Sound Multi-party Business Protocols for Service Networks*

Michele Mancioppi[1], Manuel Carro[2],
Willem-Jan van den Heuvel[1], and Mike P. Papazoglou[1]

[1] INFOLAB, Dept. of Information Systems and Management,
Tilburg University, The Netherlands
{m.mancioppi,wjheuvel,mikep}@uvt.nl
[2] Universidad Politécnica de Madrid
mcarro@fi.upm.es

Abstract. Service networks comprise large numbers of long-running, highly dynamic complex end-to-end service interactions reflecting asynchronous message flows that typically transcend several organizations and span several geographical locations. At the communication level, service network business protocols can be flexible ranging from conventional inter-organizational point-to-point service interactions to fully blown dynamic multi-party interactions of global reach within which each participant may contribute its activities and services. In this paper we introduce a formal framework enriched with temporal constraints to describe multi-party business protocols for service networks. We extend this framework with 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 while guaranteeing eventual termination.

1 Introduction

Today's application-oriented services cannot scale to meet the number and nature of demands already placed on them, let alone a new generation of more complex applications involving several organizations. Most of today's applications are based on the assumption of the ubiquitous availability of point-to-point integration between any two interacting parties from the perspective of a single organization. One of the main reasons is the use of orchestration languages (e.g., BPEL) to describe how services can interact with each other at the message level from the perspective and under control of a single service. Moreover, the interactions are limited to uni-cast scenarios. This is extremely restrictive for applications characterized by wide-scale and complex dynamic interactions.

* 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. Manuel Carro was also partially supported by Spanish MEC project TIN2005-09207-C03 *MERIT-COMVERS* and project S-0505/TIC/0407 *PROMESAS*.

A. Bouguettaya, I. Krueger, and T. Margaria (Eds.): ICSOC 2008, LNCS 5364, pp. 302–316, 2008.

1.1 Service Networks

The full potential of services technology as a means of developing mission-critical applications used by a wider spectrum of people and organizations will only be realized when business processes (which are services themselves) are able to express business collaborations and transactions that occur between multiple business process endpoints, rather than a specific business process that is executed from the perspective of a single party. Such collaborative, complex service interactions typically require specifying sequences of peer-to-peer message exchanges between a collection of end-to-end services within stateful, long-running interactions involving several parties. This gives raise to the concept of *service networks*.

Service networks comprise large numbers of long-running, highly dynamic complex end-to-end service interactions reflecting asynchronous message flows that typically span several organizations and geographical locations. The term "complex end-to-end service interaction" encompasses a succession of automated business processes, which are involved in joint inter-company business conversations and transactions across a federation of cooperating organizations. This widens considerably the scope of service-based applications by providing the possibility of developing a whole new range of innovative service-based applications.

Service networks properly sequence service activities according to the flow definitions in a process collaboration model into end-to-end service constellations, assign work items to the appropriate human actors or groups, and ensure that both human- and system-based activities are performed within specified time frames. This entails multiple technical requirements, which include binding to heterogeneous systems, synchronous and asynchronous message exchange patterns, data manipulation, flow coordination, exception management, business events, long running business transactions, and so on.

1.2 Multi-party Business Conversations in Service Networks

At the communication level, service networks essentially comprise asynchronous message flows between multiple service consumers and providers. Business conversations can be flexible, ranging from conventional inter-organizational point-to-point service interactions (as is the norm with current orchestration technologies) to *fully blown dynamic multi-party interactions* of global reach within which each participant may contribute its activities and services.

At the communication-level, service networks exchange sequences of messages grouped into operations, which have to be structured into complex conversations. The business logic which controls these operations is embedded into the implementation of the interacting parties. This is due to the limitations of standards used in practice, e.g., WSDL 1.1. The new generation of Web services standards that will be based on WSDL 2.0 will focus on custom *Message Exchange Patterns* (MEPs). MEPs are currently elementary building blocks (i.e., one-way messaging, request-reply, solicit-response and notification) from which business protocols can be constructed, describing the multi-party message interactions required by a business process. Ideally, MEPs should also include timing

