IoT-Based Compliance Checking of Multi-party Business Processes Modeled with Commitments

Marco Montali[1] and Pierluigi Plebani[2(✉)]

[1] Free University of Bozen-Bolzano, Bolzano, Italy
montali@inf.unibz.it
[2] Politecnico di Milano, Milan, Italy
pierluigi.plebani@polimi.it

Abstract. In a multi-party business process, the choreography defines the conversational protocol among the parties, so that the visibility of the parties' private processes is limited to the set of operations required to respect such a protocol. Especially in scenarios where physical resources are exchanged, knowing how a resource owned by a party is managed in the premises of another party is not possible. Thus, possible misalignments can be detected too late. At the same time, IoT is increasingly adopted to enact business processes in many domains: e.g., logistics, manufacturing, healthcare. As, with IoT, smart devices can physically flow through the different parties involved in a process, their sensing capabilities can be exploited to improve the process compliance checking. With this work we propose an approach for compliance checking that mixes commitments and smart devices. Commitments, declaratively defining mutual contractual relationships between parties, drive the configuration of smart devices that, flowing along with the process flow, check their satisfaction and, in case of misalignment, timely inform the involved parties.

Keywords: Multi-party process compliance · Timed commitments · BPMN choreography model · IoT

1 Introduction

In a multi-party business process, to properly achieve the final common goal, the involved participants agree on a process choreography which must be respected when the process is being executed. This requires that the participants enforce their services with respect to the agreed protocol [10]. To this aim, IoT is attracting more and more interest of researchers and practitioners as it can improve the service monitoring capabilities. Indeed, *smart devices* are currently adopted in organizations to analyze the environment in which the service is operating, by equipping them with sensors able to measure some physical phenomenon (e.g., temperature, presence) accurately and continuously to reduce the time-to-repair in case of error. As long as the objective of monitoring is related to its internal

F. De Paoli et al. (Eds.): ESOCC 2017, LNCS 10465, pp. 179–195, 2017.
DOI: 10.1007/978-3-319-67262-5_14

activities, a participant has total control over it. Conversely, in multi-party business processes, an interaction with the other participants means to consume a service offered by an external party and the visibility of what is happening inside the boundary of such external partners is limited to the information that partner offers. This is typical, for instance, in the logistic domain: e.g., a manufacturer gives their products to a courier that promises to deliver them to the final customer but the information about the status of the goods is usually limited to the position with a very coarse-grained (e.g., the city of the last deposit).

Based on this scenario, to improve the compliance checking of a multi-party business process, in this work we assume to couple smart devices to all the physical resources transferred among the different participants. In this way, as the smart device could embed several sensors, the owner of the resources can have a finer-grained data about the status regardless of the participant who is managing them.[1] To support this envisioned scenario, the goal of this work is to propose an approach to improve the definition and the monitoring of requirements that holds between participants in multi-party business processes. The design of the process takes advantage of an extended *BPMN choreography meta-model* able to embed *social commitments*. The resulting choreographies make explicit which conditions/properties shall be brought during their execution. Moreover, commitments explicitly account for the mutual promises/obligations arising when multiple parties interact. The explicit definition of a timed commitment lifecycle proposed in this paper that, to the best of our knowledge, has never being analysed in the literature, allows the commitments to be directly incorporated into a smart device. Thus, it is possible to track of the progression of the system and to check the compliance between occurring events affecting the state of the commitments of interest and the expected lifecycle.

The rest of the paper is organized as follows. Section 2, using a motivating example taken from the logistic domain, discusses the characteristics and the challenges in monitoring a multi-party business processes. Section 3 introduces the approach describing how the commitments are adopted and extended, as well as integrated in a BPMN choreography model. Section 4 provides the formalization of the commitments and their lifecycle validated by some example taken from our running case study. Finally, Sect. 5 discusses the related work, while Sect. 6 conclude the paper outlining possible future work.

2 Motivation

To better motivate the proposed approach, the choreography diagram referring to the logistic domain is reported in Fig. 1. This sample process is enacted by *Sea.Co.*, a seafood company. Every time a *customer* submits an order, which consists of a list of fishes where the quantity for each item and the delivery date are specified. A negotiation phase with the customer checks the feasibility of the delivery date, possibly shifting it to a date where the delivery can be

[1] Due to the technical nature of the proposed solution, the economical aspects are not yet considered in this paper.

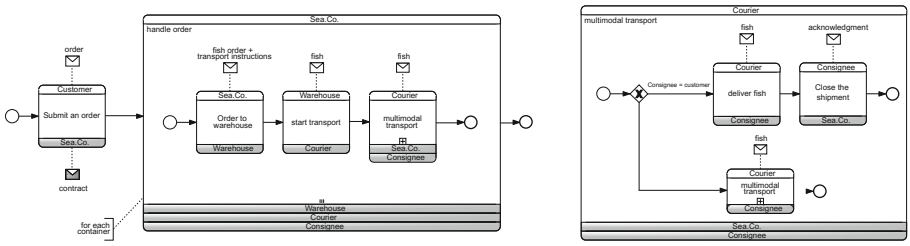

Fig. 1. Running example: BPMN Choreography diagram.

guaranteed, and a contract is finally signed by the two parties. At this point, the Sea.Co. organizes the actual delivery, in particular: *(i)* selecting the fish warehouses (among the various that have the required food units), *(ii)* from each *warehouses* a package is shipped to the customer, *(iii)* splitting the delivery of each package into phases each of them managed by a *courier*, *(iv)* determining which transportation modes are involved, *(v)* ultimately defining a timetable, compatible with the expected delivery date.

