CentER

Discussion Paper

No. 2009–05

A FORMAL FRAMEWORK FOR MULTI-PARTY BUSINESS PROTOCOLS

By Michele Mancioppi

January 2009

This is a revised version of CentER Discussion Paper No. 2008-79

September 2008

ISSN 0924-7815

TILBURG ◆ UNIVERSITY

A Formal Framework for Multi-Party Business Protocols*

Michele Mancioppi[1]

INFOLAB, Dept. of Information Systems and Management, Tilburg University, The Netherlands
m.mancioppi@uvt.nl

Abstract. Enterprise-class information systems based on the principles of Service Oriented Architecture comprise large numbers of long-running, highly dynamic complex end-to-end service interactions, called conversations, based on message exchanges that typically transcend several organizations and span several geographical locations. Conversations in service-based systems can be described using business protocols that are formal notations specifying the timed message exchanges among participants in a conversation from a local point of view (orchestrations) or global (choreographies). In this work we introduce a formal framework based on Deterministic Finite Automata enriched with temporal constraints to describe multi-party business protocols. We also explore the notion of multi-party business protocol soundness and show how it is possible to execute a multi-party protocol consistently in a completely distributed manner and at the same time ensure the progression of the execution (i.e. no "deadlocks").

Keywords: service oriented architecture, message exchange patterns business protocols, orchestrations, choreographies, soundness

JEL Classification: Y90

This is a revised version of the **CentER Discussion Paper 2008-79**

1 Introduction

The *Service Oriented Architecture* (SOA) is the approach to the design and implementation of information systems based on building applications out of software services. [1] *Services* are self-contained modules that provide functionalities to other services, applications or humans. [2] Services are provided (made accessible) over networks, where they can be discovered (located) and accessed.

Services communicate by exchanging messages according to complex patterns called *conversations*. Each service taking part in a conversation plays a *role*. The role defines how a particular service will generate and consume messages in the

*The research leading to these results has received funding from the European Community's Seventh Framework Programme under the Network of Excellence S-Cube - Grant Agreement n° 215483.

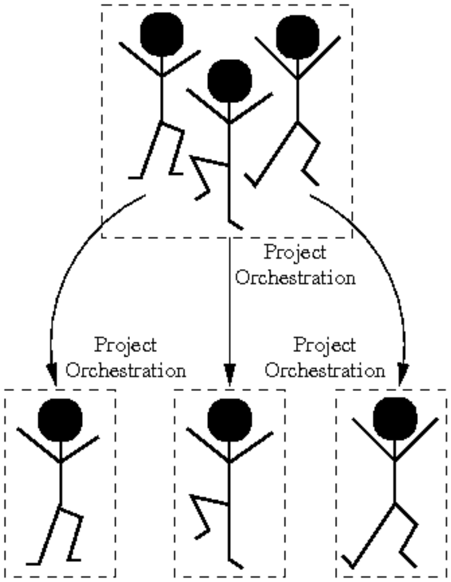


Fig. 1. Extracting Orchestrations from a Choreography

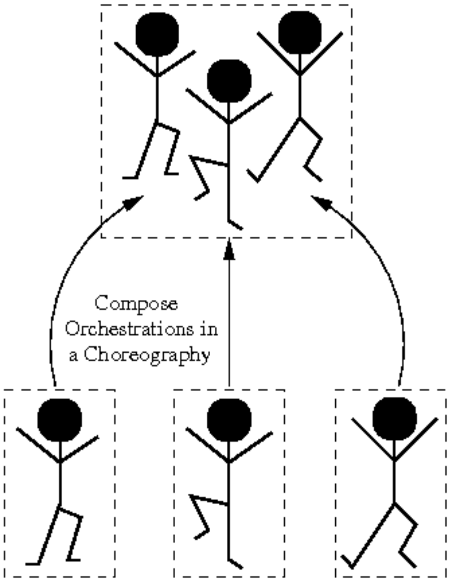


Fig. 2. Composing Orchestrations in a Choreography

context of the conversation. In SOA, conversations and roles can be respectively modeled as *choreographies* and *orchestrations*. Choreographies define the time-wise ordering of message exchanges occurring among all participants from a *global* perspective. Orchestrations restrict the overall choreographies to the *local* perspective of single participants.

As shown in Figure 1 and Figure 2[1], the global and local perspective provided by choreographies and orchestrations are deeply related. On the one hand, since orchestrations define a local perspective, they can be seen as *projections* of the global choreographies from the perspective of a given participant (Figure 1). That is, they represent the point of view of the participant on the overall conversation. On the other hand, choreographies can be interpreted as the *composition* of the orchestrations that define the local point of view of the participants (Figure 2).

The ability to describe conversations among services is instrumental to SOA. *Web services* are the current most relevant realization of SOA in enterprise-class information systems. Web services technologies realize services and SOA building on XML and open standards such as the *Simple Object Access Protocol* (SOAP) and the *Web Service Description Language* (WSDL). Over time, several languages for describing orchestrations and choreographies have been proposed in the ambit of web service technologies. Examples of such languages are *Web Services Business Process Execution Language* (WS-BPEL) [3], $BPEL^{light}$ [4], *BPEL4Chor* [5], *Yet Another Workflow Language* (YAWL) [6] and *Let's Dance* [7].

[1] The pictures are in reverent homage to Sir Arthur Conan Doyle and his wonderful "The Adventure of the Dancing Men".

Other proposals build on more formal approaches like π-calculus [8], Timed Automata [9, 10], *Deterministic Finite Automata* (DFA) [11, 12] and Petri-Nets [13, 14].

The languages currently available to describe orchestrations and choreographies can be grouped in *business protocol* and *business process* languages. Business protocol languages (e.g., Let's Dance) describe the interactions among web services by modeling the structure of the messages and the ordering of the message exchanges taking place among the participants of a conversation. Business process languages describe the way a service interacts with other services by providing an abstraction of its internal logic, usually with a workflow, where the generation and consumption of messages are just another type of task that the participant performs. Examples of business process languages are WS-BPEL and all its derivatives and YAWL. In a sense, business protocols have a black-box approach, whereas business processes have a white-box approach.

It is still unclear if either of the business protocol and business process approaches have clear advantages on the other, or whether they just apply to different SOA scenarios. On the one hand, business process languages such as BPEL^{light} intuitively seem to fit better web services realized through executable business process languages such as BPEL. To extract the orchestration implemented by the service, it would suffice to remove the activities in the workflows that do not affect the conversation. On the other hand, the black box approach of business protocol languages seem to better fit the loosely coupling and technology independence that characterize the Web service landscape, where it generally does not matter how a service it is implemented, but only how it appears to behave.

However, one thing appears certain: the tight relation between choreographies and orchestrations (i.e. compositions versus projections) suggest that there is much to gain from a comprehensive framework that covers both global and local perspectives, and that allows to move seamlessly from one to the other, and vice-versa. Except for a few cases such as [14], the approaches to business protocols proposed so far focus either on orchestrations or on choreographies. The present work aims at providing the foundation of a DFA-based business protocol framework embracing both global and local perspectives. We modified the DFA-based timed business protocols proposed by [10] to allow the description of both orchestrations and choreographies involving any number of participants.

The remainder is organized as follows. Section 2 introduces a taxonomy of the different kinds of business protocols we aim at supporting, the meta-model of the framework, and illustrates the semantics of executing business protocols. Section 3 illustrates a mapping from business protocol models to timed automata, which provides the foundation for model checking on business protocols. Section 4 models the *awareness* of the participants during the execution of a business protocol, i.e., what they know in terms of message exchanges, the state of the execution, etc. The awareness of the participants is instrumental to Section 5, which studies the *soundness* property in choreography-based business protocols. Sound choreography-based business protocols can be executed in a completely distributed way without incurring in deadlocks. Finally, Section 6 presents a summary of the work proposed, and our conclusions.

2 Business Protocols for Service Networks

The current section introduces our DFA-based meta-model for business protocols, which can describe both orchestration- and choreography- business protocols involving two or more participants. Our framework is a fundamental reworking and extension of the work presented in [10] to embrace multi-party business protocols and allow to describe choreographies as well as the already supported orchestrations.

Section 2.1 introduces a taxonomy of the business protocols that can be described with the meta-model. Section 2.2 presents the meta-model using Set Theory formalism, while Section 2.3 describes how to execute instances of the business protocol instances of models defined with our meta-model.

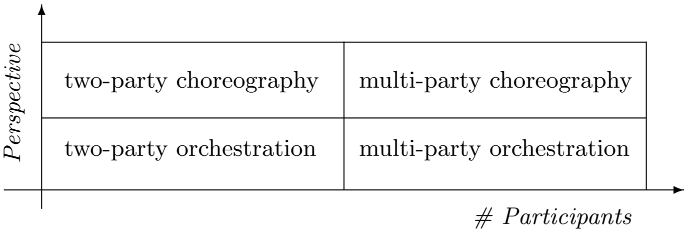
2.1 Taxonomy of Business Protocols

In our framework, business protocols are classified according to two dimensions:

- *Number of participants* involved in the conversations:
  two-party: two participants, or
  multi-party: (finitely) more than two participants
- The *perspective* adopted to describe the structure of the conversation:
  orchestration: the conversation describes the point of view of a particular participant, or
  choreography: the conversation describes all the possible interactions occurring among the participants from a neutral point of view.

The classification is based on on the combinations of the two parameters, as presented in Figure 3. The division between orchestration- and choreography-business protocols has already been introduced. Business protocols are also classified according to the number of their participants (i.e, two or more than two) because two-participant business protocols have particular properties (which are outside the scope of the present work).

This classification here proposed will be recurring in the remainder of the present work, as different classes of business protocols have different properties. For instance, business protocol soundness (see Section 5) is defined only for choreography business protocols.



two-party choreography	multi-party choreography
two-party orchestration	multi-party orchestration

Fig. 3. A taxonomy of the different types of business protocols.

2.2 Modeling Business Protocols

Business protocol models are DFA-based constructions structured as presented in Definition 21.

Definition 21 *A business protocol model B is defined as the tuple:*

$$B \ := \ (S, s_0, S_F, P, M, L, T_{id}, C, E, A)$$

such that:

$$S \neq \emptyset$$
$$s_0 \in S$$
$$\emptyset \neq S_F \subseteq S$$
$$C \subseteq C(T_{id}$$
$$E \subseteq T_{id} \times S \times S \times M \times L \times C$$
$$A \subseteq T_{id} \times S \times S \times C$$

S is the (not empty) set of the *states* of the business protocol B. Every business protocol has a unique x state s_0 and a non-empty set of final states S_F. Final states are *absorbing* (no outgoing transitions). The set P represents the participants involved in the business protocol. Different types of business protocols in the taxonomy (e.g., two-party orchestration business protocols) may define P differently. For instance, in the case of two-party orchestration models that need to describe the interaction between only two participants from the point of view of one of these, define "$P := \{\mathbf{me}, \mathbf{other}\}$" (see Section 2.2), while business protocols involving an arbitrary number of participants would go for a more general approach like "$P := \{p_i, i \in [1, n]\}$" (see, for instance, multi-party choreography models in Section 2.2).

The framework comprises two different types of transitions: *message-based* and *automatic*, respectively represented by the sets E and A. A transition is a directed connection from a *source* to a *target* state, and it is uniquely identified by a *transition identifier* in T_{id}. The message-based transition "$e := (t_{id}, s_i, s_j, m, l, c)$" is identified by t_{id} and connects the *source* state s_i with the *target* state s_j. It represents the delivery of the *message* m (comprised in the set M of the messages defined over the business protocol) among participants described by the *label* l. L is the set of labels defined by the various message-based transitions in the business protocol. Labels denote among whom the message is exchanged on the associated message-based transition. They make the framework general enough to describe all the types of business protocols listed in the taxonomy in Section 2.1: in fact, different types of business protocols need to structure labels differently (see Sections 2.2, 2.2, 2.2 and 2.2 for the details). Labels define a single *sender*, who sends the message, and a number of *recipients* that receive the message dispatched by the sender.

Each message in M is associated with exactly one message-based transition in the protocol. The messages defined in the business protocol model are actually message-types, such as XSD *complex type*, *simple type* or *element* definitions. All the message-types specified in a protocol B must be *disjoint*, meaning that, given an instance of a message exchanged during an execution of B, it can be univocally mapped to exactly one message-type in M. Message-based transitions are associated with time constraints (such as c in our previous example) that restrain when the transition can be traversed during executions of the business protocol. The structure of time constraints is illustrated later on in the Section.