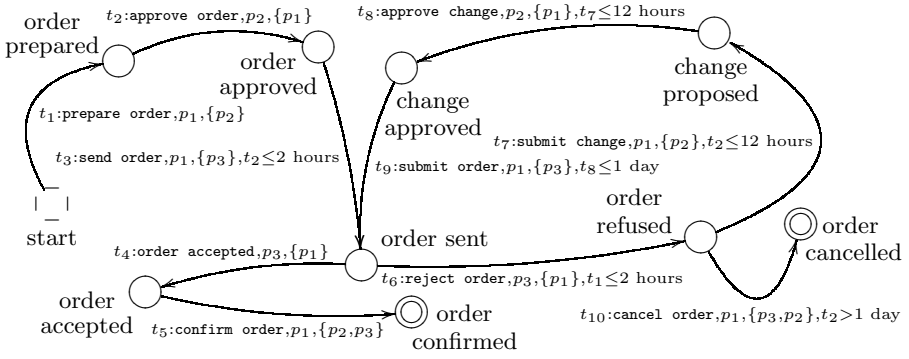

Fig. 1. The *Purchase Order* business protocol

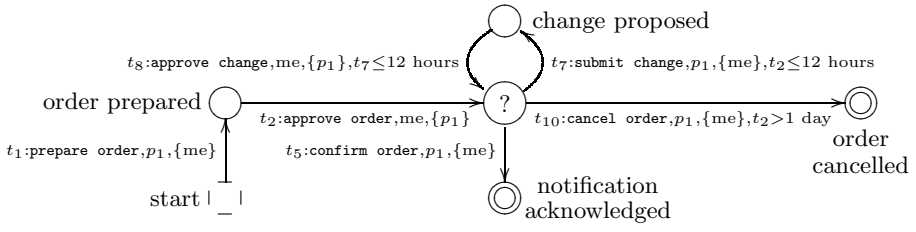

Fig. 2. The *Purchase Order* protocol from the perspective of the sales approver

constraints and have the expressive power necessary to describe complex series of interactions allowing for multiple alternative message exchanges to be performed at any given point in the execution of the MEP.

Figure 1 describes a simplified version of an end-to-end order fulfillment process represented by means of a complex set of interacting Web services. From a service network perspective the *Purchase Order* business protocol essentially corresponds to a *choreography* scenario and involves message exchanges between three parties: a buyer (p_1), a sales approver (p_2), and a seller (p_3), described from a global perspective. A buyer's order conveys information about the order. The sales approver performs credit check and stock authorization, while the final order fulfillment and billing lies with the seller.

The previous scenario can be contrasted with the one illustrated in Figure 2 which describes the *Purchase Order* protocol from the sales approver's perspective, which effectively corresponds to a service *orchestration* scenario, where all information not concerning the sales approver, both about states of the protocol and transitions, has been removed. The only message exchanges relevant for the sales approver are the ones in which it appears as either sender or recipient of messages. Note that in the service network business protocol where the sales approver is listed as recipient of a message together with other participants, information about other recipients, like transitions t_5 and t_{10} in Figure 2, is eliminated as it cannot be observed by the sales approver. Eliminating information regarding transitions that do not involve the sales approver may render

some states of the business protocol irrelevant. Collapsed states are labelled as "?" in Figure 2 and are referred to as *incognito* states. Incognito states are "super-states" in orchestrations which resume states in the original choreography that are not discernible by the participant because of lack of information. The algorithm to extract the point of view of a participant from a choreography (and the creation of the incognito states) [1] is outside the scope of this paper.

Business protocols such as the one depicted in Figure 1 can lead to erroneous results if not managed properly. Therefore, an important consideration is how to factor a multi-party business protocol to achieve end-to-end conversation sequences that are robust and safe. Of particular importance is the use of formal techniques to describe multi-party MEPs for service networks and verify correct execution and temporal properties of these protocols.

In this paper we define a formal model for multi-party business protocols based on *Deterministic Finite Automata* (DFA). Our model has been inspired by [2], which we have extended to describe multi-participant orchestration business protocols and choreographies. We also introduce 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 and timely manner without relying on any external synchronization mechanisms.

2 Formal Exposition of Business Protocols

This section introduces the basics of our formalization of business protocols. We use a graph-based representation which permits us to give an intuitive and simple semantics to the execution of runs, and which makes it easy to perform a mapping to timed automata, which enables model checking-based verification of temporal logic properties. Space constraints force us to be concise; the interested reader can find more details and examples in [1].

