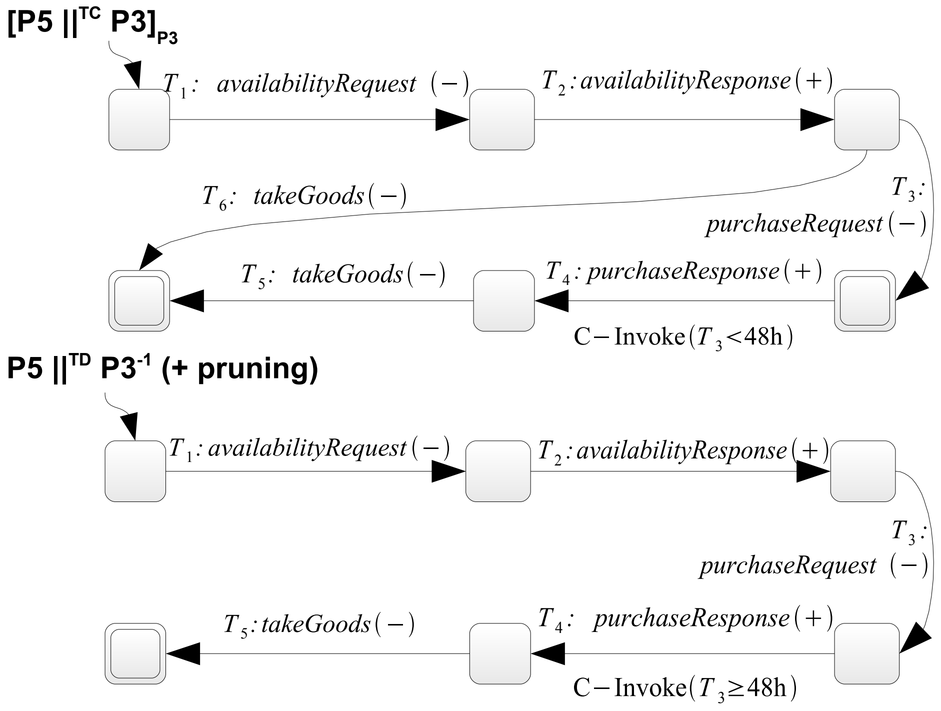


Fig. 10. Analysis of the common and differing conversations supported by P_3 and P_5.