Based on this information, several shipments will leave from the selected warehouses to the customer and, according to the defined plan, each delivery could consist of several steps, possibly involving different couriers. Yet, each courier is responsible for a specific phase of the shipment that lasts from the courier premises to the consignee premises. When the consignee corresponds to the final customer, the shipment of the related portion of the order can be considered concluded and an acknowledgment is sent to Sea.Co. Conversely, when the consignee refers to the courier which has to perform the next step in the chain, the same process is recursively repeated. On this basis, each shipment corresponds to different process instances that could differ in terms of activities performed, resources (e.g., trucks) involved, operating actors (e.g., couriers).

As the compliance checking for these internal processes has been extensively studied in the literature [10], the goal of this work is to check the compliance of the choreography: i.e., to check if all the actors operate correctly with respect to the other actors. In fact, due to the complexity of the delivery, deviations to the plan may occur. For example, in case of unexpected traffic, some phase might be dynamically rearranged (e.g., changing the route and/or the transportation mode). This, in turn, may create a ripple effect, requiring to consequently rearrange one or more consequent phases, so as to guarantee that the final delivery date is respected. On the other hand, the contract established between the Sea.Co. and the customer fixes a series of constraints (or, to be more precise, *commitments*) that the involved parties have to, or should, honor no matter how the process is dynamically rearranged. Now, the question is: "how can the Sea.Co. and the customer check the compliance of the process that is being executed?". Generally speaking, this question can be reformulated as: "how can every actor involved in a multi-party business process be enabled to check if the other actors are behaving correctly with respect to the initial agreement?"

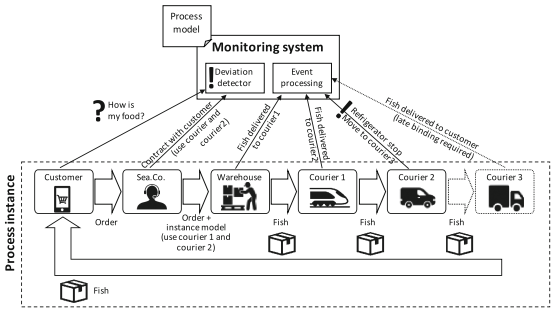

Fig. 2. Centralized monitoring

To address this question, centralized solutions [14] are available (see Fig. 2). In addition to the infrastructure enabling the execution of the process based on the exchange of messages, a central monitoring node is responsible to receive all the updates and to inform about the *process instance* execution, as well as to identify possible deviations with respect to the expected behavior (defined by the *process model*). Although the deviation detection can be not that complex to implement as all the needed information are known, the central node needs to know in advance which will be the entity that will publish or subscribe to the information about the status of the process. Moreover, each entity needs to support the protocols adopted for the communication and if a new entity will be included in the process to manage a deviation, it must adhere to these protocols. For instance, when *Courier2* realizes the refrigerator on the van has broken, it decides to involve *Courier3* to deliver the fish a safe-mode and, to make the centralized approach working, late binding mechanisms are required to make this new actor connected to the monitoring system.

The approach presented in this paper aims to overcome to this limitation extending the usage of smart devices not only to monitor how the tasks operating on a resource are behaving, but also which are the status of the resources. As the resources should move among the participants following what modeled in the choreography, monitoring if the status of the resources give some clue on how the process choreography evolves (see Fig. 3). The adoption of this approach gives two types of advantages. On the one side, instead of leaving to the involved parties the burden of communicating the status of the process instance, autonomic systems implemented on smart devices are paired to the shipping goods to continuously monitor them and, when requested, to inform about the status of the package. Such smart devices are configured by the owner of the goods, i.e., the Sea.Co. company in our realistic scenario, before leaving the warehouses.

Moreover, smart devices are responsible not only to sense the environment in which they are immersed, but they are also configured to host portions of the process model which include the *commitments* stating how, when, and where the smart device should be managed. This, in turn, allows to timely identify possible deviations and establish new, compensating commitments to handle

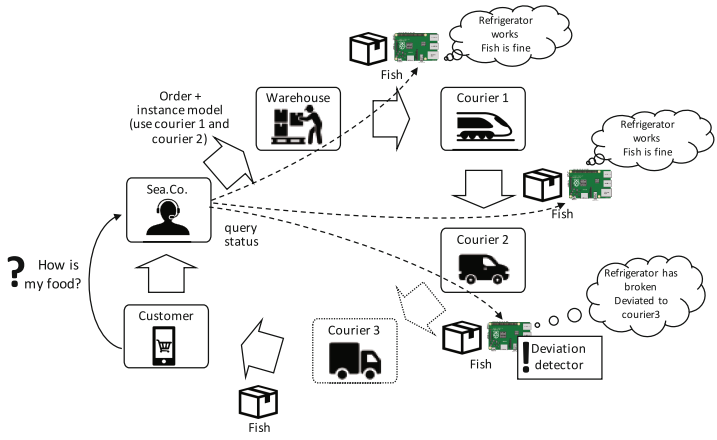

Fig. 3. IoT-based monitoring architecture.

which mutual obligations shall be fulfilled when a deviation is detected. In this way, the knowledge about the status of the process should not be a privilege only of the owner of the smart device, but it can be made available to all the parties[2].