2.1 Running Example: Purchase Order Business Protocol

Our representation of business protocols is based on DFA, which are considered appropriate for describing message exchanges for e-commerce applications [3], enriched with time conditions on the transitions. The states of the protocol are mapped to states in the DFA, and the protocol evolves by traversing transitions. Transitions are uniquely identified by their *transition identifiers*, and have associated *time conditions* that restrict when they can be traversed. There are two types of transitions: *message-based* and *automatic*. Message-based transitions are associated to a message exchange between a *sender* and a number of *recipients*. Automatic transitions are triggered by their associated time conditions becoming true when time advances.

In Figure 1, the participants communicate with each other by exchanging messages. The buyer *initiates* the conversation (and thus it is the *initiator*) by sending the message `prepare order` to the sales approver (transition t_1). The notation

$$t_1 : \texttt{prepare order}, p_1, \{p_2\}$$

means that the message-based transition t_1 represents the delivery of an instance of the message type `prepare order` by the participant p_1 to the participant p_2. The sales approver authorizes the order and replies back to the buyer with an `approve order` message (transition t_2). Following this, the buyer has to dispatch the order within two hours to the seller (transition t_3) with the `send order` message. If the seller accepts the order, it sends to the participants a message `order accepted` (transition t_4). The seller can reject an order by sending the message `reject order` (transition t_6) within two hours since the reception of the message `send order`. If the order is accepted, the conversation ends by traversing transition t_5, where the buyer sends the message `confirm order` to both seller and buyer. The buyer can, however, cancel an order that has been rejected within one day (t_{10}) or propose changes the order to the sales approver a within 12 hours. In this case, the sales approver must approve the change within 12 more hours (t_7 and t_8). The buyer then sends another message to the seller (t_9) with the updated order for the seller to either approve or reject it.

Each message-based transition in the business protocol is univocally associated with a message type. Different message types in the business protocol are disjoint: that is, given a message that is an instance of a message type, it is possible to map it back to only that message type.[1] Thus, recipients are able to tell which transition has taken place simply by observing the message they received. Each business protocol has a unique initial state, which has no incoming transitions. All transitions outgoing the initial state are message-based, and their associated time conditions are "*true*". Multiple final states are allowed, and final states are required to be absorbing (i.e. they have no outgoing transitions).

Time conditions determine when transitions can be traversed. Message-based transitions can be traversed only if their associated time expressions evaluate to "**true**" and the corresponding exchange of messages between the partners actually is actually performed. Automatic transitions are immediately traversed as soon as their associated time expressions evaluate to "**true**". Time expressions are obtained by composing *atomic predicates*, such as "*true*" or "$t_3 > 2$ hours" which refer to the time at which a transition happened using directly its transition identifier (e.g., t_3). In a sense this is similar to what happens in timed automata [4] (and we assume similar time expressions are used), where *clocks* store information on elapsed time, and they can be reset by traversing transitions. In our proposal, every transition has one associated timer (named as its transition identifier) which is reset every time the transition is traversed. We will use specific time expressions to denote either absolute points in time (e.g., 12:35AM) or durations (e.g., 2 hours), and we will let the context disambiguate if needed.

The time at which a time expression is evaluated is used as the reference for the evaluation of atomic predicates. Atomic predicates referencing a not-

[1] Requiring all message-types to be completely disjoint with each other is not a limitation in the current WS landscape: message instances can be marked with custom SOAP headers (thereby without affecting the actual contents of the message), by embedding message identifiers, or by defining the message types as XSD type or element definitions accompanied by *XPath* expressions acting as assertions.

yet-taken transition evaluate to "**false**", while "**true**" time expressions can be omitted. Time expressions combining atomic predicates are evaluated with the usual rules for conjunction, disjunction, and negation.

2.2 Taxonomy of Business Protocols

In this section we introduce a taxonomy of business protocols according to the following two dimensions:

- *Number of participants* involved in the conversations:
  two-party: two participants, or
  multi-party: more than two (but finitely more) participants.
- The *perspective* adopted to describe the structure of the conversation:
  orchestration: the conversation is described as the point of view of a particular participant (as in Figure 2), or
  choreography: the conversation describes all the possible interactions that multiple participants in a service network can have (Figure 1).

The classification is based on the combinations of the two parameters, as presented in Figure 3. We will use this classification later on, as different classes of business protocols have different properties. For instance, soundness in service networks (Section 3) is defined in terms of two-party choreography and multi-party choreography business protocols.