The automatic transition "$a := (t_{id}, s_i, s_j, c)$" is identified by the transition identifier t_{id}, and it connects the source state s_i with the target state s_j. Automatic transitions are traversed as soon as the business protocol execution is in their source state and their *associated time constraint* (c in the example above) is verified.

Time constraints in C are boolean formulas that relate time durations and instants with the most recent time of execution of previously-traversed transitions. The transitions are represented in the time constraints by their transition identifiers. The time conditions that are actually specified on a particular model B are a subset of all the (generally infinitely many) time conditions that may be specified on B (hence the notation $C \subseteq C(T_{id})$ in Definition 21). The structure of time expressions in $C(T_{id})$ is presented in Definition 22.

Definition 22 Time Constraints
The time conditions $C(T_{id})$ (the set of all the possible time constraints that can defined in the business protocol B as in Definition 21) are defined by the following Backus–Naur form notation:

$$\begin{array}{lcl} \mathbf{C} & := & \neg\,\mathbf{C} \mid (\,\mathbf{C} \vee \mathbf{C}\,) \mid (\,\mathbf{C} \wedge \mathbf{C}\,) \mid \mathit{true} \\ & & \mid\; t_i \in T_{id}\ \mathbf{OP}\ \mathbf{TIME} \\ \mathbf{OP} & := & = \mid \neq \mid < \mid \leq \mid > \mid \geq \\ \mathbf{TIME} & := & \mathbf{DURATION} \mid \mathbf{INSTANT} \end{array}$$

Time conditions are made of atomic statements that are composed by the propositional logic operators "$\vee$" and "$\wedge$". The atomic statements are in the shape of either "*true*", or "$t_i \in T_{id}$ **OP TIME**" that compare the last time of the execution of a transition (identified by its transition identifier t_i) and a time. Times in atomic statements are either durations or instants, which are respectively represented by the non-terminal symbols **DURATION** and **INSTANT**.

The evaluation of composite time conditions follow the usual rules for propositional logics. The atomic condition "*true*" is a tautology. The evaluation of atomic time conditions in the form "$t_i \in T_{id}$ **OP**" is examined in Section 2.3.

Two-Party Orchestration Business Protocols

Definition 23 Two-Party Orchestration Models
A two-party orchestration business protocol B is defined as per Definition 21

with P and L being re-defined as follows:

$$P := \{\mathbf{me}, \mathbf{other}\}$$

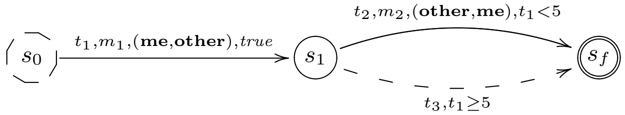
$$L \subseteq \{p \in P\} \times (P \setminus \{p\})$$

There are actually only two possible labels for two-party orchestration business protocols, namely:

$$(\mathbf{me}, \mathbf{other})$$

$$(\mathbf{other}, \mathbf{me})$$

Figure 4 presents a simple example of two-party orchestration business protocol model.



s_0
t_1,m_1,(**me**,**other**),*true*
s_1
t_2,m_2,(**other**,**me**),t_1<5
s_f
t_3,t_1≥5

Fig. 4. An example of two-party orchestration business protocol.

Labels for two-party orchestration business protocols may alternatively be defined using the concept of *polarity* [12, 15]. The polarity of a message exchange denotes the messages generated by the subject of the business protocol with $+$, and the messages consumed by the subject with $-$. That is, the polarity allows to render implicit participants' identifiers, which are no longer explicitly used to define senders and recipients of the messages.

The mapping between the labels and polarity is the following:

$$(\mathbf{me}, \mathbf{other}) \equiv (+)$$

$$(\mathbf{other}, \mathbf{me}) \equiv (-)$$

Figure 5 shows the business protocol in Figure 4 changed to use polarity instead of the labels.

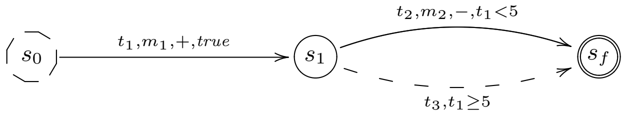


Fig. 5. The example in Figure 4 reworked using the polarity.

Two-Party Choreography Business Protocols

Definition 24 Two-Party Choreography Models
A two-party choreography business protocol B is defined as per Definition 21 with P and L are re-defined as follows:

$$P \quad \textit{with} \quad |\, P \,| = 2$$

$$L \quad \subseteq \quad \{p \in P\} \times (P \setminus \{p\})$$

Figure 6 presents the business protocol of Figure 4 expressed as a choreography instead of an orchestration, assigning the identifier p_1 to the subject, and p_2 to the other participant.

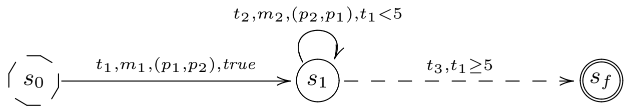


Fig. 6. An example of two-party choreography business protocol.

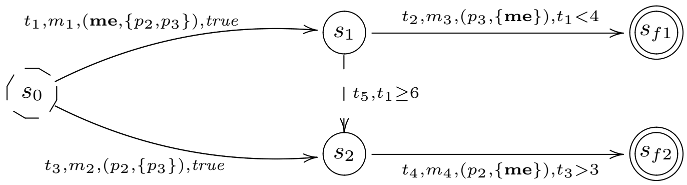
Multi-Party Orchestration Business Protocols

Definition 25 Multi-Party Orchestration Models
A multi-party orchestration business protocol B with n participants is defined as per Definition 21 with P and L are re-defined as follows:

$$P \quad := \quad \{\mathbf{me}\} \cup P' \ \textit{with} \ |\, P' \,| = \ n-1 \wedge \mathbf{me} \notin P'$$

$$L \quad \subseteq \quad (p \in P) \times (P \setminus \{p\})$$

Figure 7 presents an example of multi-party orchestration business protocol with three participants: p_2 and p_3, and the subject. In this case $P' := \{p_2, p_3\}$.



$t_1,m_1,(\mathbf{me},\{p_2,p_3\}),true$
s_1
$t_2,m_3,(p_3,\{\mathbf{me}\}),t_1<4$
s_{f1}
s_0
$t_5,t_1\geq6$
s_2
s_{f2}
$t_3,m_2,(p_2,\{p_3\}),true$
$t_4,m_4,(p_2,\{\mathbf{me}\}),t_3>3$

Fig. 7. An example of multi-party orchestration business protocol.

Multi-Party Choreography Business Protocols

Definition 26 Multi-Party Choreography Models
A multi-party orchestration business protocol B with n participants is defined as per Definition 21 with P and L are re-defined as follows:

$$P \quad with \quad | P | = n$$

$$L \quad \subseteq \quad (p \in P) \times (P \setminus \{p\})$$

Figure 8 presents an example of multi-party choreography business protocol with three participants: p_1, p_2 and p_3.

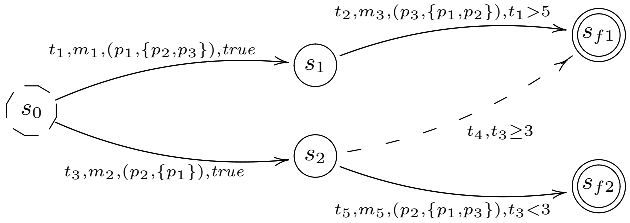

$t_2,m_3,(p_3,\{p_1,p_2\}),t_1>5$
$t_1,m_1,(p_1,\{p_2,p_3\}),true$
s_{f1}
s_1
s_0
$t_4,t_3\geq3$
s_2
$t_3,m_2,(p_2,\{p_1\}),true$
s_{f2}
$t_5,m_5,(p_2,\{p_1,p_3\}),t_3<3$

Fig. 8. An example of multi-party choreography business protocol.

2.3 Executing Business Protocols

The proposed business protocol framework is based on DFA and assumes a discrete time model. The execution of its models follows the usual rules for deterministic discrete finite automata [16, 17], plus the following additional rules due to the time constraints associated with the transitions:

- Message-based transitions can take place only when the associated time expression evaluates to *true*
- The execution of an automatic transition originating from the current state occurs as soon as its associated time expression evaluates to *true*. If a state has multiple automatic transitions, it might be the case that more than one of them becomes verified at the same time, and thus should be traversed. Only one racing transition can be traversed though. In such situation, the automatic transitions are said to be *racing*. To ensure determinism in the execution of the business protocol, the racing automatic transition traversed is always the one with the least transition identifier.

Consider the model presented in Figure 9. In this case, if an instance enters the state s_1 and p_2 does not generate the message m_2 within 5 units of time, then the automatic transitions t_3 and t_4 are racing, and t_3 is traversed because it has the least transition identifier.

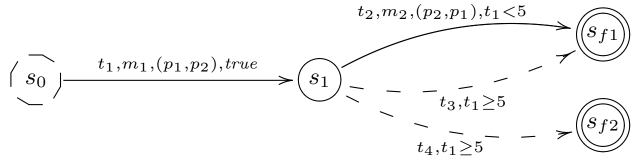


Fig. 9. An example of business protocol model with racing automatic transitions.

Execution Paths The execution of an instance evolves by traversing transitions. The order in which the transitions can be traversed depends from the structure of the model. The first transition traversed always originates in the initial state and must have the associated time condition "*true*". Every transition except the first must originate in the state that was the target of the previous traversed transition. The execution is over when a final state is reached, because the final states are required to be absorbing, and thus there is outgoing transition for the execution to traverse.

The execution instances of the same model can possibly traverse different sequences of transitions. For instance, consider the model presented in Figure 8. According to the messages that are generated by the participants, instances of this model can traverse three different sequences of transitions:

$$< \; t_1 \;\rightarrow\; t_2 \; >$$

$$< \; t_3 \;\rightarrow\; t_4 \; >$$

$$< \; t_3 \;\rightarrow\; t_5 \; >$$

The notation $t_1 \;\rightarrow\; t_2$ stands for "the transition t_1 is executed, followed by transition t_2". Each of these possible ways of traversing the transitions of a model is an *execution path*. Each execution path starts with a message-based transition that originates in the initial states, and ends with a transition that has a final state as target.

Execution paths of length n (i.e., made of the traversal of n transitions) defined on a generic business protocol B as per Definition 21 have the following general structure:

$$(T_{id})^n$$

That is, execution paths are tuples made of n transitions identifiers that represent the transitions traversed. We represent the generic execution path ex_B^n of length n defined on the business protocol B as:

$$ex_B^n \;:=\; t_{id}^1 \;\rightarrow\; \ldots \;\rightarrow\; t_{id}^n$$

using the $\rightarrow$ instead of commas (as it would usually apply for tuples) to better represent the idea of sequencing. The function $\sigma(ex_B^n, i)$ returns the i-th transition identifier mentioned in the execution path ex_B^n (see Section A for the definition of the function σ).

The set EX_B^i contains all the execution paths of length $i \in \mathbb{N}^+$ defined on the business protocol B. The set EX_B of all the possible execution paths of any length defined on the business protocol B is defined as:

$$EX_B := \bigcup_{i \in \mathbb{N}^+} R^i_B$$

Because of the DFA-based semantics of the business protocols, each execution path ex^n_B in EX_B satisfies the following constraints:

$$\textbf{source}(\sigma(ex^n_B, 1)) = s_0 \tag{1}$$

$$\forall\ 1 < i \leq n \in \mathbb{N}\ .\ \textbf{target}(\sigma(ex^n_B, i-1)) = \textbf{source}(\sigma(ex^n_B, i)) \tag{2}$$

$$n \in \mathbb{N}^+\infty \rightarrow \textbf{target}(\sigma(ex^n_B, n)) \in S_F \tag{3}$$

The function $\textbf{source}(t_{id})$ returns the source state of the transition identified by t_{id}, and $\textbf{target}(t_{id})$ returns the target state (see Section A for the definition of the functions **source** and **target**).

Condition 1 requires that the first transition traversed in the execution path originates in the initial state s_0. Condition 2 expresses that, given two subsequently traversed transitions in an execution path, the target state of the former is also the source state of the latter. Condition 3 requires that that last transition traversed in an execution path is a final state.