3 Commitments in Multi-party Business Processes

In our approach we advocate the use of (social) commitments [4,5,16] as a way for specifying the conditions under which the multi-party business process should be executed. This section briefly introduces commitments and their lifecycle (also called commitment machine in the literature), and then provides an informal description of how commitments are used in our approach; a more formal definition of commitments, and how they can be managed, is introduced in the next section. The modeled commitments will be used to configure the smart devices, so as to make them able to check if actual instantiations of those commitments are indeed satisfied or not. To informally describe what is a commitment and how it can be useful for our purposes, we adopt the graphical notation introduced in [16] (see Fig. 4). More specifically, a commitment involves two actors: the *debtor*, who is willing to offer a service under certain circumstances, and a *creditor*, who takes advantage of this service. *Antecedent* and *consequent* are two logic expressions which define under which conditions the service must be provided and consumed. Focusing on the lifecycle, a commitment is initially *null* and needs to be created. Once created, if the antecedent does holds it goes to a *detached* state, otherwise in the *conditional* one. The latter represents a state in which the commitment exists but is not yet active, as the antecedent still needs

[2] For the sake of simplicity, this paper does not address privacy issues. These are aspects that definitely need to be investigated in future work.

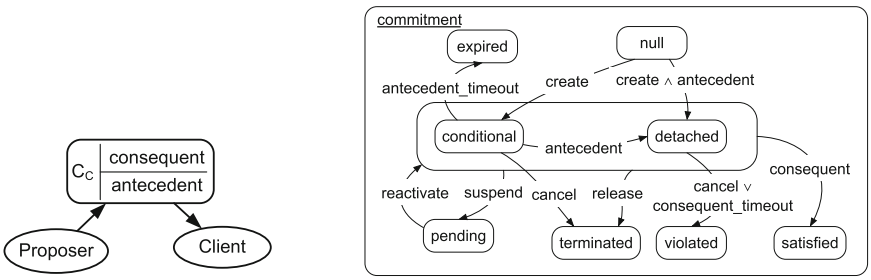

Fig. 4. Commitment notation (left) and lifecycle (right) [16].

to become true and consequently trigger the actual obligation of the debtor to make the consequent true as well. This is what happens in the detached state. When the consequent holds, then the commitment is declared as *satisfied*; it also might happen that the commitment is released or – only if it is in the conditional state – canceled. The operations are under the responsibility of the creditor and debtor of the commitment, and may be employed to flexibly evolve the multi-party interaction. Such a flexibility distinguishes social commitments from normative (in particular, deontic) approaches where obligations are considered in a rigid, immutable way. Timeouts can be also attached to the antecedent and consequent to force their validity for a period of time. Indeed, if the antecedent timeout expires then also the commitment is declared as *expired*. Conversely, if the consequent timeout expires then the commitment is declared as *violated* as the debtor was not able to perform what it has been promised although the pre-condition for its fulfillment (i.e., the antecedent) were holding.

3.1 Commitment Templates

As well-exemplified in this survey [15], commitments are typically used to declaratively capture (business) interactions, abstracting away from control-flow details. In this light, commitment-based approaches are usually considered complementary to activity-/flow-centric ones. The first distinctive feature of our contribution is to establish a synergy between these two paradigms. To do so, we propose an extension of commitments to make them attachable to BPMN choreography models, and in particular to choreography activities. In this way, the choreography takes care of the flow-related constraints, whereas commitments focus on the contractual nature of the collaboration. Specifically, a choreography activity provides the context of existence for certain commitments. This means that, at runtime, whenever an instance for such an activity is executed, corresponding instances for those commitments are created and evolved in accordance with the course of execution. In other words, the lifecycle of choreography activities becomes connected to that of its attached commitments. More details on this aspect, which to the best of our knowledge has never been explored in the past, are given in Sect. 4.1.

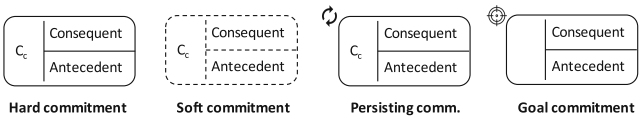

Fig. 5. Types of commitments.

In particular, we introduce four types of commitments classified along two orthogonal dimensions (related shapes are shown in Fig. 5):

- *Importance:* we distinguish between *hard* (solid line) and *soft* commitments (dashed lines). In the former case the consequent must be valid to consider the commitment fulfilled. In the latter case, the creditor is expecting that the debtor will do its best to fulfill the commitment. This distinction provides the basis for a fine-grained handling of commitment violations and corresponding compensations.
- *Time of validity:* the linkage between commitments and choreography activities calls for a distinction between *persisting* (cycle icon decoration) and *goal* commitment (target icon decoration). In the former case, the consequent must be valid *during* the execution of its target activity, possibly even spanning its entire execution. In the latter case, the consequent must become valid when the activity completes.

To discuss our extension more formally, we introduce the concept of *commitment template*: a schema for a multitude of "ground" commitments reflecting the same contractual relationship, but instantiated on different activity instances, that is, possibly different actual participants and/or timestamps and/or targeted objects. This reflects the dual nature of commitments: at design time, as modeling abstractions to capture "types" of business relationships, and at runtime, as computational abstractions to track the evolution of "instances" of such relationships. The importance of this duality has been increasingly recognized in the literature, constituting an interesting point of departure from standard logical approaches to commitments [4,6,12].