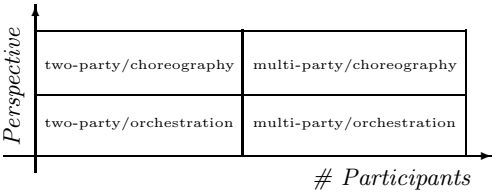

Fig. 3. Taxonomy of Business Protocols

2.3 Execution of Business Protocols

The execution of a business protocol essentially implies traversing the states following the usual automata conventions, in addition to the considerations stated in Section 2.1 regarding when a transition can be traversed according to its time condition. A sequence of consecutive transitions that goes from the initial state to a final state is an *execution path*. Examples of execution paths on the business protocols presented in Figure 1 are the following:

$$ex_1 := t_1 \rightarrow t_2 \rightarrow t_3 \rightarrow t_4 \rightarrow t_5$$

$$ex_2 := t_1 \rightarrow t_2 \rightarrow t_3 \rightarrow t_6 \rightarrow t_7 \rightarrow t_8 \rightarrow t_9 \rightarrow t_4 \rightarrow t_5$$

There are usually multiple execution paths in the same business protocol (actually, there may be infinite if there are loops).

A particular execution of a business protocol is called a *run*, and it consists of a sequence of *steps* (t, τ), corresponding to the traversal of transition t at time τ. A run r_B^n of length n on the business protocol B is represented as:

$$r_B^n := (t_1, \tau_1) \rightarrow \ldots \rightarrow (t_i, \tau_i) \rightarrow \ldots \rightarrow (t_n, \tau_n)$$

where $(t_i, \tau_i) \rightarrow (t_{i+1}, \tau_{i+1})$ represents that t_i was traversed at time τ_i, followed by the step (t_{i+1}, τ_{i+1}). In a sense, runs are instances of execution paths. Different executions that follow the same execution paths may give rise to runs that differ on the times associated with the steps. The information available in the run (traversed transitions, their order, and their associated time), is used to evaluate time conditions in the remainder of the execution.

The set of all the runs that can take place on the business protocol B respecting the time constraints in it is denoted by R_B.[2] Runs in R_B are said to be *accepting* on B. Accepting runs have to follow the usual word accept rules for automata (they start in the initial state, end in a final state, and every transition starts in the state the previous one ended in) and the rules concerning the semantics of time conditions. For example, the following run is not accepting with respect to the business protocol illustrated in Figure 1 because it violates the time constraints associated with the transition t_3:

$$(t_1, 0h) \rightarrow (t_2, \tau_2) \rightarrow (t_3, \text{2h:30m}) \rightarrow (t_4, \tau_4) \rightarrow (t_5, \tau_5)$$

More precisely, if we denote by c_t the time conditions associated with transition t are:

- For any step (t, τ), c_t must evaluate to **true** at time τ.
- For any two steps $(t_a, \tau_a) \rightarrow (t_b, \tau_b)$ where t_b starts in state s:
  - $\tau_a < \tau_b$ must hold (i.e., time increases monotonically).
  - If t_b is message-based, no condition of any automatic transition departing from s may have evaluated to **true** in the time span from τ_a to τ_b.
  - If t_b is an automatic transition, τ_b is the least time greater than τ_a in which c_{t_b} evaluated to **true**, and no time condition of any transition starting at s may have evaluated to **true** in the time span from τ_a to τ_b.

2.4 Mapping Business Protocols to Timed Automata

Expressing and checking temporal properties, like "every possible accepting run of the business protocol completes in at most 20 seconds", is important to, for example, ensuring that time-related QoS properties hold. In order to pave the way towards model-checking of business protocols, we provide a mapping from business protocols to timed automata. Timed automata are labelled transition system which use real-valued variables (*timers*) to model clocks. Timed automata accept *timed words*, where some input symbols can be accepted (i.e., the corresponding transition is taken) only at certain points in time specified by expressions on the timers.

