An Artifact-Driven Approach to Monitor Business Processes Through Real-World Objects

Giovanni Meroni[1(✉)], Claudio Di Ciccio[2], and Jan Mendling[2]

[1] Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milan, Italy
giovanni.meroni@polimi.it
[2] Institute for Information Business, Vienna University of Economics and Business, Vienna, Austria
{claudio.di.ciccio,jan.mendling}@wu.ac.at

Abstract. Nowadays, many business processes once intra-organizational are becoming inter-organizational. Thus, being able to monitor how such processes are performed, including portions carried out by service providers, is paramount. Yet, traditional process monitoring techniques present some shortcomings when dealing with inter-organizational processes. In particular, they require human operators to notify when business activities are performed, and to stop the process when it is not executed as expected. In this paper, we address these issues by proposing an artifact-driven monitoring service, capable of autonomously and continuously monitor inter-organizational processes. To do so, this service relies on the state of the artifacts (i.e., physical entities) participating to the process, represented using the E-GSM notation. A working prototype of this service is presented and validated using real-world processes and data from the logistics domain.

Keywords: Artifact-driven process monitoring · Physical artifacts · E-GSM · Inter-organizational monitoring service · Autonomous process monitoring

1 Introduction

In recent years, a large number of organizations opted to outsource some of their business services to external service providers, either partially or entirely [12]. By doing so, many traditionally intra-organizational business processes have become inter-organizational. The adoption of this strategy has brought several advantages. For example, organizations can now focus on their core business, rather than having to deal with support processes, e.g., logistics. Furthermore, specialized service providers usually deal with the externalized processes more efficiently and effectively than internal divisions of organizations operating on different markets. However, outsourcing has also brought some issues, one of which is the inability for an organization to directly control how the outsourced processes are executed. It is up to the service provider to execute these processes as agreed with

M. Maximilien et al. (Eds.): ICSOC 2017, LNCS 10601, pp. 297–313, 2017.
https://doi.org/10.1007/978-3-319-69035-3_21

the organization. In such a case, a service capable to constantly monitor the execution of inter-organizational processes becomes crucial. A process monitoring service allows an organization to know a.o. *(i)* when business activities composing the process are executed, and *(ii)* if their execution order complies with the process model, namely the formal specification of how the process should be performed. This way, countermeasures can be taken in case violations in the execution occur, and a better coordination among the organization and the service providers can be achieved.

Traditionally, monitoring services are included in Business Process Management Systems (BPMSs), namely the software components responsible for automating the execution of business processes [10]. However, a BPMS presents shortcomings when monitoring inter-organizational processes. Firstly, unless an activity is completely automated and fully executed by the BPMS, human operators have to manually notify the BPMS that an activity starts or ends. Such a task disrupts the operator's work, and can be easily forgotten or postponed, thus negatively affecting the reliability of the monitoring. Secondly, whenever the process is not executed as agreed, BPMS usually halt the execution of the process until the violation is manually solved by a human operator. Consequently, the process execution is not tracked until the violation is solved. This is undesirable: In an inter-organizational process, service providers could continue running their processes even though the BPMS halted. Such an issue can be partially mitigated by instructing the BPMS not to halt in case of violations, so as to successively resort to mining techniques to detect the disruptions in the recorded execution log. However, such an approach impedes an organization to promptly react to violations.

To overcome these issues, we propose a novel monitoring service which can autonomously *(i)* monitor the execution of non-automated activities, as long as they interact with machine-tracked real-world objects, and *(ii)* identify incorrectly executed activities yet continue monitoring the process after a violation occurs. The approach we present is built upon the usage of the Extended-GSM (E-GSM) artifact-centric language [20] for the automated monitoring of processes. Our approach is implemented with a software prototype. We demonstrate the efficacy of our approach with an application on a real-world use case from the logistics domain.

The remainder of this paper is structured as follows. Section 2 introduces a motivating example used to describe, in Sect. 3, our approach. The architecture of a monitoring service based on our approach is discussed in Sect. 4. Section 5 validates our work against real processes and data. Finally, Sect. 6 surveys related work and Sect. 7 concludes the paper outlining the future research plans.

2 Motivating Example

To better understand the need for an inter-organizational monitoring service, we focus on a real scenario taken from the logistics domain, which will be used throughout this paper. However, logistics is only one of the possible case studies.