In our setting, the notion of commitment template is used to extend the standard BPMN choreography meta-model, as depicted in Fig. 6. Concretizing

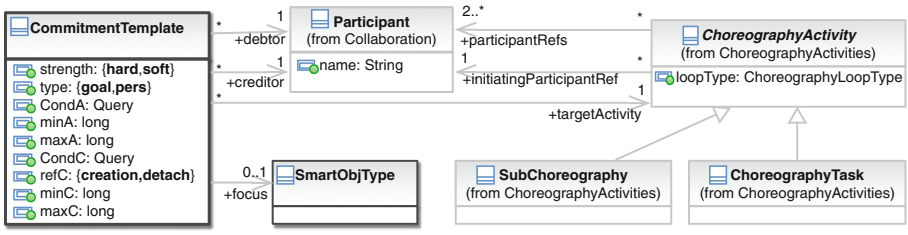

Fig. 6. Commitment-aware extension of the BPMN choreography metamodel

what discussed above, the entity type `CommitmentTemplate` captures a commitment template by declaring its target `ChoreographyActivity`. Among the `Participant` (types) referenced by the choreography activity, two are selected as debtor and creditor of the commitment template. This induces the constraint that, at runtime, each instance of the commitment template will relate a debtor d and a creditor c, with the constraint that d and c participates to the activity playing the corresponding roles attached to the commitment template. Consider, e.g., a template established between a `Warehouse` and a `Courier` in the context of the choreography activity `start transport`. At runtime, commitment instances for that template will be created and evolved by relating actual couriers and warehouses, in turn involved in the execution of instances of `start transport`. Alongside `CommitmentTemplate`, we also extend the choreography meta-model with the notion of `SmartObjType`, which models a type of smart object that may exist in the system. It is then possible to (optionally) declare the focus of a commitment template, relating it to a smart object type. This association has a twofold nature: on the one hand, it explicitly tracks whether the reason/subject of a commitment corresponds to a physical (smart) object; on the other hand, it provides a context for querying the characteristics/data of such an object. This, in turn, provides the basis for defining the antecedent and consequent of the commitment template. Additionally, a commitment template comes with a number of attributes (cf. Fig. 6). We review them one by one. The `strength` of a commitment template indicates whether the commitment is hard or soft, whereas the `type` indicates whether the template has a goal or persistence nature. Such two attributes determine the graphical appearance of commitment templates, as specified in Fig. 5. The two attributes `CondA` and `CondC` respectively identify the antecedent and consequent conditions of the commitment template. Such conditions may be concretely specified in different query languages, possibly expressed over the attributes/properties of a smart object type. Such query languages may range from standard SQL when commitments insist over relational data (such as, e.g., in the case of [6,12]), to query languages over dynamically evolving data such as the CQL continuous query language or proprietary languages to query sensor data provided by smart objects. For the sake of generality, we abstract away from the specific query language at hand. The remaining attributes are used to express quantitative temporal constraints on the commitment template. These are used to refine the representation of the antecedent and consequent, defining relative temporal windows within which they are checked. Specifically, `minA` and `maxA` respectively denote the minimum and maximum delay within which the antecedent condition has to be achieved so as to detach the commitment. The reference point for these two extremes is the time at which the commitment is created, which coincides with the starting time of an instance of its target activity starts.[3] Similarly, `minC` and `maxC` respectively denote the minimum and maximum timestamp within which the consequent condition has to be achieved or maintained so as to declare the commitment as satisfied. For

[3] Absolute temporal constraints can be seamlessly realized as syntactic sugar, scheduling the execution of the target activity at a fixed time.

the `minC`-`maxC` time window, two reference points may be selected: the creation time or the detach time. This is specified through the `refC` attribute. The latter choice is particularly relevant when the time window associated to the consequent has to be determined depending on the exact moment when the commitment was detached, i.e., the moment where a "conditional" obligation turned into an actual one. In the spirit of [13], for goal commitments `minC` represents the minimal delay at which the goal has to be achieved, while `maxC` captures the deadline of the goal; for persistence commitments, instead, the time window delimited by `minC` and `maxC` is the interval within which the consequent is expected to hold. Differently from [13], though, the achievement/maintenance of the commitment consequent are bound to that of its target activity. In this light, goal commitments implicitly impose temporal constraints on when an activity is expected to end, whereas persistence commitments may be released by the completion of an activity.

3.2 Modeling Commitments

The proposed extension of the BPMN choreography metamodel enables the decoration choreography activities with commitments. Thus, a process designer can specify not only the conversation among the parties, but also which are the contractual obligations and their characteristics. By connecting the commitments to a BPMN Choreography model we link the lifecycle of commitments to the lifecycle of the activities. Referring to the example shown in Fig. 7, there is a goal commitment in which the Sea.co. is the debtor, while the consignee is the creditor. As the commitment is attached to the whole activity, and no explicit antecedent is included in the commitment, then the commitment becomes immediately detached when the activity starts. This shows one of the benefits obtained through the commitment-activity linkage. Being a goal commitment, we are expecting that the consequent becomes true when the activity ends. In more details, the diagram is stating that the fish has to be delivered within 25 days. This can be obtained by setting $\texttt{minC} = 0$ and $\texttt{maxC} = 25d$ for the commitment template, with reference point the detach time (which, in this case, coincides with the creation time). As said, in this case the antecedent is implicitly linked to when the activity starts. Similarly, the validity of the commitment is related to the termination of the activity. Thus, if the commitment consequent (i.e., fish delivery) is achieved when the activity ends, then it will be considered as satisfied, otherwise it will be considered violated. This implicitly sets a deadline on the `handle order` activity, since whenever it takes more than 25 days, then the commitment becomes violated.