The mapping, adapted from the one in [2] for two-party orchestration business protocols, converts states of the business protocol B into states of the *Equivalent Timed Automaton* (ETA) T_B preserving the markings for initial and final states. A timer is created for each transition using the identifier of that transition, which is reset to zero every time the transition is taken. Transitions in B

[2] Borrowing terminology from automata theory, R_B is the *language* of B.

are converted into transitions in T_B, labelled with the original identifier in the business protocol. The time conditions associated with message-based transitions are copied unmodified, but the conditions associated with automatic transitions require some additional tweaking. It is necessary to enforce in the ETA that, in case that several automatic transitions starting in the same state of a business protocol can evaluate to true simultaneously, only one (e.g., the one with the least transition identifier) is traversed.

Therefore, for an automatic transition t starting at state s we just need to translate its associated time conditions into T_B as the (logical) conjunction of the time conditions for t and the conjunction of the **negation** of the time conditions associated to any other automatic transition originating at s whose transition identifier is smaller than t.[3]

The resulting ETA is deterministic. The language of T_B is, by construction, exactly R_B (see Section 2.3). Thus, checking whether a run can or can not take place on a certain business protocol boils down to checking if the run belongs to the language of its equivalent timed automaton.

The mapping to timed automata makes it possible to analyze, among other properties, the *inclusion* and *equivalence* of languages of business protocols, i.e., if a given business protocol can execute all or only some of the runs of another protocol. Inclusion and equivalence of languages is the corner-stone for any work on replaceability and compatibility of business protocols. Since the ETAs we generate are deterministic,[4] and they have by construction exactly the same language of the respective business protocols, the analysis of the inclusion of languages [4] on them solves the problem for the business protocols.

3 Sound Multi-party Business Protocols

For a given choreography business protocol (either two-party or multi-party) it is important to ensure that can be executed in a completely distributed and timely manner using the message exchanges in the protocol as the only communication means. If participants exchange messages at their own discretion (e.g., outside their time windows), the business protocol may be wrongly executed; they must therefore know when messages can be exchanged. Business protocols in which completely distributed execution is possible are called *participant-sound*.

An additional interesting property of choreography business protocols, called *time-soundness*, is the ability to avoid protocol stalls that could be caused by the *discretionary ability* of participants to decide whether to generate or not messages in a business protocol. Message-based transitions define *time windows* within which senders can generate messages with a *valid timestamp*. Time-soundness guarantees that the protocol eventually concludes disregarding messages which are not generated within the appropriate time frame.

[3] An example of this procedure can be found in [1].

[4] Deciding of language inclusion for non-deterministic timed automata is undecidable. However, the inclusion problem for deterministic timed automata is decidable [4].

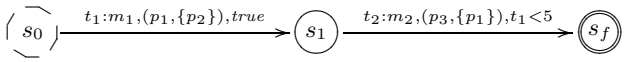

Fig. 4. A non participant-sound multi-party choreography business protocol

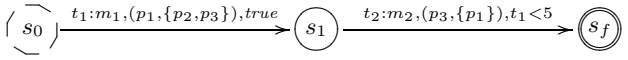

Fig. 5. A participant-sound version of the protocol in Figure 4

3.1 Participant-Soundness

A participant-sound business protocol can be executed correctly using as the only communication means among the participants the message exchanges in the protocol. Consider the business protocol presented in Figure 4: participant p_3, which is not involved in the message exchange upon traversing transition t_1, does not know that the protocol has entered state s_1 and that it is therefore expected to generate message m_2. Thus, the protocol is not participant-sound, because if p_3 relies only on the messages exchanged with the other participants, it will not have information enough to take part in the execution without risking to break the protocol by generating a message in the wrong moment. Consider now Figure 5, obtained by adding p_3 as recipient for the message-based transition t_1. Unlike the protocol in Figure 4, the one in Figure 5 is participant-sound: upon the receipt of message m_1, p_3 knows that the protocol is now in the state s_1, and that it can generate message m_2.

In order for a business protocol to be participant-sound, its participants must be able to evaluate time conditions associated to message-based transitions in which they act as either senders or recipients.

Recipients of message-based transitions can *observe* traversed transitions that affect them by retrieving the types of the messages they receive and using the one-to-one mapping between message types and message-based transitions (Section 2.1). Similarly, a time condition associated with an automatic transition (Section 2.1) defines the time windows in which that transition is immediately traversed if the protocol is currently in its source state. Automatic transitions do not require communication among participants: a participant *infers* that an automatic transition takes place when they can evaluate their time windows and knows that the execution is in the transition's source state.