In fact, our solution is beneficial to every inter-organizational business process interacting with real-world objects.

A manufacturer located in the United Kingdom, M, has a long-term provisioning contract with customer C, located in Germany. To send its goods to C, M relies on logistics company L, headquartered in Amsterdam, which owns several inland terminals located nearby the principal airports of Europe. Instead of performing the actual shipments, L outsources them to several truck shippers S, each one responsible for one or more legs (i.e., for all the shipments from the headquarters to a specific terminal and vice-versa). The shipment process from M to C is organized as follows. At first, a container is shipped from the plant of M to a terminal located near London Heathrow Airport, which serves the UK market. We call this leg M-TU. Then, the container is shipped to the headquarters of L (TU-HQ leg). After that, the container is shipped to a terminal located near Frankfurt, which serves the German market (HQ-TG leg). Finally, the container is delivered to C (TG-C leg).

The TU-HQ leg is organized as follows. Firstly, the container is loaded onto a truck of S (Load container), which subsequently starts traveling in the UK (Travel in UK) until either a break is taken (Take break in UK), or the entrance to the Channel tunnel is reached. The alternation of traveling hours with breaks forms a loop which we name UK Loop. In the first case, once the break ends, the truck continues traveling in the UK. In the second case, the truck takes the Channel tunnel (Take Channel tunnel), then continues traveling on continental Europe (Travel in EU) until either it reaches the headquarters, or it takes a break (Take break in EU). We name this loop of travel and breaks within continental Europe as EU Loop. In the first case, the container is unloaded (Unload container) and the process ends. In the last case, the truck continues traveling in Europe once the break ends. The other legs are organized similarly. Once the container is loaded, the truck starts traveling (either in the UK or in continental Europe, depending on the location of the leg) until either the destination is reached, or a break is taken. Similarly to the TU-HQ leg, once the break ends, the truck starts traveling again.

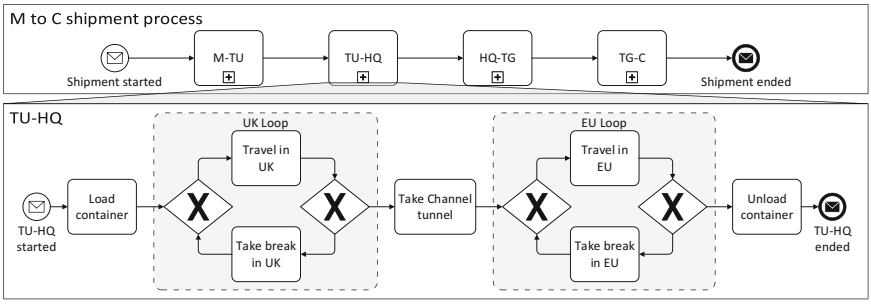

Fig. 1. BPMN diagram of the running example: High-level process model (top), and expanded subprocess TU-HQ (bottom).

The upper part of Fig. 1 depicts the whole shipment process using the Business Process Model and Notation (BPMN) language. The lower part of Fig. 1, on the other hand, depicts the TU-HQ leg. It is worth noting that none of the involved organizations has full control of the execution of the whole process. Since each leg is outsourced to a different truck shipping company S, S controls only those activities inside its own leg, and cannot alter the execution of the other ones. P, C, and L, who are the only organizations interested in the whole process, have no direct control on it. Therefore, to allow each organization to know how the whole process is being run, a monitoring service is needed.

3 Approach

The underlying idea of our approach is that the execution of an activity involving real-world objects is reflected in the modification of their status. In the example of Fig. 1, e.g., the loaded truck updating its position from the European end of the Channel tunnel towards Frankfurt in the physical world indicates the enactment of activity Travel in EU in the sub-process TU-HQ. Updates on the status of trucks are typically provided by AIS/GPS on-board units to the systems of the logistic control rooms. The transmitted information is elaborated and can lead in the process environment to a change of state of the related artifacts. The platform can thus observe the real-world objects involved in the process execution, and compare the evolution of their status with the expected enactment of the process. This allows for a monitoring that does not require a human intervention to signal the progress of process instances. When the process instances' execution differs from the prescribed one, a violation is detected. The platform becomes aware of such a discrepancy when the observed artifacts' state changes do not match with the model of the running process. It can identify which activities are affected, flag them as non compliant, and alert the involved stakeholders.