In the same process, the warehouse and the first courier agrees on another commitment. In this case, being a soft commitment, the *start transport* activity should possibly be executed in 5 days. Similarly to the previous case, this goal commitment moves to the *detached* state when the fish is ready to leave the warehouse, while it can be considered satisfied if the consequent is verified, i.e., when the food is on board of the first courier. This latter condition can be specified by querying

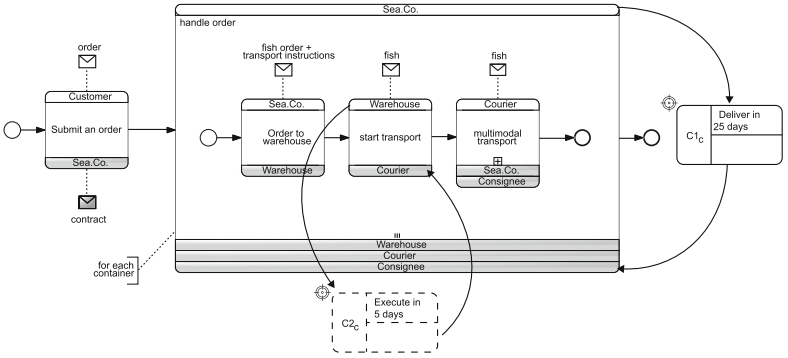

Fig. 7. Example of hard/goal and soft/goal of commitments.

a positioning sensor for the smart device attached to the food container, or by simply checking when the start transport activity completes.

When the antecedent is specified, like the case in Fig. 8, the activation of the commitment occurs when the activity starts and the antecedent becomes true (maybe at a later time). In our running example, this occurs when the courier responsible of a transportation phase signals that the refrigerator used to transport the fish is broken. If multiple couriers are involved in the SeaCo-to-customer transportation, each one will be attached to an instance of the *multimodal transport* activity, and in turn to an instance of such a commitment template. When the refrigerator of a courier gets broken, a corresponding instance of such a commitment is detached and, contrarily from the previous cases, starts monitoring the maintenance of a property related to the fish temperature, being a persisting commitment. In particular, the consequent in this case is not expected to hold when the multimodal transport activity ends, but for the whole time window that spans from detach moment, to that marking the completion of the activity. This means that, while the multimodal transport activity is under execution,

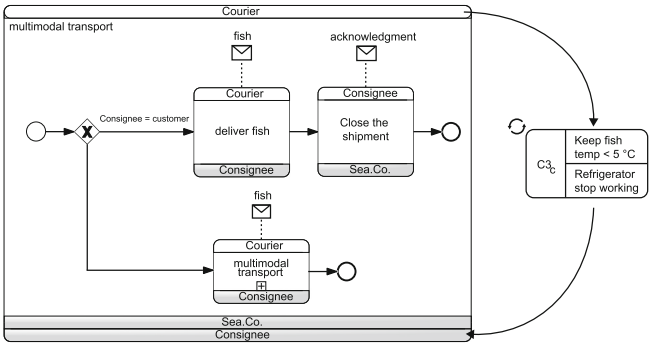

Fig. 8. Example of hard, persisting commitment.

as soon as the fish temperature reaches 5 °C, the commitment instance will become violated. Also in the case, we may declare that the commitment template focuses on a type of smart object that is attached to a fish container, and is equipped with a sensor providing timely information about the fish temperature.

4 Tracking Commitments

When the commitments are applied to a physical resource that is exchanged among the parties, we propose to use smart devices to monitor if the resource is managed according to the defined commitments. When doing this, we need to be sure that the smart device is able to understand if the actors that are managing the resources are respecting the defined obligations. Before entering into the details of the timed-commitment lifecycle which puts the formal basis for managing the evolution of a commitment (that has been informally introduced in the previous section), it is fundamental to clarify how, starting from a BPMN Choreography model extended with commitments, is possible to derive the associations between smart devices and commitments to be tracked. Assuming that for each resource to be monitored one smart device is used, the configuration of the smart device D^R related to a resource R requires to perform the following steps:

- *Identification of relevant activities:* being A the set of `Choreography Activity`, $A^R \in A$ corresponds the subset of `ChoreographyActivity` for which the resource is either the receiving or the sending message.
- *Identification of relevant commitments:* being C the set of `Commitment Template`, $C^R \in C$ corresponds to the subset `CommitmentTemplate` for which the `debtor` or the `creditor` refers to one of the `Participant` in A^R.

Being C^R the commitments to be tracked by the smart devices D^R, we assume that the smart device supports the needed capabilities to check the antecedent and the consequent of these commitments: e.g., the smart device monitoring a fish package will have a sensor for temperature on-board, and it is able to recognize (manually or automatically) when an activity starts or ends. Once deployed on a smart device, the tracking of a commitment is possible by considering the evolution of a commitment template as expressed by the timed commitment lifecycle formalized in the next section.