If a participant knows when a transition can or cannot be traversed, it is said to be *transition-aware* of that transition:

Definition 1 (Transition-awareness (Working Definition)). *A participant* **p** *is transition-aware of a transition* **t** *in a business protocol* **B** *if every time* **t** *occurs during an execution of* **B** *one of the following holds:*

- **t** *is message-based, and* **p** *is involved in it (that is,* **p** *is either sender or recipient in* **t***), or*
- **t** *is automatic, and* **p** *can infer that* **t** *has occurred.*

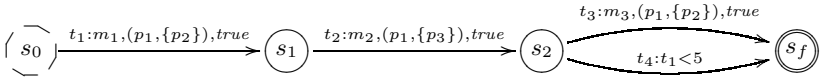

Fig. 6. State-awareness with automatic transitions

Transition-awareness affects the ability of participants to evaluate time conditions. A participant can evaluate a time condition if and only if it is transition-aware of all the transition identifiers appearing in that time condition. Due to transition-awareness, the participant always knows when the transition is traversed, and can keep track of the time of the most recent occurrence to evaluate time conditions defined on the basis of that transition.

Definition 1 is only a "working definition" because, while it delivers the intuition of transition-awareness, it does not explain when participants can infer the execution of automatic transitions. In order to formally explain this, there is some more ground work to do.

Similarly to transition-awareness, a participant is said to be *state-aware* of a state if it is aware of every time that the business protocol enters or leaves that state. This means that the participant is being informed of all transitions incoming and outgoing a specific state. Consider the multi-party choreography business protocol in Figure 6. The participant p_1 is state-aware of s_2 because it is aware of all transitions entering this state, all the message-based transitions leaving it (t_3), and it can evaluate when t_4 can be taken because it was also aware of transition t_1, needed to evaluate the condition $c_{t_4} \equiv t_1 < 5$. On the other hand, p_3 is not transition-aware of t_1. Therefore, it cannot evaluate correctly c_{t_4}, and thus it is not state-aware of s_2 as it cannot tell when s_2 is left through t_4.

Transition- and state-awareness, which are mutually dependent, are the keys to participant-soundness: if participants are aware of the message-based transitions in which they are involved, they have enough understanding of the protocol not to break it. The definitions of state- and transition-awareness follow:

Definition 2 (State-awareness). *A participant* **p** *in a business protocol* **B** *is state-aware of* **s** *if* **p** *is transition-aware of all transitions entering* **s** *and, for every transition* **t** *exiting* **s***:*

- *If* **t** *is message-based, then* **p** *is either the sender or a recipient of* **t**, *and* **p** *can evaluate the time condition associated with* **t**.
- *If* **t** *is automatic, then* **p** *can evaluate the time condition associated with* **t**.

Definition 3 (Transition-awareness). *The participant* **p** *in the business protocol* **B** *is transition-aware of a transition* **t** *with source state* **s** *if and only if one of the following conditions hold:*

- **t** *is message-based,* **p** *is sender in it, and* **p** *is state-aware of* **s**, *or*
- **t** *is message-based, and* **p** *is recipient in it, or*
- **t** *is automatic, and* **p** *is state-aware of* **s**.

Definitions 2 and 3 show how awareness *spreads* through the executions of a business protocol: a participant is state-aware of a state if it is transition-aware of the transitions entering that state.[5] A participant has to be state-aware of its source state in order to be transition-aware of automatic and message-based transitions in which it acts as the sender. A participant's awareness of states and transitions follows *paths of awareness*, made up of sequences of transitions and states that it is aware of, throughout the executions of a protocol. The paths always start with a message-based transition in which the participant is a recipient, and always end with message-based transitions entering final states, or states the participant is not aware of. The only exception to this rule are the paths of awareness for the participant that initiates a protocol. These start with a message-based transition originating in the initial state and where the participant is the sender and not a recipient.

Participants need also to know when the protocol has terminated. If a participant is not aware of the completion of a protocol run (i.e., if it does not know that the protocol has entered a final state), it might wait forever for messages that will never arrive. To prevent this from happening, participants are required to be aware of all final states.