Since business protocol models expressed with our framework are ultimately DFA with a particular syntax for the labels and additional rules for the execution, Theorem 21 holds.

Theorem 21 Finite Execution Paths in Loop-less Models
A business protocol model B has a finite number of execution paths if and only if B does not have loops.

Proof. Directly from the fact that a business protocol model is structured as a DFA. The execution paths are simply the possible ways of traversing the DFA. It is well known that a DFA without loops has only a finite number of distinct paths in it.

The following Lemma 22 is an immediate result from Theorem 21.

Lemma 22 Infinitely Many Execution Paths in Models with Loops
If a business protocol model B has loops in it, there are infinitely many execution paths defined on B.

Proof. By negation of the hypothesis and result of Theorem 21.

Well-formed Business Protocol Models The meta-model resulting from Definitions 21 and 22 allow to build models that, taking into account the DFA-like intended semantics of the business protocol instances, are not acceptable. For instance, the meta-model does not formally constraint final states to be absorbing. Definition 27 sets the additional constraints that *well-formed* business protocol models must satisfy.

Definition 27 Well-Formed Models
The business protocol model B as per Definition 21 is well-formed if and only if it satisfies all the following propositions:

$$\neg \exists\, s_f \in S_F \,.\, \exists\, (t_{id}, s_f, s, m, l, c) \in E \vee\ \exists\, (t_{id}, s_f, s, c) \in A \tag{4}$$

$$\exists\, (t_{id}, s_0, s_i, m, l, c) \in E \wedge c = true \tag{5}$$

$$\forall s \in S\ \exists\, ex^n_B \in EX_B, i \in [1, n] \,.\, s \ =\ \sigma(ex^n_B, i) \tag{6}$$

Condition 4 prevents well-formed models from having transitions originating in a final state (i.e., final states must be absorbing). Condition 5 requires well-formed business protocol models to have at least one message-based transitions outgoing the initial state with the time-condition "*true*" (i.e., a tautology). This is needed to have the guaranteed that there is always a viable transition to begin an instance with. Finally, condition 6 requires that each state is "touched" by at least one execution path, that is the state is either target or source (or both) to at least one transition in the execution path. The rationale of condition 6 is to exclude from the well-formed models the ones with non reachable states. Non reachable states, while syntactically correct in DFA, are for all practical purposes useless for business protocols, and are thus disallowed. For instance, the model presented in Figure 10 is non well-formed because there is no execution path that touches the state s_3, thus contradicting condition 6. Similarly, the model in Figure 11 is not well-formed because it violates condition 6. In fact, there is no execution path touching the (final) state s_2 because the model has no execution paths at all, since the final state s_2 can never been reached from the initial state s_0.

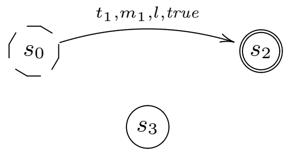


Fig. 10. A non-well formed model: the state s_3 is non-reachable.

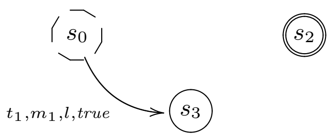


Fig. 11. A non-well formed model: the final state s_2 is non-reachable.

In the remainder we will take into account only well-formed models. Thus, unless contrary specified, whenever referring to a model, it will actually mean "well-formed model".

Runs and the Accepted Language of Business Protocol Models The execution paths on a model define in which order instances can traverse the transitions. However, the execution paths alone do not suffice for describing how instances behave over their executions because they do not capture *when* the transitions are traversed: that is, the time dimension of the execution.

The time-wise trace of the execution of a business protocol instance is called *run*. A run r_B on the business protocol B can be intuitively represented as:

$$r_B^n := < (t_{id}^1, \tau^1) \rightarrow \ldots \rightarrow (t_{id}^i, \tau^i) \rightarrow \ldots \rightarrow (t_{id}^n, \tau^n) >$$

A run is a sequence of *steps* (t_{id}^i, τ^i). Each step couples the identifier of a transition with the time that transition is traversed. The notation "$(t_{id}^i, \tau^i) \rightarrow (t_{id}^j, \tau^j)$" represents the traversing of the transition t_{id}^i at time τ^i, followed by the traversing of the transition t_{id}^j at time τ^j. The same transition identifier may appear in multiple steps of the run, in case of loops in the execution path followed by the instance. The formalization of the structure of runs is presented in Definition 28.

Definition 28 *A* run r_B^n *of* n *steps on the model* B *as in Definition 21 is defined as:*

$$r_B^n \in \{(t_{id} \in T_{id}, \tau \in \mathbb{T})\}^n, n \in \mathbb{N}^+$$

The times in the run's steps are reported as relative to the instant of traversing of the first transition in the run. Therefore, $\mathbb{T}$ in Definition 28 represents relative (discrete) times.

In the remainder of the work it will need to extract information from runs. The operator $\sigma(r^n, i)$ returns the i-th step of the run r^n (if $1 \leq i \leq n$, $\perp$ otherwise). The operator $\theta(r^n, i)$ returns the i-th state entered during the run r^n if $1 \leq i \leq n+1$. The 0-th state traversed by a run is always the initial state. The symbol $\perp$ is returned by $\theta(r^n, i)$ if $i \leq n+1$. Both the σ and θ operators are formally defined in Section A.

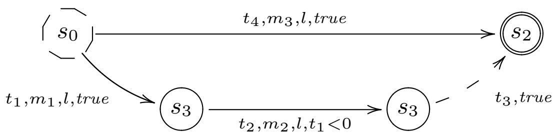


Fig. 12. A model with no accepting runs for some of its execution paths.

While intuitive, Definition 28 allows for runs that can never take place on a model. For instance, consider runs whose first step does not regard a transition originating from the initial state, or a run whose last step does not lead to a final state.

In a sense, the runs that can actually take place on a model are *instances* of the execution paths available on that model. The run r_B^n is an instance of the execution path ex_B^m, and we write $ex_B^m \succ r_B^n$ if and only if:

$$m = n \tag{7}$$

$$\forall i \in [i, n] \,.\, \sigma(r_B^n, i) = (t_{id}, \tau) \rightarrow \sigma(ex_B^n, i) = t_{id} \tag{8}$$

That is, a run is an instance of an execution path on a given model if and only if the run and the execution path have the same length (Condition 7), and the steps in the run regard transition identifiers mentioned in the same order as in the execution path (Condition 8).

A run r_B on the model B is said to be *well-formed* if and only if it is an instance of some execution path ex_B. Well-formed runs restrict the runs on a model to the ones that "make sense" from the point of view of traversing the business protocol model as a DFA. However, the definition of well-formed run does not consider the additional semantics of business protocols due to the time conditions associated with the transitions. For instance, consider the following well-formed run r on the model in Figure 12:

$$r :=< (t_1, 0) \rightarrow (t_2, 5) \rightarrow (t_3, 10) >$$

The run r can never result from the execution of the model in Figure 12 because the second step, $(t_2, 5)$, violates the time condition "$t_1 < 0$" associated with t_2.

An *accepting run* on a model is a well-formed run which can result from the execution of that model. For instance, consider the model presented in Figure 12. The model has two execution paths:

$$< \ t_1 \ \rightarrow \ t_2 \ \rightarrow \ t_3 \ >$$

$$< \ t_4 \ >$$

The execution path $< \ t_4 \ >$ has only one accepting run as instance, namely:

$$< \ (t_4, 0) \ >$$

Instead, the execution path $< \ t_1 \ \rightarrow \ t_2 \ \rightarrow \ t_3 \ >$ has no accepting runs associated with it, because the time condition $t_1 < 0$ on the transition t_2 can never be satisfied, since the time associated with a step in a run is always bigger or equal than 0 (and it is always 0 in case of the first step).

The set R_B^n is defined as the set of all the accepting runs of length n on B. Thus R_B, the set of all the accepting runs on B, is defined as:

$$R_B := \bigcup_{i \in \mathbb{N}^+} R_B^i$$

Borrowing a terminology from automata theory, R_B is also called the *accepted language* of B. In case B is not a trivial model, R_B has an infinite cardinality (i.e., it contains infinitely many elements)[2]. Thus, it makes little sense to devise algorithms to enumerate all the accepting runs on a given model. No matter the complexity of the algorithm, on the general case it would never terminate. Instead, it is crucial to be able to calculate the *inclusion* of languages of business protocol models: that is, given two models, check if the accepting runs supported by one are a sub-set, super-set or the same set of the other, or if the two sets of

[2] Actually, the cardinality of R_B, with B a non trivial model, is generally more than numerable (i.e., strictly more than the natural numbers). This result can be trivially obtained with Cantor's diagonal method, mapping natural numbers to the times in the runs' steps.

accepting runs are disjoint or only partially overlap. Another interesting problem connected to the accepted language is the *acceptance* of a run on a model: i.e., to verify whether a certain run is accepted by a certain model.

Both the problems of acceptance and inclusion of languages are well known in the literature covering DFA, and they are proven to be tractable [18]. However, the business protocols are not normal DFA because of the additional semantics contributed by the time conditions. Actually, they are much more similar to timed automata than to DFA. The problem of inclusion of languages for timed automata is proven to be intractable in the general case, but tractable for deterministic discrete timed automata, as it is tractable the acceptance problem (i.e. if a timed word is accepted by a given automaton). Section 3 presents a mapping from business protocol models to deterministic discrete timed automata that enables to study the acceptance and inclusion of languages problems for business protocols.

Actual Languages of Business Protocol Models The language R_B of a model B is build on the alphabet:

$$\Sigma := T_{id} \times \mathbb{T}$$

That is, the alphabet of business protocol model is made of couples of the transition identifiers of the transitions that are traversed and the time (relative to the traversing of the first run) at which the traversing occurs. This formulation of the language of a model is very intuitive, and will simplify later on the formulation of a mapping from business protocol models to timed automata (see Section 3). However, it does not immediately deliver the information of *which* messages are exchange *when* and *among which* participants.

This inconvenience is immediately solved by defining the concepts of *actual run* and *actual language* of a model B. An actual run on a model B is an accepting run of B where the transition identifiers are substituted by message and label or the ε symbol in case of automatic transitions. Formally, the actual run $\tilde{r}^n$ corresponding to the accepting run r^n on the model B is such that:

$$\forall i \in [1, n] \;.\; \sigma(\tilde{r}^n_B, i) = \begin{cases} (m, l) & \text{if } \sigma(r^n_B, i) = (t_{id}, \tau) \wedge \;\exists\, (t_{id}, s, s', m, l, c) \in E \\ \varepsilon & \text{otherwise} \end{cases}$$

The actual language $\tilde{R}_B$ of a model B is the set of the actual runs defined on it. $\tilde{R}_B$ is defined as:

$$\tilde{R}_B := \{\tilde{r}^n_B \;\mid\; r^n_B \in R_B\}$$

Notice that the mapping between accepted and actual runs it is not bijective, because all the automatic transitions are represented by the same symbol ε.

The applications of the actual language of a model, such as to *replaceability* and *compatibility* analyses, are beyond the scope of this work.

Representing Business Protocol Instances The meta-model proposed in Section 2.2 provides the syntax for modeling conversations. A model is an *instantiation* of the meta-model describing one particular conversations (i.e., complex pattern of message exchanges). A conversation described by a model is carried out by executing an instance of the model. The same conversation can be executed multiple times by having different instances on the same model. Each instance is associated with a run, that describes how the model is traversed by that particular instance. Different instances may traverse the model exactly in the same (i.e., they have the same run).

However, the run of an instance does not suffice to describe entirely how the messages where exchanged by that instance. In the run, the time associated with the traversing of the transitions is relative to the moment the run began (in fact, the first step has always associated with the time 0). Therefore, to completely describe a particular instance it is necessary to report also the absolute time (e.g., *2008-08-20 10:00:21Z*).

Formally, an instance i_B can be described as:

$$i_B := (B, r_B^n, \tau \in \mathbb{T}_{abs})$$

where B is the model from which i was instantiated, r_B^n is the run that i_B has executed, and τ is the absolute time (from the domain $\mathbb{T}_{abs}$) the run has commenced. Having the model B as component of the representation of i_B is meant to simplify the formalization of adaptation and migration strategies of instances (which is outside the scope of the present work).