4.1 Timed Commitment Lifecycle

Consider a specific commitment template, indicating its target activity and debtor/creditor types, and providing values for its various attributes. At a given time, an instance of such a commitment can be in one of the states depicted in its commitment lifecycle (cf. Fig. 4). We now formally ground this abstract lifecycle, indicating when, and how, a transition between states occur. More

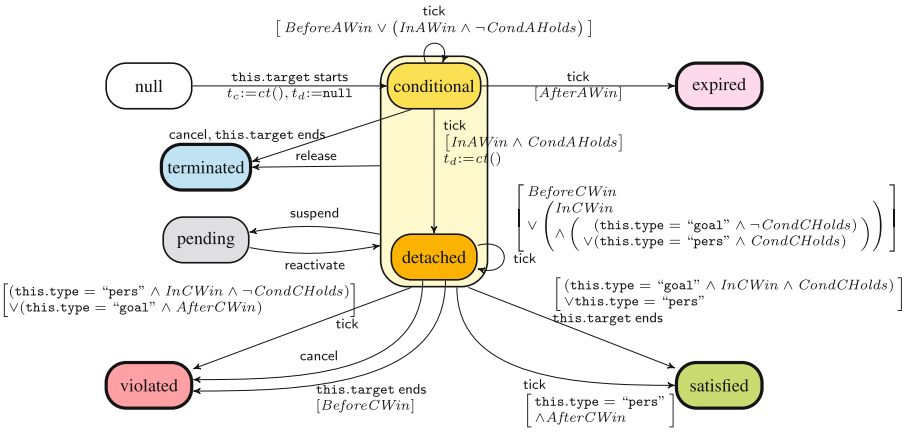

Fig. 9. Formalization of the timed commitment lifecycle with a target activity; *tick* denotes an arbitrary event, just used to inform the commitment machine about the current time.

specifically, a transition occurs in response to events, possibly depending on the validity of the commitment antecedent/consequent. We consider three types of events. First, we have *activity-related events*, i.e., the `start` and `end` of (an instance of) the choreography activity targeted by the commitment. Second, we have *explicit commitment manipulation events*, used to `suspend`, `release`, `reactivate`, or `cancel` a commitment instance. Interestingly, such events may actually be automatically generated in response to events issued on the activity lifecycle. For example, the designer may decide that whenever a choreography activity instance is suspended, then all commitment instances attached to it will be suspended, too. This is just an example of the benefits of our approach. Third, ***`Tick`*** *events*, represents the current time flowing. These events are useful to communicate the new current time to the commitment lifecycle, and in turn evaluate the quantitative temporal conditions attached to it [4]. Ticks may be internally generated, or communicated from the external environment, based on who is aware of the flow of time.

With these events at hand, we devise the timed commitment lifecycle of Fig. 9, where the keyword `this` refers to the specific commitment template of interest, function $ct()$ returns the time associated to the currently processed event, while t_c and t_d get respectively assigned the time at which the commitment is created or detached. Our approach formalizes the abstract diagram of Fig. 4 with concrete, testable transitions, employing the following macros[4]:

[4] The commitment machine we propose enriches standard commitment machines from the literature, adding temporal conditions on transitions. It is worth noting that, as usual in the commitment literature, the interpretation of such different states, and the corresponding set up of reactions, sanctions, and countermeasures, has to be handled in a domain-specific way on top of the commitment machine, not within the machine itself.

- *BeforeAWin* = $ct() \leq t_c + \texttt{this.minA}$ checks that the current time is before the antecedent time window.
- *InAWin* = $ct() > t_c + \texttt{this.minA} \wedge ct() \leq t_c + \texttt{this.maxA}$ checks that the current time falls within the antecedent time window.
- *AfterAWin* = $ct() \geq t_c + \texttt{this.maxA}$ checks that the current time is after the antecedent time window.
- *BeforeCWin*, *InCWin* and *AfterCWin* reconstruct the previous three macros for the consequent time window. The additional complication, here, is that the reference point depends on the `this.refC` attribute. E.g., *InCWin* is formalized as:

$$\begin{cases} ct() > t_c + \texttt{this.minC} \wedge ct() \leq t_c + \texttt{this.maxC} & \text{if } \texttt{this.refC} = \texttt{creation} \\ ct() > t_d + \texttt{this.minC} \wedge ct() \leq t_d + \texttt{this.maxC} & \text{if } \texttt{this.refC} = \texttt{detach and } t_d \neq \texttt{null} \end{cases}$$

- *CondAHolds* and *CondCHolds* are respectively true if `this.condA` and `this.condC` hold at time $ct()$.

We briefly comment on the formalization. Call *active* a commitment (instance) that is either *conditional* or *detached*. A commitment instance becomes active when an instance of its target activity starts. Specifically, the commitment instance becomes conditional or detached depending on whether its antecedent condition evaluates to true at the creation time, and its antecedent time window has a minimum displacement of 0 (`minA` = 0). A *conditional* commitment instance becomes:

- *Expired* as soon as the deadline of its antecedent time window, calculated w.r.t. its creation time, is over.
- *Terminated* if its target activity instance ends (marking the fact that the commitment instance never required an actual obligation to be fulfilled).
- *Detached* when, within its associated antecedent time window (calculated w.r.t. the creation time), its antecedent condition evaluates to true.

The explicit cancellation of an active commitment instance has the effect of terminating or violating the commitment instance, depending on whether it has been detached or not. The other transitions of an active commitment instance depend on whether it has a goal or persistence nature. In the first case, it becomes:

- *Violated* as soon as the consequent time window (calculated w.r.t. the creation or detach time depending on the `refC` attribute) expires, witnessing that the target activity instance has not completed on time. If the commitment instance is detached, also a premature completion of the activity instance leads to violation.