It is possible to define participant-soundness on the basis of the definition of state-awareness:

Definition 4 (Participant-soundness). *A multi-party choreography business protocol is participant-sound if:*

1. *for every message-based transition* **t** *originating in a state* **s**, *the sender* **p** *of* **t** *is state-aware of* **s**;
2. *all participants are state-aware of the final states that belong to execution paths that contain transitions that the participants are aware of.*

That is, participant-soundness expresses the capability that senders have to generate their messages within the correct time windows and that all have participants to acknowledge the termination of the protocol execution if they were apart of it (note that it may be the case that some participants did not participate in the execution path followed by a certain instance).

3.2 Time-Soundness

As already explained, senders of message-based transitions can decide not to generate messages (and, thus, not to actually traverse transitions). This may suspend the execution of protocols. Consider the protocol in Figure 7: if p_2 does not wish to generate the message m_2 while the protocol is in the state s_1, the execution will never reach a final state, and therefore the protocol may *stall*.

Business protocols which never stall because of the participants' discretional ability to decide whether and when they would participate in message exchanges

[5] For reasons of simplicity, the formulation of Definition 2 and 3 are such that they may be affected by circularity in case the business protocol has loops in it. Definition of transition- and state-awareness not affected by circularity (but for all other practical purposes equivalent to the ones here proposed) are available in [1].

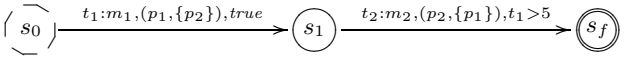

Fig. 7. A business protocol which may stall in state s_1 when the participant p_2 decides not to generate the message m_2

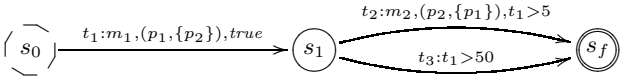

Fig. 8. A time-sound version of the business protocol presented in Figure 7

are called *time-sound*. The implication is that a business protocol is time-sound if, in every non-initial state of the protocol in which participants apply discretionality, there is always a possibility for the protocol to proceed to a new state. Message-based transitions leaving the initial state (like transition t_1 in Figure 7) are treated as special cases. Since the initial state has no incoming transitions, the traversal of a message-based transition originating in the initial state always represents the beginning of a protocol execution. Figure 8 shows a time-sound protocol, as per Definition 5, below:

Definition 5 (Time-soundness). *A business protocol is time-sound if for every non-initial state that has at least one outgoing message-based transition, there is at least one outgoing automatic transition whose associated time condition is satisfied infinitely often.*[6]

Because of the discrete time model adopted by business protocols, having infinitely often verifiable time conditions associated with automatic transitions guarantees that, no matter when a state is entered, at some point in the future one automatic transition will be traversable, and therefore the protocol execution will traverse it. While this does not prevent infinite loops from occurring, it is enough to prevent a protocol from stalling in a specific state. Note that these automatic transitions may very well lead the protocol to some *emergency state* to escape from unexpected situations (e.g., deadlines not met).

3.3 Full Soundness

Multi-party choreography business protocols that are both participant- and time-sound are *fully sound*. Fully sound business protocols exhibit a *progression property*: their execution is never indefinitely blocked in a non-initial and non-final state. This can alternatively be formulated as follows: finite runs of fully sound business protocols complete (i.e., reach a final state) in finite time. Theorem 1 proves the property of progression.

[6] That is, there exist infinitely many moments in time in which the time condition evaluates to "**true**".

Theorem 1 Progression of Fully Sound Business Protocols *Assuming that 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 an accepting 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, 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*" (Section 2.1), 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, the n-th (and final) step ends in a final state of B, and the execution is completed. Since there are no transitions outgoing final states, no messages breaking the protocol can be generated.
- *inductive case*: step $i - 1$ $(i - 1 < n)$ ends in a state s. Since the run is accepting, the state s is a neither initial nor final. Since s is not final, it has at least one outgoing transition. The transition t 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-soundness property of B. As traversing an automatic transition does not generate any messages, 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 is satisfied. Otherwise, an automatic transition is traversed instead. Due to the participant-soundness of B, p can evaluate the time-window, and thus it has all the information necessary to generate the message without breaking the protocol.

Theorem 1 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 successfully delivered;

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

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

Reaction time of participants: participants take decisions and react without delays, i.e., no noticeable computation is performed in states.

Current enterprise systems can actually offer a run-time environment in which fully sound business protocols can be efficiently executed. Enterprise service buses provide reliable communication channels (e.g., through implementations of the WS-ReliableMessaging specification). Time-efficiency requirements on the communication channel and participants' reaction time can be achieved by employing strategies from communication networks such as *Time Division Multiplexing*, while protocols like the *Internet Network Time Protocol* can be used to ensure that participants have a consistent view of time.