Notice that this formalization assumes that the time of instantiation coincides with the time of the starting of the run. The time of the completion of the instance coincides with the time of the traversing of the last transition in the run r_B^n. The absolute time the traversing of the i-th step in r_B^n can be straightforwardly retrieved by combining τ and the relative time in $\sigma(r_B^n, i)$ (the relative time of the traversing i-th step in r_B^n).

Evaluating Atomic Time Conditions The execution of an instance requires the evaluation of the time conditions associated with the transitions. As explained in Section 2.2, the evaluation of composite time conditions follow the usual rules for propositional logics for combining the evaluation of the nested atomic time conditions, such as "$t_4 \leq 2$ hours". The atomic time conditions are in the form $t_i \in T_{id}$ **OP** (see Definition 22), and it is based on the data collected in the run up to the moment of the evaluation.

The evaluation of "$t_4 \leq 2$ hours" returns *true* if and only if the last traversing of transition t_4 has occurred within the past two hours. The last time of traversing of t_4 can be found by looking up the run backwards for the most recent step involving that transition. If t_4 is not found in the run (i.e., the transition has not yet been traversed), "$t_4 \leq 2$ hours" evaluates to **false**.

Logical Characterization of the Time Windows Associated to Transitions and States The time conditions associated with the transitions restrict when those transitions can be traversed during the execution of the model into *time-windows*. That is, a time-window is in a sense the “collection” of (relative) times at which one transition can be traversed in a model. Time-windows can be formally characterized as follows:

$$\begin{aligned} W_B(t_{id}) := \{(ex^n_B, \{\tau \mid \exists\, r^n_B \in R_B \;.\; ex^n_B \succ r^n_B \wedge\; \exists\, i \in [1,n] \;. \\ .\; \sigma(r^n_B, i) = (t_{id}, \tau)\})\} \end{aligned}$$

That is, the time-window $W_{t_{id}}$ on the model B is the set of all the couples $(ex^n_B, Time \subseteq \mathcal{T})$, where ex^n_B is an execution path of length n on B, and $Time$ is a set of possible (relative) instants at which t_{id} can be traversed in some run instance of ex^n_B.

Similarly for the case of the transitions, we can characterize the time-windows associated to the states that represent when an execution of the model can be in that state. The characterization differs in case the state is initial, final, or neither of the previous. In case the state s is neither initial nor final:

$$\begin{aligned} W_B(s) := \{(ex^n_B \in EX_B, \{\tau \mid \exists\, r^n_B \in R_B \;.\; ex^n_B \succ r^n_B \wedge\; \exists\, i \in [1,n] \;. \\ .\; \theta(r^n_B, i) = s \;\wedge\; \tau \in [\pi_\tau(\sigma(r^n_B, i-1)), \pi_\tau(\sigma(r^n_B, i))]\}\} \end{aligned}$$

where $\pi_\tau(t_{id}, \tau') = \tau'$ (that is, a simple projection operator to extract the time from a step of a run). That is, the time-window for a state is the collection of relative time instants (grouped by execution path) belonging to intervals in which a run is in that state. Similarly, we can define the time window for the initial state s_0:

$$\begin{aligned} W_B(s_0) := \{(ex^n_B \in EX_B, \{0\} \cup \{\tau \mid \exists\, r^n_B \in R_B \;.\; ex^n_B \succ r^n_B \wedge\; \exists\, i \in [1,n] \;. \\ .\; \theta(r^n_B, i) = s \;\wedge \tau \in [\pi_\tau(\sigma(r^n_B, i-1)), \pi_\tau(\sigma(r^n_B, i))]\}\} \end{aligned}$$

The time window for the initial state it is calculated in the same way for the one for any non-initial and non-final state, except that it always comprises the time 0 (for obvious reasons). Finally, we can define the time window for a state $s_f \in S_F$ as follows:

$$W_B(s_f) := \{(ex^n_B \in EX_B, \{\tau \mid \exists\, r^n_B \in R_B \;.\; ex^n_B \succ r^n_B \wedge \tau \geq \pi_\tau(\sigma(r^n_B, n))\}$$

Simply put, the time-window of a final state spans from the least instant you can reach that final state on (because once a final state is reached, it is never left).

3 Mapping Business Protocols to Discrete Timed Automata

The present section provides a mapping from business models to timed automata defined on a discrete time model, and explores some of the potential in terms of model checking capabilities that the mapping unlocks.

Definition 31 Equivalent Timed Automaton
The business protocol B as in Definition 21, maps to the Equivalent Timed Automaton *(ETA) $T_B := (S_T, s_T^0, S_T^f, X_T, E_T)$ on the alphabet $\Sigma_B := T_{id}$ defined as:*

$$
\begin{aligned}
S_T &:= S && (9)\\
\{s_0\} &:= s_T^0 && (10)\\
S_F &:= S_T^f && (11)\\
X_T &:= T_{id} && (12)\\
E_T &:= \{(s_i, s_j, (t_{id}, m, l), \{t_{id}\}, c) \;\mid\; \exists\, (t_{id}, s_i, s_j, m, l, c) \in E\} \cup \\
&\quad \cup \{(s_i, s_j, t_{id}, \{t_{id}\}, c \wedge \bigwedge_{\{c' \mid \exists\, (t'_{id}, s_i, s, c') \in A \wedge t'_{id} < t_{id}\}} \neg c') \;\mid \\
&\quad \mid\; \exists\, (t_{id}, s_i, s_j, c) \in A\} && (13)
\end{aligned}
$$

The Clause 9 of Definition 31 means that the states of the equivalent timed automaton are the same as the states of the business protocol model. Similarly, Clause 10 and 11 say that the states that are initial or final in the business protocol model are the same in the ETA. Clause 12 defines a clock in ETA for each transition identifier in the business protocol model, which is needed because since we need to keep track of the last occurrence of every transition in order to evaluate the time conditions. Finally, Clause 13 defines the transitions in the ETA, which come from message-based and automatic transitions in the model. Clause 13 can be split in two parts:

$$\{(s_i, s_j, t_{id}, \{t_{id}\}, c) \;\mid\; \exists\, (t_{id}, s_i, s_j, m, l, c) \in E\} \tag{14}$$

$$
\begin{aligned}
&\{(s_i, s_j, t_{id}, \{t_{id}\}, c \wedge \bigwedge_{\{c' \mid \exists\, (t_{id}, s_i, s, c') \in A \wedge t'_{id} < t_{id}\}} \neg c') \;\mid \\
&\quad \mid\; \exists (t'_{id}, s_i, s_j, c) \in A\}
\end{aligned} \tag{15}
$$

Clause 14 creates a transition in the ETA per each message-based transition in the model. The transition in the ETA resets the clock corresponding to the message-based transition, and it consumes the transition identifier as symbol. The time condition associated with the transition in the ETA is the same as the one associated to the message-based transition in the model. Clause 15, similarly to Clause 14, creates a transition in the ETA for every automatic transition in the model. Likewise the case for message-based transitions, the transition in the ETA resets the clock corresponding to the automatic transition in the model, and consumes as symbol the automatic transition's identifier. However, the definition of the time condition for the transition in the ETA corresponding to an automatic transition in the model is more complicated, because it must encode a part of the semantics of the business protocols that has no immediate correspondence in timed automata: if the time conditions of multiple automatic transitions become verified at the same time, the transition with the least transition identifier is traversed. To represent this execution rule in the ETA, it is necessary to rework the time conditions of transitions corresponding to automatic transitions in the model as the conjunction of the time condition c on the automatic transition and the negation of all the time conditions associated to other automatic transitions in the model that originate in the same state, and whose transition identifier is smaller, which are contained in the set:

$$\{c' \ \mid \ \exists\,(t'_{id}, s_i, s, c') \in A \wedge t'_{id} < t_{id}\}$$

Consider the business protocol model and its ETA presented respectively in Figure 13 and Figure 14. The model in Figure 13 is *generic* (i.e., as defined in Definition 21, and abstracting the content of the labels), from brevity and to underline that the definition of ETAs is not restricted to any particular sub-type of business protocol, such as two-party orchestration or multi-party choreography. Notice that, in the ETA, the transition identifiers appearing a the beginning of the labeling of the transitions it is not supposed to represent an identifier for those transitions (as in the business protocol models), but instead it represents the *symbol* that is consumed by the ETA upon the traversing of the transition.

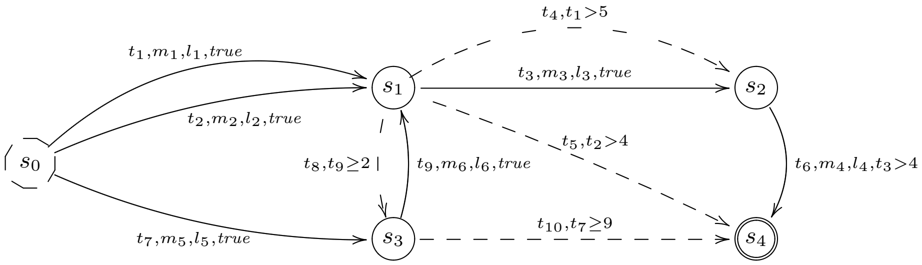


Fig. 13. A business protocol model.

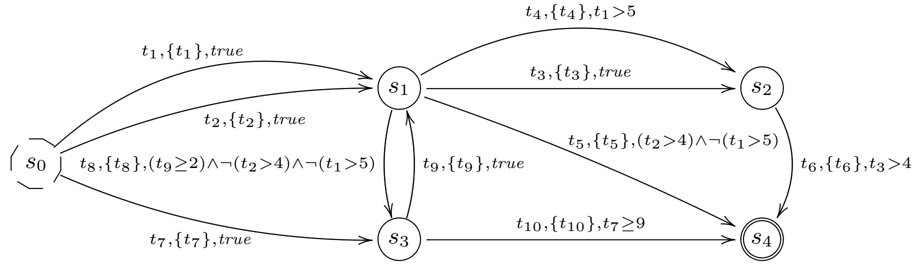


Fig. 14. The ETA corresponding to the model in Figure 13.

Likewise with business protocol models, in the graphical representation of the ETA the source and target states are omitted from the transitions, as they are already graphically represented. The time condition associated with the transition that consumes the symbol t_5 in the ETA is obtained by making the conjunction with the time condition "$t_2 > 4$" associated with t_5 in the model with "$\neg(t_1 > 5)$", which is the negation of the time condition "$t_1 > 5$" associated with the automatic transition t_4 in the model, which originates in the same state s_1 as t_5. Similarly, the time condition "$(t_9 \geq 2) \wedge \neg(t_2 > 4) \wedge \neg(t_1 > 5)$" on the transition t_9 in the ETA is obtained by making the conjunction of the condition on t_8 and the negation of the conditions on t_4 and t_5 in the model.

The business protocol models are deterministic, and therefore the respective ETA also are. Notice that the mapping from models to ETA is not invertible, because it losses the information on the message-based transitions, as well as the distinction of which transitions are message-based and which not.

The key result revolving around ETA is presented in Lemma 31.

Lemma 31 Correspondence of the Language of Models and ETAs *The accepted languages of the business protocol model B and its ETA T_B are exactly the same.*

Proof. Straightforward by the construction of the ETA.

In practice, Lemma 31 guarantees us that if a (timed) property holds on the accepted language of an ETA, it also holds on the accepted language of the respective model, because they are exactly the same. Therefore, all the model-checking techniques that have been devised for verifying timed properties on the accepted languages of (deterministic discrete) timed automata are also applicable to business protocol models through their ETA. The mapping here proposed enables the use of ACTL* [17]. This allows, for instance, to verify whether all the possible accepted runs of a model complete within a certain amount of time, or whether a certain transition is never traversed before another.

The equivalence of the languages of a business protocol model and its ETA allows to use *Timed Regular Expressions*[19, 20] or *Data Languages*[21] in conjunction with business protocols, which may express in a compact way the expressiveness of business protocol models. However, the relation between business protocols, Timed Regular Expressions and Data Languages is future work.

4 Participant Awareness in Choreography-Based Business Protocols

Business protocols model the message-based interactions occurring among distributed participants. The participants are expected to generate and consume messages only in particular time-windows defined by the time conditions associated with the message-based transitions. The generation of messages outside the legitimate time-windows make the business protocol instances fail. Participants can generate and consume messages at the right moment only if they are able to keep track of the current state of the business protocol instances they participate in. The capability of participants to follow the execution of the instance through traversing of transitions and entering and leaving states is called *awareness*.