- *Satisfied* if its corresponding activity instance completes on time, and in a moment in which the consequent condition holds.

Conversely, in the latter case, it becomes:

- *Violated* when, during the consequent time window, the consequent condition becomes false, thus witnessing that the promised condition has not been maintained.
- *Satisfied* as soon as the consequent time window passes or its corresponding activity instance is completed, witnessing that the consequent condition has been continuously maintained until this time point.

Notice the complementary behavior of goal vs. persistence commitments in Fig. 9, when the commitment is detached. A goal commitment is satisfied if its target activity is completed at a time that falls within the consequent time window, and at which the consequent holds; it is instead violated if the deadline of the consequent time window expires while the commitment is still in the detached state. Contrariwise, a persistence commitment is violated during the consequent time window as soon as the consequent is not maintained anymore, whereas it gets automatically satisfied if the commitment is still detached when the consequent time window expires.

4.2 Implementation

The lifecycle presented in the previous section can be directly used as an actual computational artifact during the system execution, tracking the evolution of commitment instances as new events occur. When the commitment instance resides on a smart object, checking the antecedent/consequent amounts to issue the corresponding query on the data maintained the object, and/or retrieved through its sensors. The actual implementation obviously depends on the specific programming language of choice, and the computational resources available. To show the feasibility of the implementation, we have encoded the different transition rules of the lifecycle in the (Reactive) Event Calculus (REC) [3], a logic-based calculus of events that has been already used to formalize and monitor business constraints [13] and timed commitments [4]. The query language to express commitment conditions is in this case natively provided by REC itself.

The complete formalization in REC, together with the encoding of our case study (cf. Sect. 3.2) and its embedding into a monitoring test application, can be downloaded from http://tinyurl.com/kd8wtre. Figure 10 shows the result produced by REC on a hypothetical partial run of our case study.

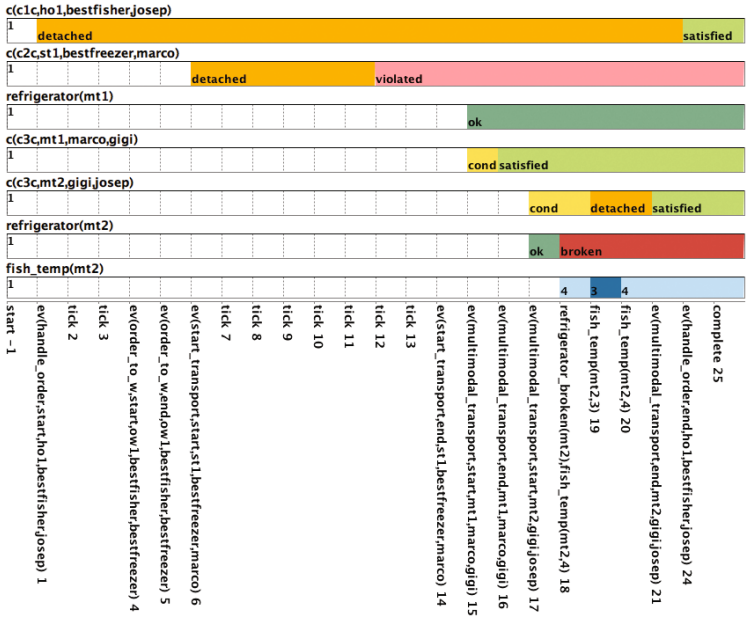

Fig. 10. Monitoring timed commitment instances in REC.

5 Related Work

Checking the compliance of a business process requires to verify that the execution of a process is respecting what has been conceived by the process designer. In the literature, there are several approaches and solutions able to cope with this issue and [10] organizes them in a systematic literature review. Among the dimensions of analysis, the survey discusses the compliance monitoring functionality that a monitoring system should support and in particular the importance of considering time, data, and resources in the constraints. Going towards this direction, and similarly to [2] where collaborative processes modeled with BPMN has been extended to include monitoring instructions, our approach extends the BPMN choreography model to attach commitments where constraints on time, data, and resources are possible to be defined.

Focusing more on the peculiarities of cross-organizational processes, [9] has identified some research challenges among which there is the need to model cross-organizational compliance rules. To this aim, we rely on commitments [15], exploited in [16] to model the interaction among several participants inspired by the agent-based system literature, and translated into automaton as suggested in [7,12]. At the same time, [8] focuses on the way in which the compliance rules are specified and verify if there are not conflicts between them. Even though approaches for monitoring timed extensions of commitments have been already proposed in the past [5,15,16], the explicit definition of a timed commitment lifecycle proposed in this paper, to the best of our knowledge, has never been devised.

Once the constraints are modeled, their verification can be done a-posteriori, through log analysis [1], or at run-time [11]. Our approach is close to the second case and, as a element of novelty, we assume to exploit smart devices to perform the compliance checking. Indeed, smart devices are now adopted to execute some of the tasks composing a business process, as well as to monitor the status of the resources manged in the process [17,18]. As their computational power is getting more and more significant, we investigated the possibility to exploit this capabilities.