4 Related Work

Recently, a simplified version of BPEL 2.0, called BPEL^{Light} [5], has been proposed to encode complex, executable MEPs. BPEL^{Light} extends BPEL 2.0 with a WSDL-less interaction model that makes it possible to specify processes representing orchestration-based MEPs independently from Web service technologies. While this proposal has the advantage of being directly executable on suitable middleware, there is currently no direct support for model checking and validation.

Other DFA-based formalisms have been proposed to describe asynchronous message exchanges between two parties as orchestrations or choreographies. All these formalisms encode message exchanges as transitions connecting states of the protocol, and can ultimately be mapped to our formalism.

Two-party orchestration business protocols have been studied in [6,7]: for this particular class of business protocols, very relevant problems like compatibility and replaceability [2,8], evolution and migration strategies for running instances [9], have been already addressed.

An approach to matchmaking of two-party choreography business protocols (though not supporting time constraints) is presented in [10] as part of a search engine for ad-hoc business processes, as well as an extraction process for orchestration-based business protocols. These are called *views* and are created from choreographic protocols.

5 Conclusions and Future Work

Current orchestration languages (e.g., BPEL) describe how services can interact with each other at the message level from the perspective and under control of a single service. Moreover, the interactions are limited to uni-cast scenarios. This is extremely restrictive for applications characterized by wide-scale and complex dynamic interactions.

In this paper we introduced an intuitive formal model for describing multi-party service network business protocols based on Deterministic Finite Automata. In addition, we defined the notion of full soundness for multi-party

business protocols. Fully sound multi-party choreography business protocols rely solely on message exchanges as the only means of communication and can be executed consistently in a completely distributed manner, while guaranteeing termination. Our framework allows the description of business protocols and verification of their temporal properties using model checking of timed automata. Fully sound multi-party choreography business protocols contribute towards a comprehensive theory of management of business protocols for service networks.

Future work on choreography business protocols will focus on the evolution of business protocols and how it impacts time-related QoS parameters like turn around time, transaction rates, etc. Another area of work concerns protocol replaceability and compatibility analysis and versioning techniques for business protocols.

References

1. Mancioppi, M.: A Formal Framework for Multi-Party Business Protocols. CentER Discussion Paper 2008-79, Tilburg University (September 2008), http://center.uvt.nl/pub/dp2008.html
2. 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 2007. LNCS, vol. 4801, pp. 599–614. Springer, Heidelberg (2007)
3. Benyoucef, M., Keller, R.K.: An Evaluation of Formalisms for Negotiations in E-commerce. In: Kropf, P.G., Babin, G., Plaice, J., Unger, H. (eds.) DCW 2000. LNCS, vol. 1830, pp. 45–54. Springer, Heidelberg (2000)
4. Alur, R., Dill, D.L.: A Theory of Timed Automata. Theoretical Computer Science 126(2), 183–235 (1994)
5. Nitzsche, J., van Lessen, T., Karastoyanova, D., Leymann, F.: BPEL$^{\text{light}}$. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 214–229. Springer, Heidelberg (2007)
6. Yellin, D.M., Strom, R.E.: Protocol Specifications and Component Adaptors. ACM Transactions on Programming Languages and Systems 19(2), 292–333 (1997)
7. Benatallah, B., Casati, F., Toumani, F.: Representing, Analysing and Managing Web Service Protocols. Data Knowledge Engineering 58(3), 327–357 (2006)
8. 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 2004. LNCS, vol. 3288, pp. 524–541. Springer, Heidelberg (2004)
9. Ryu, S.H., Saint-Paul, R., Benatallah, B., Casati, F.: A Framework for Managing the Evolution of Business Protocols in Web Services. In: Roddick, J.F., Hinze, A. (eds.) APCCM. CRPIT, vol. 67, pp. 49–59. Australian Computer Society (2007)
10. Wombacher, A., Mahleko, B.: Finding Trading Partners to Establish Ad-hoc Business Processes. In: Meersman, R., Tari, Z., et al. (eds.) CoopIS 2002, DOA 2002, and ODBASE 2002. LNCS, vol. 2519, pp. 339–355. Springer, Heidelberg (2002)