The awareness of participants is further split into *state-awareness* and *transition-awareness*. Intuitively, a participant is state-aware of a given state if, every time that state is entered or left, the participant knows. Similarly, a participant is aware of a given transition if every time that transition is traversed, the participant knows. In the following text, the word "awareness" will be used instead state-awareness or transition-awareness in case the context makes clear which kind of awareness is taken into account.

Transition- and state-awareness are tightly related. On the one hand, in order to be aware of a state, a participant needs to be aware of the transitions that lead to and leave from that state. On the other hand, since automatic transitions are not *observable*, i.e., participants do not get notification of their occurring, unlike the message-based transitions where the notification is the message itself. Participants have to infer that an automatic transition is traversed by excluding the traversing of message-based or automatic transitions originating from the same state. That is, in order to be aware of an automatic transition, the participant must be aware of the transition's source state.

4.1 Transition-Awareness

The intuition of transition awareness for message-based transitions is captured formally in Definition 41.

Definition 41 Transition-Awareness: Message-Based Transitions *The participant p in the choreography model B is aware of the message-based transition $(t_{id}, s, s', m, (p_{send}, P_{rec}), c)$, and we write $\textbf{aware}_m(p, e)$, if and only if:*

$$\textbf{aware}_m(p \in P, (t_{id}, s, s', m, (p_{send}, P_{rec}), c) \in E) \leftrightarrow p = p_{send} \vee p \in P_{rec}$$

In effect, Definition 41 says that a participant is aware of a message-based transition if and only if it is either the sender or one of the recipients in that transition.

Definition 41 does not apply to automatic transitions. The notion of awareness with respect to automatic transitions is more complicated to formulate. Participants infer the occurrence of an automatic transition if the run has been in the source state of that transition long enough the time condition associated with that automatic transition. In order to do this, a participant has to:

- be able to observe all the message-based transitions outgoing the state (i.e., being transition aware of them);
- be able to evaluate the time conditions associated with all the automatic transitions outgoing the state, so to be sure that the automatic transition they infer is the first one to become satisfied.

Because of the second condition, the awareness with respect to an automatic transition relies on being able to evaluate the time conditions associated with the transitions. We do not have yet the instruments to formally define if a participant can evaluate a particular time condition (we will see later on that this definition is mutually recursively defined on the basis of awareness). But we can postulate the existence of the function $\mathbf{can\_eval}(p \in P, c \in C)$ that returns *true* if and only if the participant p can evaluate the time condition c, and build on top of it the formalization of transition awareness with respect to automatic transitions, which is presented in Definition 43.

Definition 42 (Temporary) Transition-Awareness: Automatic Transitions
The participant p is aware of the automatic transition a in the multi-party choreography model B, and we write $\mathbf{aware}_a(p, a)$, if and only if:

$$\begin{aligned}\mathbf{aware}_a(p,a) \leftrightarrow\ & \mathbf{source}(a) = s \wedge (\forall e \in (\mathbf{in}(s) \cap E) \ .\ \mathbf{aware}_m(p,e)) \wedge \\ & \wedge (\forall a \in (\mathbf{in}(s) \cap A) \ .\ \mathbf{aware}_a(p,a)) \wedge \\ & \wedge (\forall e \in (\mathbf{out}(s) \cap E) \ .\ \mathbf{aware}_m(p,e)) \wedge \\ & \wedge (\forall a := (t_{id}, s, s', c) \in (\mathbf{out}(s) \cap E) \ .\ \mathbf{can\_eval}(p,c)\end{aligned}$$

Breaking down Definition 42 in four part, it says that a participant p is transition aware of an automatic transition a with source state s if and only if:

1. $\forall e \in (\mathbf{in}(s) \cap E) \ .\ \mathbf{aware}_m(p,e)$: p is aware of all the message-based transitions incoming to the state s;
2. $\forall a \in (\mathbf{in}(s) \cap A) \ .\ \mathbf{aware}_a(p,a)$: p is aware of all the automatic transitions incoming to the state s;
3. $\forall e \in (\mathbf{out}(s) \cap E) \ .\ \mathbf{aware}_m(p,e)$: p is aware of all the message-based transitions outgoing to the state s;
4. $\forall a := (t_{id}, s, s', c) \in (\mathbf{out}(s) \cap E) \ .\ \mathbf{can\_eval}(p,c)$: p can evaluate the time constraints associated with all the automatic transitions outgoing s.

While capturing the intuition of transition awareness for automatic transitions, Definition 42 has two technical defects (and this is why it is just a "temporary" definition). First of all, it is *circular* in case of cycles in the model. For instance, consider the model in Figure 15: to be aware of the automatic transition t_3, the participant p_2 must be aware of all the automatic transitions incoming to the state s_1, i.e. t_3 itself. A more complicated example of circularity is shown in Figure 16, where p_1 is transition-aware of t_2 if it is also aware of t_3, and vice-versa. An even more complicated example of circularity is shown in Figure 17, where p is aware of all t_2, t_3 and t_4, or none.

The second defect of Definition 42 is that it does not take into account message-based transitions looping on a state. For instance, consider the model presented in Figure 18. Even if p_1 is not aware of t_2, intuitively p_1 is able to determine when the automatic transition t_3 is traversed, because traversing t_2 does not change the state of the execution.

The circularity of Definition 42 affects models that have cycles made of automatic transitions, which are called *automatic cycles*. Formally, an automatic cycle ϕ^n of length $n \in \mathbb{N}^+$ on a multi-party choreography model B as in Definition 26 is defined as:

$$\phi^n := < a_1, \ldots, a_n > \ ,\ \forall i \in [1,n] \ .\ a_i \in A \wedge \forall i,j \in [1,n] \ .\ i \neq j \rightarrow a_i \neq a_j$$

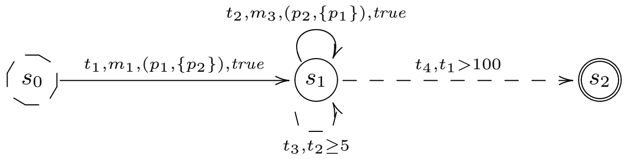


Fig. 15. A model in which the awareness of p_1 with respect to t_3 is affected by circularity.

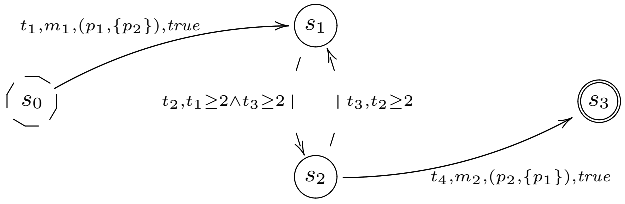


Fig. 16. A model in which the awareness of p_1 with respect to both t_2 and t_3 is affected by circularity.

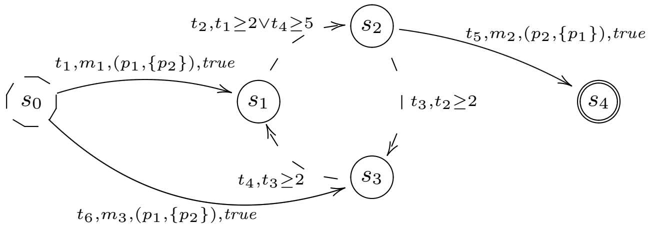


Fig. 17. A model in which the awareness of p_1 with respect to t_2, t_3 and t_4 is affected by circularity.

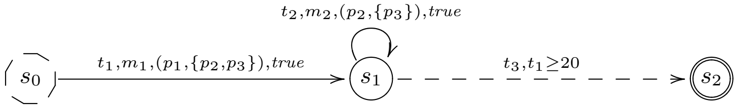


Fig. 18. An example of looping message-based transition.

$$\forall i \in [1, n-1] \ . \ \mathbf{target}(a_i) = \mathbf{source}(a_{i+1}) \wedge \mathbf{target}(a_n) = \mathbf{source}(a_1)$$

We write $a \in \phi^n := < a_1, \ldots, a_n >$ if:

$$\exists \ i \in [1, n] \ . \ a_i = a$$

Definition 43 is obtained by modifying Definition 42 to tackle the issues with circularity and looping message-based transitions.

Definition 43 Transition-Awareness: Automatic Transitions
The participant p is aware of the automatic transition a with source state s in the multi-party choreography model B, and we write $\mathbf{aware}_a(p, a)$, if and only if:

$$\begin{aligned}\mathbf{aware}_a(p,a) := {} & (\forall e \in (\mathbf{in}(s) \cap E) \, . \, \mathbf{target}(e) \neq s \rightarrow \mathbf{aware}_m(p,e)) \wedge \\ & \wedge (\forall a' \in (\mathbf{in}(s) \cap A) \, . \, (\nexists \, \phi \mid a \in \phi \wedge a' \in \phi) \rightarrow \mathbf{aware}_a(p,a')) \wedge \\ & \wedge (\forall e \in (\mathbf{out}(s) \cap E) \, . \, \mathbf{source}(e) \neq s \rightarrow \mathbf{aware}_m(p,e)) \wedge \\ & \wedge (\forall a' := (t_{id}, s, s', c) \in (\mathbf{out}(s) \cap E) \, . \, \mathbf{can\_eval}(p,c))\end{aligned}$$

That is, the circularity is removed by excluding the automatic transitions that lay on cycles from ones inbound to s the participant p has to be aware of. The problem of looping message-based transitions is tackled by requiring the transition awareness only of the non-looping ones.

Lemma 41 is an immediate result from Definition 43:

Lemma 41 *A participant p is aware of an automatic transition a originating from a state s if and only it is aware of all the automatic transitions originating from s.*

Proof. Straightforward when noticing that Definition 43 has no direct reference to a other than to retrieve its source state s. After that, all the automatic transitions originating from s, a comprised, are treated the same through the universal quantification in:

$$\ldots \forall a' := (t_{id}, s, s', c) \in (\mathbf{out}(s) \cap E) \, . \, \mathbf{can\_eval}(p,c) \ldots$$

Therefore, if p is aware of an automatic transition originating from a state, it is also aware of all the other automatic transitions originating from the same state.

For simplicity, we can combine the definition of awareness for message-based and automatic transitions as shown in Definition 44.

Definition 44 Transition-Awareness
The participant p is aware of the transition t in $E \cup A$ in the multi-party choreography model B, and we write $\mathbf{aware}_t(p, t)$, if and only if:

$$\mathbf{aware}_t(p \in P, t \in E \cup A) := \begin{cases} \mathbf{aware}_m(p,t) & \mathit{if}\, t \in E \\ \mathbf{aware}_a(p,t) & \mathit{if}\, t \in A \\ \mathbf{false} & \mathit{otherwise} \end{cases}$$

To complete the definition of transition-awareness, we need to provide a definition of the function $\mathbf{can\_eval}(p \in P, c \in C)$. Time conditions are combinations through predicate logic of atomic time conditions based on the last occurrence of a transition. Thus, a participant must be aware of all the transitions whose identifiers are mentioned in a time condition to be able to evaluate it. The function **extract** returns the set of all the transition identifiers that appear nested in the time condition c, and it is formally defined in Definition 45.

Definition 45 *The function* **extract**$(c \in C) \rightarrow 2^{T_{id}}$ *(where* $2^{T_{id}}$ *is the powerset of* T_{id}*) is defined on the structure of time conditions (see Definition 22) as follows:*

$$\mathbf{extract}(c) := \begin{cases} \mathbf{extract}(c_1) & \textit{if } c = \neg\, c_1 \\ \mathbf{extract}(c_1) \cup \mathbf{extract}(c_2) & \textit{if } c = (c_1 \vee c_2) \\ \mathbf{extract}(c_1) \cup \mathbf{extract}(c_2) & \textit{if } c = (c_1 \wedge c_2) \\ \{t_{id}\} & \textit{if } c = t_{id}\ \mathbf{OP}\ k \\ \emptyset & \textit{if } c = \textit{true} \end{cases}$$

Using the function **extract**, it is possible to define the function **can_eval** as shown in Definition 46.