We propose a four-steps procedure to provide the necessary information. The first step is taking as input a BPMN process diagram, one of the most used formalisms for process modeling, representing the process to be monitored. The second step requires the designer to enrich the BPMN diagram by including information on the artifacts participating in the process. The third step automatically translates the BPMN diagram into an E-GSM process, suited for monitoring distributed processes. The fourth step automatically defines criteria to map real-world objects to the artifacts at runtime. This way, organizations can reuse existing process models, without having to learn new languages and remodel processes from scratch. Our approach poses the following three main requirements.

R1. *The platform must be made aware of the process model and the involved artifacts.* Such an input can be provided at deploy time for the process.

R2. *The platform must be made aware of the physical entities to observe.* The second requirement pertains to the run-time link between real-world objects and artifacts. Not the same truck will be used for all deliveries: Different real-world objects may embody the same process artifact. However, it may not be

possible to know at design-time which real-world objects will be involved in the carry-out of every process instance. Oftentimes such an information is available only after the process instance started.

Such a binding should be definable at runtime. By the same line of reasoning, the information on the previously involved artifacts may be no longer relevant to the ongoing process at some stage, as in the case of the truck moving away from the logistics company headquarters once activity TU-HQ is concluded. Hence the following requirement.

R3. *The binding and unbinding of physical entities to process instances has to be made declarable.*

In the following, we explain how our solution meets those requirements.

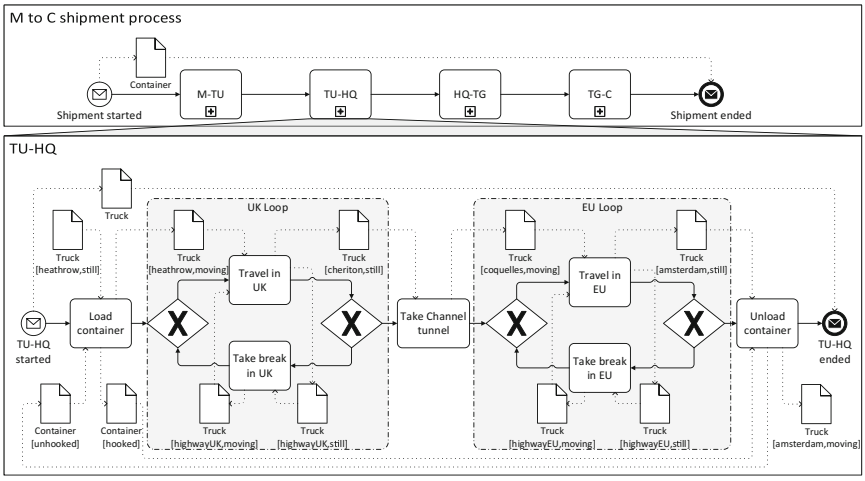

Fig. 2. BPMN process model enriched with information on the participating artifacts.

3.1 Enrichment of the BPMN Process Model with Artifacts

A BPMN process diagram specifies which activities are executed in a process and their control flow relationships. However, to be able to infer when activities start or end based on the state of the artifacts, the diagram must capture this information (requirement R1). Furthermore, the following binding and unbinding mechanisms among artifacts and real-world objects must be specified in the diagram: *(i)* When an artifact starts interacting with the process (R3); *(ii)* How the object impersonating the artifact is notified to the process (R2); *(iii)* When an artifact is no longer related to the process (R3).

To this extent, we resort on the standard BPMN *data objects*, rather than introducing yet another extension of BPMN. Data objects traditionally serve for documentation purposes, yet we use them to model the artifacts and their interactions with the process. Moreover, we establish the following set of rules

to guarantee at design-time that the process model contains enough information to completely and unambiguously automate the monitoring of the process at run-time. The explanatory examples provided for the rules are shown in Fig. 2.