6 Conclusions

In this work, we have introduced an approach for checking the compliance of a multi-party business process by extending BPMN choreography model with timed commitments. Classical commitments have been extended in this work to consider hard and soft constraints as well as persisting and goal commitments. The resulting enriched choreography model can be used to properly configure smart devices that will be in charge of checking the validity of those commitments due to the proposed lifecycle of extended commitments. Although this approach is in its infancy, we can now check possible deviations of process instance in a distributed way exploiting smart devices, inheriting the constraints defined at design-time. Nevertheless, there are several limitations that need to be addressed in future work. Firstly, although if the control-flow that can be defined for a choreography model is more simple than what possible to express in a collaboration diagram, how to manage switches and loops is currently an open issue that needs to be investigated. Furthermore, the proposed approach lives on the assumption that the communication is always up, and the smart device is always reachable. We will extend or approach to by considering reliability and communication failures.

Acknowledgments. This work has been partially funded by the Italian Project ITS Italy 2020 under the Technological National Clusters program and by the UNIBZ CRC Project *Planning for WORkflow Management* (PWORM).

References

1. van der Aalst, W.M.P., de Beer, H.T., van Dongen, B.F.: Process mining and verification of properties: an approach based on temporal logic. In: Meersman, R., Tari, Z. (eds.) OTM 2005. LNCS, vol. 3760, pp. 130–147. Springer, Heidelberg (2005). doi:10.1007/11575771_11
2. Baumgraß, A., Herzberg, N., Meyer, A., Weske, M.: BPMN extension for business process monitoring. In: Proceedings of International Workshop on Evolution of Information Systems and their Design Methods (EMISA 2014) (2014)
3. Bragaglia, S., Chesani, F., Mello, P., Montali, M., Torroni, P.: Reactive event calculus for monitoring global computing applications. In: Artikis, A., Craven, R., Kesim Çiçekli, N., Sadighi, B., Stathis, K. (eds.) Logic Programs, Norms and Action. LNCS, vol. 7360, pp. 123–146. Springer, Heidelberg (2012). doi:10.1007/978-3-642-29414-3_8

4. Chesani, F., Mello, P., Montali, M., Torroni, P.: Representing and monitoring social commitments using the event calculus. J. Auton. Agents Multi-agent Syst. **27**(1), 85–130 (2013)
5. Chopra, A.K., Singh, M.P.: Generalized commitment alignment. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015)
6. Chopra, A.K., Singh, M.P.: Cupid: commitments in relational algebra. In: Proceedings of the 29th AAAI Conference on Artificial Intelligence. AAAI Press (2015)
7. Ferrario, R., Guarino, N.: Commitment-based modeling of service systems. In: Snene, M. (ed.) IESS 2012. LNBIP, vol. 103, pp. 170–185. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28227-0_13
8. Knuplesch, D., Reichert, M., Fdhila, W., Rinderle-Ma, S.: On enabling compliance of cross-organizational business processes. In: Proceedings of the International Conference on Business Process Management (BPM 2013) (2013)
9. Knuplesch, D., Reichert, M., Mangler, J., Rinderle-Ma, S., Fdhila, W.: Towards compliance of cross-organizational processes and their changes. In: La Rosa, M., Soffer, P. (eds.) BPM 2012. LNBIP, vol. 132, pp. 649–661. Springer, Heidelberg (2013). doi:10.1007/978-3-642-36285-9_65
10. Ly, L.T., Maggi, F.M., Montali, M., Rinderle-Ma, S., van der Aalst, W.M.: Compliance monitoring in business processes: functionalities, application, and tool-support. Inf. Syst. **54**, 209–234 (2015)
11. Maggi, F.M., Montali, M., Westergaard, M., van der Aalst, W.M.P.: Monitoring business constraints with linear temporal logic: an approach based on colored automata. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 132–147. Springer, Heidelberg (2011). doi:10.1007/978-3-642-23059-2_13
12. Montali, M., Calvanese, D., De Giacomo, G.: Verification of data-aware commitment-based multiagent systems. In: Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS) (2014)
13. Montali, M., Maggi, F.M., Chesani, F., Mello, P., van der Aalst, W.M.P.: Monitoring business constraints with the event calculus. ACM TIST **5**(1), 17 (2013)
14. Sahai, A., Machiraju, V., Sayal, M., van Moorsel, A., Casati, F.: Automated SLA monitoring for web services. In: Feridun, M., Kropf, P., Babin, G. (eds.) DSOM 2002. LNCS, vol. 2506, pp. 28–41. Springer, Heidelberg (2002). doi:10.1007/3-540-36110-3_6
15. Singh, M.P.: Commitments in multiagent systems: some history, some confusions, some controversies, some prospects. In: The Goals of Cognition: Essays in Honor of Cristiano Castelfranchi, pp. 601–626. College Publications (2012)
16. Telang, P.R., Singh, M.P.: Specifying and verifying cross-organizational business models: an agent-oriented approach. IEEE Trans. Serv. Comput. **5**(3), 305–318 (2012)
17. Thoma, M., Meyer, S., Sperner, K., Meissner, S., Braun, T.: On IoT-services: survey, classification and enterprise integration. In: 2012 IEEE International Conference on Green Computing and Communications (2012)
18. Tranquillini, S., Spieß, P., Daniel, F., Karnouskos, S., Casati, F., Oertel, N., Mottola, L., Oppermann, F.J., Picco, G.P., Römer, K., Voigt, T.: Process-based design and integration of wireless sensor network applications. In: Barros, A., Gal, A., Kindler, E. (eds.) BPM 2012. LNCS, vol. 7481, pp. 134–149. Springer, Heidelberg (2012). doi:10.1007/978-3-642-32885-5_10