Definition 46 *The participant* $p \in P$ *can evaluate the time condition* $c \in C$ *if and only if:*

$$\mathbf{can\_eval}(p, c) := (\forall t_{id} \in \mathbf{extract}(c)\ .\ \mathbf{aware}_t(p, \mathbf{deref}(t_{id})))$$

where the function **deref**$(t_{id} \in T_{id}) \rightarrow (E \cup A \cup \{\bot\})$ *returns the transition identified by* t_{id} *and it is defined as:*

$$\mathbf{deref}(t_{id}) := \begin{cases} e & \textit{if } \exists\, e := (t_{id}, s, s', (p_{send}, R_{rec}), c) \in E \\ a & \textit{if } \exists\, a := (t_{id}, c) \in A \\ \bot & \textit{otherwise} \end{cases}$$

The function **can_eval**$(p \in P, c \in C)$ returns *true* if the participant p can evaluate the time condition c, and **false** otherwise.

4.2 State-Awareness

A participant is state-aware of a state if, every time the state is entered or left, the participant knows it. That is, a participant is aware of a state is it is aware of all the transitions incoming to and outgoing from that state. Formally:

Definition 47 State-Awareness
The participant p is aware of the state s in the multi-party choreography model B, and we write **aware**$_s(p, s)$*, if and only if:*

$$\mathbf{aware}_s(p, s) := \forall t \in (\mathbf{in}(s) \cup \mathbf{out}(s))\ .\ \mathbf{source}(t) \neq \mathbf{target}(t) \rightarrow \mathbf{aware}_t(p, t)$$

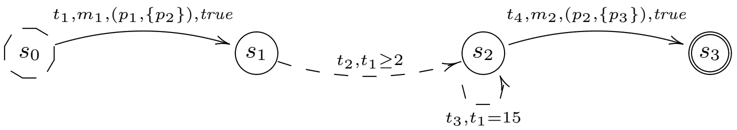
Definition 47 captures the intuition that, is a participant is aware of all the transitions that can lead to and move away from a state, the participant will always know if the execution is in that state. Notice that, likewise awareness with respect to automatic transitions, transitions looping on the state are not considered, because traversing looping transitions do not cause the current state to change.

Definition 47 is very similar to Definition 43. They both require the participant to be aware of all the transitions incoming to a state, and aware of all the transitions outgoing from it. Theorem 42 relates the awareness of a participant with respect to an automatic transition originating from a certain state, and the awareness of the same participant with respect to that state.

Theorem 42 *If the participant* p *is aware of the automatic transition* a *in the multi-party choreography business protocol model* B*, then* p *is aware of the source state of* a*. Formally:*

$$\mathbf{aware}_a(p, a) \rightarrow \mathbf{aware}_s(p, \mathbf{source}(a))$$

Proof. By Lemma 41, if p is aware of a, then p is also aware of all the other automatic transitions originating from the same state $s := \mathbf{source}(a)$. Due to the definition of awareness for automatic transitions, p is also aware of all the transitions incoming to s and the message-based transitions outgoing s. Since p is state aware of all the transition incoming to s, and all message-based and automatic transitions outgoing s, that is, *all* the transitions outgoing s, then p is state-aware of s due to Definition 47. The opposite implication does not hold, as shown in the counter-example in Figure 19. In this model, p_1 is aware of s_2, but not of t_3. The transition-awareness of t_3 is not required for the state-awareness of s_2 because t_3 loops on it.



t_1,m_1,(p_1,{p_2}),true
s_0
s_1
t_2,t_1≥2
s_2
t_4,m_2,(p_2,{p_3}),true
s_3
t_3,t_1=15

Fig. 19. A counter-example for "$\mathbf{aware}_s(p, \mathbf{source}(a)) \rightarrow \mathbf{aware}_a(p, a)$".

4.3 Paths of Awareness

The definition of awareness with respect to an automatic transition is recursive. In particular, in order for a participant to be aware of an automatic transition originating from a state s, p must also be aware of all the automatic transitions entering s. Moreover, state-awareness is defined on the basis of transition-awareness. This creates a "chain" effect for awareness in execution paths. For instance, consider the multi-party choreography model presented in Figure 20. Table 1 reports which participants are aware of which transitions in the model in Figure 20. Figure 21 represents the same model as Figure 20, omitting the details of the transitions and highlighting with double lines the transitions that p_1 is aware of.

Consider the following execution path on the model in Figure 20:

$$ex_1 :=< t_1 \rightarrow t_3 \rightarrow t_6 \rightarrow t_7 \rightarrow t_6 \rightarrow t_8 \rightarrow t_{10} >$$

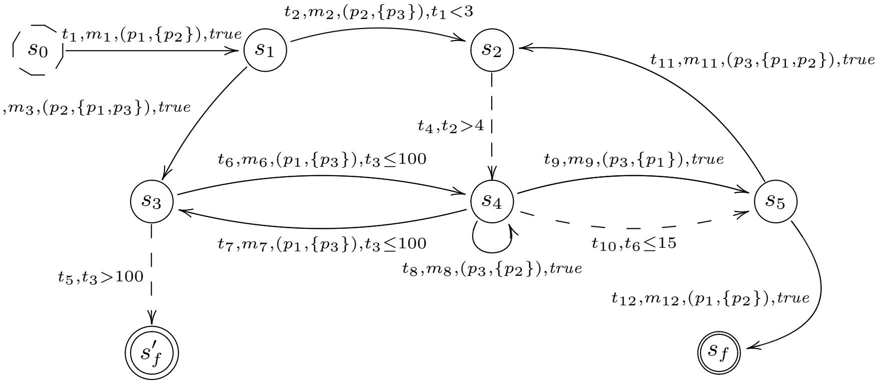

s_0
$t_1,m_1,(p_1,\{p_2\}),true$
s_1
$t_2,m_2,(p_2,\{p_3\}),t_1<3$
s_2
$t_{11},m_{11},(p_3,\{p_1,p_2\}),true$
$t_3,m_3,(p_2,\{p_1,p_3\}),true$
$t_4,t_2>4$
$t_6,m_6,(p_1,\{p_3\}),t_3\leq 100$
$t_9,m_9,(p_3,\{p_1\}),true$
s_3
s_4
s_5
$t_7,m_7,(p_1,\{p_3\}),t_3\leq 100$
$t_{10},t_6\leq 15$
$t_8,m_8,(p_3,\{p_2\}),true$
$t_5,t_3>100$
$t_{12},m_{12},(p_1,\{p_2\}),true$
s'_f
s_f

Fig. 20. A multi-party choreography model.

The participant p_1 is aware of all the transitions in the ex_1. Instead, consider the following execution path:

$$ex_2 :=< t_1 \rightarrow t_2 \rightarrow t_4 \rightarrow t_7 \rightarrow t_6 \rightarrow t_9 \rightarrow t_{10} \rightarrow t_4 \rightarrow t_7 \rightarrow t_6 \rightarrow t_8 \rightarrow t_{11} >$$

In ex_2, the participant p_1 is aware of only some of the transitions. By hiding with the symbol "?" the sub-sequences of transitions p_1 is not aware of from the execution path ex_2, we have the following:

$$ex_2^{p_1} :=< t_1 \rightarrow ? \rightarrow t_7 \rightarrow t_6 \rightarrow ? \rightarrow t_{10} \rightarrow ? \rightarrow t_7 \rightarrow t_6 \rightarrow t_8 \rightarrow t_{11} >$$

$ex_2^{p_1}$ is the *path of awareness* of participant p_1 regarding the execution path ex_2. A path of awareness represents which sub-sequences of an actual execution path can the participant keep track of. In a sense, paths of awareness are abstractions of execution paths on a model. The mapping between paths of awareness and execution paths is not 1-to-1. In fact, different execution paths can appear to a participant as the same path of awareness. For instance, consider the following execution path:

$$< t_1 \rightarrow t_2 \rightarrow t_4 \rightarrow t_8 \rightarrow t_7 \rightarrow t_6 \rightarrow t_8 \rightarrow t_9 \rightarrow t_{10} \rightarrow t_4 \rightarrow t_7 \rightarrow t_6 \rightarrow t_8 \rightarrow t_{11} >$$

It differs from ex_2 because it traverses t_8 twice. The path of awareness $ex_3^{p_1}$ is identical to $ex_2^{p_1}$, because p_1 is not aware of t_8.

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
p_1	✓		✓		✓	✓	✓		✓		✓	✓
p_2	✓	✓	✓	✓				✓			✓	✓
p_3		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	

Table 1. The transition-awareness of participants in the model presented in Figure 20.

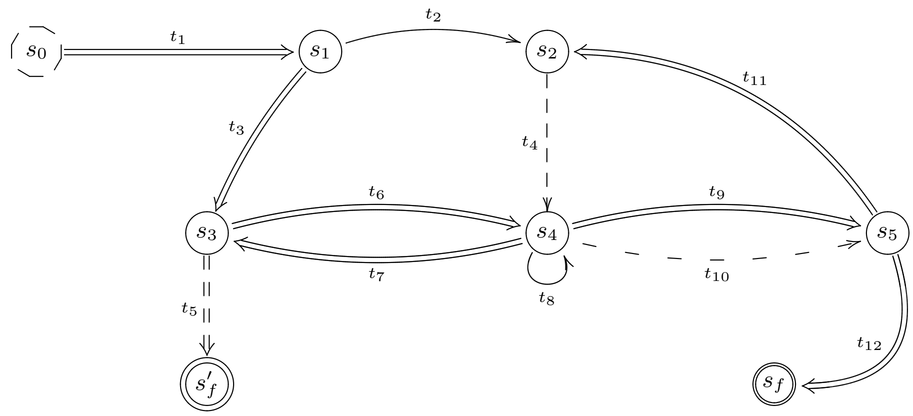


Fig. 21. The same model in Figure 20 high-lighting with a double line the transitions that p_1 is aware of.

A *shred* is a sequence of maximum size of defined transition identifiers in a path of awareness, such as "$t_7 \rightarrow t_6 \rightarrow t_8 \rightarrow t_{11}$" in $ex_2^{p_1}$. For instance, t_7 is not a shred in $ex_2^{p_1}$, because "$? \rightarrow t_7 \rightarrow ?$" never appears $ex_2^{p_1}$.

Path of awareness and shreds are instrumental to define efficient algorithms that calculate the state- and transition awareness of the participants on a model, which can be engineered as a graph-colouring problem. However, this is left as future work.

5 Soundness in Choreography Business Protocols

This section characterizes two classes class of choreography business protocols models: *participant-sound*, that can be executed by their participants in a completely distributed way, and *time-sound*, that are guaranteed not to hang because of the discretionality of participants about actually generating messages. Choreographic models that are both participant- and time- sound are said to be *sound*. All the runs of sound models complete in finite time without occurring in exception (e.g., because of generation of messages violating time conditions).

Time- and participant-soundness, likewise the state- and transition-awareness on which are based, are defined only on well-formed choreography models. Therefore in the remainder of the section, unless specified differently, every model mentioned is meant to be a well-formed choreography one.

The current section is organized as follows: Section 5.1 defines time-soundness, while Section 5.2 defines participant soundness. Finally, Section 5.3 introduces the definitions of full-soundness and proves the main Theorem 51 associated with it.

5.1 Time Soundness

Message-based transitions and their associate time constraints define when a participant *may* generate a message, namely the time-windows associated with the message-exchange. However, the generation of messages is not compulsory. This principle is called participant's *discretionality*, and it has implications on the execution of models.

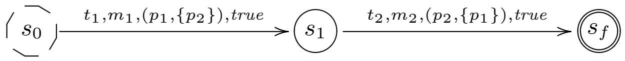

s_0
t_1,m_1,(p_1,{p_2}),*true*
s_1
t_2,m_2,(p_2,{p_1}),*true*
s_f

Fig. 22. A business protocol which may stall in state s_1 in case the participant p_2 decides not to generate the message m_2.

For instance, consider the business protocol model presented in Figure 22. An instance currently in state s_1 would remain in that state forever in case the participant p_2 does not want to generate the message m_2. Since the time constraint associated with the transition t_2 is "*true*", the participant p_2 has no means to understand that the message m_2 will never be delivered. The state s_1 is not final, and it means that the execution will never complete.

The discretionality of participants does not affect final states, that are absorbing, and because entering a final state automatically completes the execution of an instance.

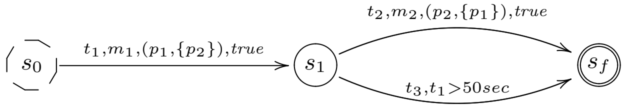

t_2,m_2,(p_2,{p_1}),*true*
s_0
t_1,m_1,(p_1,{p_2}),*true*
s_1
s_f
t_3,t_1>50*sec*

Fig. 23. A time-sound version of the business protocol presented in Figure 22.

The possibility of business protocol instance to get stuck forever in a non-final state is a serious concern in a distributed multi-participant environment. For a participant is far more desirable to conclude a conversation reaching a final state representing a fault, than having to wait forever for a message that may never come. Thus the need to characterize the class of models that are not affected by such issues.

The choreography-based models that can not get stuck in non-final state because of participants' discretionality are called *time-sound*. For instance, consider the business protocol presented in Figure 23, which is obtained by adding the automatic transition t_3 to the model presented in Figure 22. If the message m_2 is not generated within 50 *secs* since entering the state s_1, the automatic transition t_3 will be traversed. Because of t_3, there is the guarantee that instances of that model will not get stuck forever in s_1.

The model in Figure 23 delivers the intuition behind time-sound models: a model is time-sound if, in each state that may be affected by participants' discretionality (i.e., non-final and with outgoing message-based transitions; initial states are a special case, as they are concerned only if they have incoming transitions), there is an outgoing automatic transition that leads to a different state and its time condition is *fair*[3]. A time condition is fair if, at any point in time since the beginning of the execution of an instance, there is a future point in time in which the condition is verified. The time condition "$t_1 > 50$ *sec*" is fair because it always evaluates to "**true**" after 50 *sec* since the traversing of t_1. Similarly, "*true*" is fair because, being a tautology, it always evaluates to "**true**".

In general, proving the fairness of a timed propositional formula is a hard problem. In the case of timed constraints however, it is possible to prove the fairness of a timed constraint by examining its structure. The function $\mathbf{fair}(c \in C) \rightarrow \{\mathbf{true}, \mathbf{false}\}$ is defined in Definition 51 on the BNF structure of the time conditions presented in Definition 22.

Definition 51 *The function* $\mathbf{fair}(c \in C) \rightarrow \{\mathbf{true}, \mathbf{false}\}$ *is defined as:*

$$\mathbf{fair}(c) := \begin{cases} \mathbf{true} & \textit{if } c = \textit{true} \\ \mathbf{true} & \textit{if } c = t_{id}\ \mathbf{op}\ \mathbf{TIME} \wedge \\ & \quad \mathbf{op} \in \{\neq, >, \geq\} \\ \mathbf{false} & \textit{if } c = t_{id}\ \mathbf{op}\ \mathbf{TIME} \wedge \\ & \quad \mathbf{op} \in \{<, \leq\} \\ \neg\mathbf{fair}(c_1) & \textit{if } c = \neg c_1 \\ \mathbf{fair}(c_1) \vee \mathbf{fair}(c_2) & \textit{if } c = c_1 \vee c_2 \\ \mathbf{fair}(c_1) \wedge \mathbf{fair}(c_2) & \textit{if } c = c_1 \wedge c_2 \end{cases}$$

The function $\mathbf{fair}(c)$ returns **true** if c is fair, and **false** otherwise. The bases cases are defined on the atomic statements "*true*" and "t_{id} **OP TIME**" of Definition 22. The base case "t_{id} **OP TIME**" is split in two according to the particular **OP** used.

Notice that the function $\mathbf{fair}(c)$ does not take into account the case in which an atomic statement "t_{id} **OP TIME**" evaluates to **false** because the transition identified by t_{id} has not been traversed yet. For instance, consider the model presented in Figure 24.

In order to face this particular case, the function $\mathbf{fair}(c)$ has to take into account the whole business protocol model instead of just a time condition in it. In particular, assuming that the time condition c is associated to the transition t'_{id}, it would need to modify the case for "$c = t_{id}\ \mathbf{op}\ \mathbf{TIME} \wedge \mathbf{op} \in \{\neq, >, \geq\}$" to require that t_{id} is guaranteed to execute at least once before the first traversing of t'_{id}, i.e. that t_{id} is a *dominator* of t'_{id}. The definition of *dominator* is presented in Definition 52.

[3] meaning that it has the property of *fairness* in the Model Checking meaning of the word.

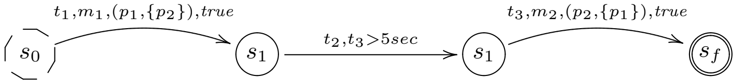


Fig. 24. The time condition "$t_3 > 5sec$" associated with t_2 it is not fair, because t_3 is never traversed before t_2.

Definition 52 Transition Dominators
The transition identified by t_{id} is a dominator *of the transition identified by t'_{id} in the model B, and we write t_{id} dom t'_{id}, if and only if:*

$$\forall ex_B^n \in EX_B \;.\; (\exists\, i \in [1,n] \;.\; \sigma(ex_B^n, i) = t'_{id}) \rightarrow (\exists\, j \in [1,i] \;.\; \sigma(ex_B^n, j) = t_{id})$$

That is, in every execution path defined over B, t_{id} always appears at least once before the first occurrence of t'_{id}.

The condition specified in Definition 52 can be efficiently checked by adapting to business protocol models one of the existing algorithms for dominance analysis on Control Flow Graphs[22], moving the focus from analyzing dominance among nodes to dominance among transitions.

Using the concept of dominator it is possible to re-define the function **fair** as shown in Definition 53 to take into account the case of atomic statements like "t_{id} **OP TIME**" be evaluated to **false** because the transition identified by t_{id} has not been traversed yet.

Definition 53 *Given the time condition c associated with the transition identified by t_{id}, the function $\mathbf{fair}(t_{id} \in T_{ID}, c \in C) \rightarrow \{\mathbf{true}, \mathbf{false}\}$ is defined as:*

$$\mathbf{fair}(t_{id}, c) := \begin{cases} \mathbf{true} & \textit{if } \; c = \textit{true} \\ t'_{id} \; \textit{dom} \; t_{id} & \textit{if } \; c = t'_{id} \; \mathbf{op\ TIME} \wedge \\ & \quad \wedge \; \mathbf{op} \in \{\neq, >, \geq\} \\ \mathbf{false} & \textit{if } \; c = t'_{id} \; \mathbf{op\ TIME} \wedge \\ & \quad \wedge \; \mathbf{op} \in \{<, \leq\} \\ \neg\mathbf{fair}(t_{id}, c_1) & \textit{if } \; c = \neg c_1 \\ \mathbf{fair}(t_{id}, c_1) \vee \mathbf{fair}(t_{id}, c_2) & \textit{if } \; c = c_1 \vee c_2 \\ \mathbf{fair}(t_{id}, c_1) \wedge \mathbf{fair}(t_{id}, c_2) & \textit{if } \; c = c_1 \wedge c_2 \end{cases}$$

Using Definition 53 it is finally possible to formally define the time-soundness of a choreography-based business protocol model.

Definition 54 Time Sound Choreography Models
A choreography-based business protocol model B is time-sound, and we write $sound_t$ B if and only if:

$$\mathbf{in}(s_0) \neq \emptyset \;\rightarrow\; \exists\, (t_{id}, s, s', c) \in A \;.s \neq s' \wedge \mathbf{fair}(t_{id}, c)) \tag{16}$$

$$\begin{aligned} &\forall s \in ((S \setminus \{s_0\}) \setminus S_F) \;.\; (\exists\, (t_{id}, s, s', m, l, c) \in E \wedge s \neq s') \rightarrow \\ &\qquad (\exists\, (t_{id}, s, s'', c) \in A \;.\; s \neq s'' \wedge \mathbf{fair}(t_{id}, c)) \end{aligned} \tag{17}$$

Condition 16 concerns the initial state s_0: if there is any transition inbound to it (i.e., the set returned by the function $\mathbf{in}(s_0)$ is not empty), that is there is a possibility that the initial set will be entered after that the instance has started, then there must be an automatic transition outgoing it whose time condition is infinitely often verified. Similarly, Condition 17 requires that every non-initial and non-final state that has at least one outgoing message-based transition ending up in a different state, is also the origin of an automatic transition towards a different state[4] whose time condition is infinitely often verified.

5.2 Participant Soundness

During the execution of a business protocol instance, the generation of a message outside the allowed time-window causes the instance to fail. In order to prevent this, participants must be able to correctly evaluate the time-windows during which messages can be generated. In turn, the ability to correctly evaluate the time-window during a messages can be generated requires the participant to:

- know that the instance is in a state in which the message can be generated;
- evaluate the time constraint associated with the message-based transition that carries that message.

Using the terminology introduced in Section 4, given the message-based transition "$(t_{id}, s, s', m, (p, P_{rec}), c)$", in order to never generate the message m causing the fail of an instance, p must be state-aware of s (the state in which t_{id} originates) and p can evaluate c. Formally:

$$p\ \mathbf{aware}_s\ s\ \wedge\ p\ \mathbf{can\_eval}\ c$$

Participant-sound business protocols are such that their participant never cause instances to fail because of the generation of messages outside the allowed time-windows. That is, every sender of a message-based transition is state aware of the source state of the transition, and the sender can evaluate the time condition associated with the transition.

There is also another issue that participant-sound models have to tackle. It is desirable that participants know when instances that they have taken part in are over. That is, all participants *involved* in a run must be aware of the final state that ends that run. A participant is involved in a run if it appears as sender or recipient of at least one of the message-based transitions that have been traversed during the run.

The formal definition of participant-soundness is provided in Definition 55.

Definition 55 Participant Sound Choreography Models
The choreography model B is participant-sound, and we write $sound_p\ B$ if and only if both the following conditions are satisfied:

$$\forall (t_{id}, s, s', m, (p_{send}, P_{rec}), c) \in E\ .\ p_{send}\ \mathbf{aware}_s\ s\ \wedge\ p_{send}\ \mathbf{can\_eval}\ c \qquad (18)$$

$$\forall ex^n_B\ .\ s_f := \theta(ex^n_B, n)\ \wedge\ \forall i \in [1, n]\ \sigma(ex^n_B, n) = (t_{id}, s, s', m, (p_s, P_{rec}), c) \rightarrow$$
$$\rightarrow \forall p \in (\{p_s\} \cup P_{rec})\ .\ p\ \mathbf{aware}_s\ s_f \qquad (19)$$

[4] N.B.: possibly the same state as the target of the message-based transition.

5.3 Full Soundness

Participants of participant-soundness choreography business protocol models that participants can not mistakenly cause instances of those models to fail because of the generation of messages outside the allowed time-windows. Time-sound choreography business protocol models do not get stuck forever in a non-initial and non-final state because of the discretionality of the participants with regard of the generation of the messages. A choreography business protocol model is *full-sound* if and only if it is both participant- and time-sound.

Definition 56 Full-Sound Choreography Models
A choreography model B is full-sound*, and we write $sound_f\ B$, if and only if:*

$$sound_f\ B \leftrightarrow (sound_t\ B \wedge sound_p\ B)$$

Theorem 51 is a direct consequence of Definition 56.

Theorem 51 Progression of Fully Sound Models
If participants do not willingly violate time-windows for message generation, every accepting run r_B^n of finite length on a multi-party service network business protocol B reaches a final state s_f in finite time. Moreover, at no step in its execution the protocol is broken because of the generation of messages outside their respective time-windows.

Proof. Proven by induction on the construction of a well-formed run. There are two basic cases, first and last step of the run, and an inductive one on the i-th step of the run, with $i \in (1, n)$.

- *first step*: since r_B^n is accepting (and thus well-formed), the first step traverses a message-based transition originating in the initial state. The time in the execution does not start until the first transition is traversed. Consequently, this case presents no problem. The time condition associated to a message-based transition originating in the initial state must be "*true*" (because B is well-formed), and thus the initiator participant can not possibly violate the time-window for the generation of the first message.
- *last step*: since r_B^n is accepting (and thus well-formed), the n-th (and final) step ends in a final state of B, and the execution is completed. There are no transitions outgoing final states, therefore no messages breaking the protocol can be generated.
- *inductive case*: the step $i - 1$ ends in a state s. State s is a non-final state, because r_B^n is accepting (and thus well-formed). Since s is not final and the model B is well-formed, s has at least one outgoing transition. The transition to be traversed at the i-th step can be either automatic or message-based. If t is automatic, then the time condition associated with t is infinitely often satisfied due to the time-soudness property of B. The traversal of an automatic transition does not generate any messages, therefore no time-window can be violated. If t is a message-based transition, the sender p of t generates the associated message m in a finite amount of time, and this happens before the

time condition of any automatic transition becomes verified. Otherwise, an automatic transition is instead traversed. Due to the participant-soundness of B, p can evaluate the time-window, and thus generate the message without breaking the protocol.

Theorem 51 proves a fundamental result regarding fully sound business protocols: runs in which participants adhere to the execution rules (i.e., do not generate messages outside the allowed time-windows) always complete in finite time. Moreover, because of the participant-soundness of the protocol, participants can execute the protocol in a completely distributed way. This result builds on the following assumptions:

Reliability of communication channel: sent messages are always delivered successfully;

Reliable time measurement: participants have consistent means of measuring time, i.e., private clocks evolving at the same speed;

Time-efficiency of communication channel: sent messages are delivered to all recipients instantaneously;

Reaction time of participants: participants take decisions and react without delays, i.e., participants do not perform any noticeable computation on states.

These assumptions are not unreasonable for current enterprise systems, which offer run-time environments where fully sound business protocols can be executed efficiently. Enterprise service buses offer reliable communication channels (i.e., by implementing the WS-ReliableMessaging specification) and can rely on protocols like *Internet Network Time Protocol* (NTP) to ensure that participants measure the elapsing of time in a consistent way. The requirements on the time-efficiency of the communication channel can be overcome by employing strategies from communication networks such as *Time Division Multiplexing* (TDM), or adopting a granularity in time measurements (e.g., milli-seconds, seconds, etc) suitable to mask the delays in the delivery of messages.

6 Summary

The Service Oriented Architecture approach to the design and implementation of information systems revolves around connecting and composing services that provide functionalities to other services and service consumers.

Services communicate via message exchanges organized in complex patterns called conversations. Each service in the conversation is a participant. A conversation can be described from a global perspective with a choreography, or from the local perspective of one of its participants using an orchestration.

Choreographies and orchestrations that describe the same conversation are connected. Orchestrations can be seen as projections on choreographies of the point of view of one single participant. Conversely, choreographies can be seen as compositions of the orchestrations of the participants.

Modeling languages that describe orchestrations and choreographies can be grouped in business protocol- and business process languages according to how they model the structure of the conversation. On the one hand, business protocol languages describe how participants produce and consume messages without providing any detail concerning their internal logic. On the other hand business process languages describe the order of the message exchanges by modeling an abstraction of the internal logic of the participants (for instance with a workflow containing activities like "Deliver Message" and "Receive Message").

The ability to describe orchestrations and choreographies are critical for SOA. In the literature there are frameworks for business protocols proposed that describe both in a coherent way, but they are based on notations hard to use for web service designers (namely, colored Petri Nets). In this work we aimed at filling this gap by proposing a business protocol framework based on deterministic finite automata that can describe both orchestrations and choreographies. We have presented the meta-model (i.e., the syntax), the semantics of the execution of business protocol models. A mapping from business protocol models to timed automata provides model checking on business protocol models, as well as a foundation to the study of the expressiveness of business protocols in terms of formal languages, Timed Regular Expressions and Data Languages.

Through the concept of state- and transition-awareness we have studied the knowledge that participants accumulate about the business protocol instances they participant to. Further building on the definition of awareness, we have characterized the classes of choreography business models that are time-, participant- and full-sound. The instances of time-sound models do not get stuck during their execution because of the discretionality of participants with regard of the generation of messages (message generation is not mandatory). Participant-sound models are such that participants have enough information about the execution so that they never cause the failure of instances because of violations of the business protocol due to the generation of messages at the wrong moment. Full-sound models are both time-sound and participant-sound, and their characteristics make them suitable for execution in a completely distributed environment where the only means of communication among the participants are the message exchanges within the conversation.

References

1. Papazoglou, M.P.: Service-oriented computing: Concepts, characteristics and directions. In: WISE, IEEE Computer Society (2003) 3–12
2. Papazoglou, M.P.: Web Services: Principles and Technology. Pearson Education Limited (2008)
3. Web Services Business Process Execution Language (WSBPEL) TC, O.: Web Services Business Process Execution Language Version 2.0. OASIS Standard, OASIS (April 2007)
4. Nitzsche, J., van Lessen, T., Karastoyanova, D., Leymann, F.: BPEL$^{\text{light}}$. In Alonso, G., Dadam, P., Rosemann, M., eds.: BPM. Volume 4714 of Lecture Notes in Computer Science., Springer (2007) 214–229

5. Decker, G., Kopp, O., Leymann, F., Weske, M.: BPEL4Chor: Extending BPEL for modeling choreographies. In: ICWS, IEEE Computer Society (2007) 296–303
6. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet Another Workflow Language. Information Systems **30**(4) (2005) 245–275
7. Zaha, J.M., Barros, A.P., Dumas, M., ter Hofstede, A.H.M.: Let's Dance: A language for service behavior modeling. In Meersman, R., Tari, Z., eds.: OTM Conferences (1). Volume 4275 of Lecture Notes in Computer Science., Springer (2006) 145–162
8. Lapadula, A., Pugliese, R., Tiezzi, F.: A calculus for orchestration of web services. In Nicola, R.D., ed.: ESOP. Volume 4421 of Lecture Notes in Computer Science., Springer (2007) 33–47
9. Benatallah, B., Casati, F., Ponge, J., Toumani, F.: On temporal abstractions of web service protocols. In Belo, O., Eder, J., e Cunha, J.F., Pastor, O., eds.: CAiSE Short Paper Proceedings. Volume 161 of CEUR Workshop Proceedings., CEUR-WS.org (2005)
10. Ponge, J., Benatallah, B., Casati, F., Toumani, F.: Fine-grained compatibility and replaceability analysis of timed web service protocols. In Parent, C., Schewe, K.D., Storey, V.C., Thalheim, B., eds.: ER. Volume 4801 of Lecture Notes in Computer Science., Springer (2007) 599–614
11. Wombacher, A., Mahleko, B.: Finding trading partners to establish ad-hoc business processes. In Meersman, R., Tari, Z., eds.: CoopIS/DOA/ODBASE. Volume 2519 of Lecture Notes in Computer Science., Springer (2002) 339–355
12. Benatallah, B., Casati, F., Toumani, F.: Analysis and management of web service protocols. In Atzeni, P., Chu, W.W., Lu, H., Zhou, S., Ling, T.W., eds.: ER. Volume 3288 of Lecture Notes in Computer Science., Springer (2004) 524–541
13. Lohmann, N., Massuthe, P., Wolf, K.: Operating guidelines for finite-state services. In Kleijn, J., Yakovlev, A., eds.: ICATPN. Volume 4546 of Lecture Notes in Computer Science., Springer (2007) 321–341
14. Deng, X., Lin, Z., Chen, W., Xiao, R., Fang, L., Li, L.: Modeling web service choreography and orchestration with colored petri nets. In Feng, W., Gao, F., eds.: SNPD (2), IEEE Computer Society (2007) 838–843
15. Yellin, D.M., Strom, R.E.: Protocol specifications and component adaptors. ACM Transactions on Programming Languages and Systems **19**(2) (1997) 292–333
16. Bozga, M., Maler, O., Tripakis, S.: Efficient verification of timed automata using dense and discrete time semantics. In Pierre, L., Kropf, T., eds.: CHARME. Volume 1703 of Lecture Notes in Computer Science., Springer (1999) 125–141
17. Wozna, B., Zbrzezny, A.: Checking actl* properties of discrete timed automata via bounded model checking. In Larsen, K.G., Niebert, P., eds.: FORMATS. Volume 2791 of Lecture Notes in Computer Science., Springer (2003) 18–33
18. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation (2nd Edition). Addison Wesley (November 2000)
19. Asarin, E., Caspi, P., Maler, O.: Timed regular expressions. Journal of the ACM **49**(2) (2002) 172–206
20. Bouyer, P., Petit, A.: Decomposition and composition of timed automata. In Wiedermann, J., van Emde Boas, P., Nielsen, M., eds.: ICALP. Volume 1644 of Lecture Notes in Computer Science., Springer (1999) 210–219
21. Bouyer, P.: A logical characterization of data languages. Information Processing Letters **84**(2) (2002) 75–85
22. Sutter, B.D., Put, L.V., Bosschere, K.D.: A practical interprocedural dominance algorithm. ACM Transactions on Programming Languages and Systems **29**(4) (2007)

A Auxiliary Definitions

Definition A1 *Given the business protocol B as in Definition 21, the function* $\mathbf{in}(s \in S) \rightarrow (E \cup A)$ *is defined as:*

$$\mathbf{in}(s) := \{(t_{id}, s_i, s, m, l, c) \in E\} \cup \{(t_{id}, s_j, s, c) \in A\}$$

The function $\mathbf{in}(s)$ returns the transitions in B that have s as target state.

Definition A2 *Given the business protocol B as in Definition 21, the function* $\mathbf{out}(s \in S) \rightarrow (E \cup A)$ *is defined as:*

$$\mathbf{out}(s) := \{(t_{id}, s, s_i, m, l, c) \in E\} \cup \{(t_{id}, s, s_j, c) \in A\}$$

The function $\mathbf{out}(s)$ returns the transitions in B that have s as source state.

Definition A3 *Given the business protocol B as in Definition 21, the function* $\mathbf{source}(t \in T_{id}) \rightarrow (S \cup \{\varepsilon\})$ *is defined as:*

$$\mathbf{source}(t_{id}) := \begin{cases} s_i & \textit{if} \, \exists \, (t_{id}, s_i, s_j, m, l, c) \in E \\ s_i & \textit{if} \, \exists \, (t_{id}, s_i, s_j, c) \in A \\ \varepsilon & \textit{otherwise} \end{cases}$$

The function $\mathbf{source}(t_{id})$ returns the source state of the transition identified by t_{id}, or ε in case there is no such transition.

Definition A4 *Given the business protocol B as in Definition 21, the function* $\mathbf{target}(t \in T_{id}) \rightarrow (S \cup \{\varepsilon\})$ *is defined as:*

$$\mathbf{target}(t_{id}) := \begin{cases} s_j & \textit{if} \, \exists \, (t_{id}, s_i, s_j, m, l, c) \in E \\ s_j & \textit{if} \, \exists \, (t_{id}, s_i, s_j, c) \in A \\ \varepsilon & \textit{otherwise} \end{cases}$$

The function $\mathbf{target}(t_{id})$ returns the target state of the transition identified by t_{id}, or ε in case there is no such transition.

Definition A5 *Given the execution path ex_B^n of length n defined on the model B as per Definition 21, the function $\sigma(ex_B^n, i \in \mathbb{N}^+) \rightarrow T_{id}$ is defined as:*

$$\sigma(ex_B^n, i) := \begin{cases} t_{id}^i & \textit{if} \, i \, \in \, [i, n] \\ \perp & \textit{otherwise} \end{cases}$$

where t_{id}^i is the i-th transition identifier mentioned in the execution path ex_B^n.

The function $\sigma(ex_B^n, i)$ returns the i-th transition identifier mentioned in the execution path ex_B^n, or $\perp$ otherwise.

Definition A6 *Given the run r_B^n of length n defined on the model B as per Definition 21, the function $\sigma(r_B^n, i \in \mathbb{N}^+) \rightarrow T_{id} \times \mathbb{T}$ is defined as:*

$$\sigma(ex_B^n, i) := \begin{cases} t_{id}^i & \textit{if } i \ \in \ [i, n] \\ \bot & \textit{otherwise} \end{cases}$$

where t_{id}^i is the i-th transition identifier mentioned in the execution path ex_B^n.

The function $\sigma(ex_B^n, i)$ returns the i-th transition identifier mentioned in the execution path ex_B^n, or $\bot$ otherwise.

Definition A7 *Given the run r_B^n of length n defined on the model B as per Definition 21, the function $\theta(r_B^n, i \in \mathbb{N}^+) \rightarrow T_{id}$ is defined as:*

$$\theta(ex_B^n, i) := \begin{cases} \mathbf{source}(\sigma(ex_B^n, i)) & \textit{if } i \ \in \ [i, n] \\ \mathbf{target}(\sigma(ex_B^n, n)) & \textit{if } i = n + 1 \\ \bot & \textit{otherwise} \end{cases}$$

The function $\theta(ex_B^n, i)$ returns the i-th transition identifier mentioned in the execution path ex_B^n, or $\bot$ otherwise.