- An artifact must be modeled with data objects. The name of the data object identifies the artifact (e.g., Truck), whereas the data state identifies in which condition the artifact is supposed to be (e.g., [highwayUK,moving]).
- Each monitored activity must have at least one input and one output data object. The activity is supposed to start (resp., finish) only when all input (output) data objects exist and have the specified data state. If an activity has two input (output) data objects referring to the same artifact in different data states, the artifact must assume one of the specified states. For example, Travel in UK starts when Truck is either in state [highwayUK,moving], or in [heathrow,moving]. It ends when Truck is either in state [highwayUK,still] or in [cheriton,still].
- For each artifact, at least one output data object with no data state must be defined in the diagram and associated to a start event. The artifact is supposed to begin interacting with the process when that event occurs. Beforehand, the artifact and its state is ignored. The payload of the event indicates the object that instantiates the artifact. In the example, Container starts interacting with the process at its initial event, and Truck is bound to the beginning of TU-HQ.
- For each artifact, zero or more input data objects with no data state can be defined in the diagram and associated to an end event. The artifact is supposed to become unrelated to the process when the event occurs (after such an event, the artifact and its state will be ignored when the process is executed). For instance, Truck will be no longer related to the process once TU-HQ finishes.
- Data associations must not contradict the semantics of the control flow as they are used to identify when activities start or end. For example, Travel in UK cannot be declared to start only when Truck is in [heathrow,moving], otherwise it could not start again after a break along the journey through UK, far from the airport (despite the loop in the process model). Therefore Truck[highwayUK,moving] is set as another input for Travel in UK and as an output of Take break in UK.

Example. Figure 2 shows the process model obtained by extending the one presented in Sect. 2 according to the previously mentioned rules. The input and output data objects of Unload container indicate the preconditions and postconditions for that activity to be executed. To execute Unload container, the container must be hooked to the truck, and the truck must already be parked in the headquarters of L. When Unload container finishes, the container will be unhooked from the truck, and the truck will leave the headquarters of L. As the container participates in the whole process, its data object is associated to the start and end events of the process. On the other hand, a specific truck may only participate to a single subprocess. As such, the data object representing the truck is associated to the start and end events of each subprocess.

3.2 Generation of the E-GSM Process Model

Due to its imperative nature, BPMN treats control flow information in a prescriptive way: The only possible executions of the process are the ones that comply with the control flow. Therefore, no other way of enacting the process can take place than the prescribed ones. This assumption is suitable for intra-organization execution scenarios. However, when it comes to inter-organization monitoring scenarios, a different paradigm is needed in order to deal with deviations that may arise from the different parties involved. To overcome this limitation, we make use of the E-GSM language [3], an extension of the Guard-Stage-Milestone (GSM) notation [15] especially devised for monitoring: E-GSM treats control flow in a descriptive way, and as such it can monitor any possible execution of a process. When a deviation from the control flow is detected, an E-GSM engine flags the part of the process causing such a deviation as non compliant, without halting the monitoring.

In E-GSM the units of work that can be performed when the process is executed are represented by *stages*. Stages can be atomic, thus representing a single task, or can nest other stages, thus representing a process fragment. The conditions that determine when stages become *opened* (the unit of work is being performed) are represented by *data flow guards*, which we will indicate as "`DFG`". The conditions that determine when stages are *closed* (the unit of work is completed) are represented by *milestones*, indicated as "`M`". Each stage must have at least one data flow guard and one milestone attached. Control-flow dependencies among stages are represented by *process-flow guards*, henceforth identified by the acronym "`PFG`". They are assessed before a stage becomes opened. If they are evaluated as false, the stage is flagged either as *out of order* (executed although it should not) or *skipped* (not executed when it should). Starting from the enriched BPMN process model obtained in the previous step, an E-GSM model of that process can be automatically produced. To do so, we apply the following translation rules. They are based upon [4], which we extend to detect when activities are executed based on the state of the artifacts. The effect of the application of such rules on the model of Fig. 2 is shown in Fig. 3.

- Given a BPMN atomic activity (e.g., Unload container), a corresponding E-GSM stage is produced (e.g., `UnloadContainer`).
- For each artifact Ar, if a change in its state occurs, events are raised to signal that it leaves the previous state (henceforth denoted as Ar^l) and enters the current one (Ar^e). For instance, when Truck transitions from [heathrow,still] to [heathrow,moving], events $Truck^l$ and $Truck^e$ are produced. $Truck^l$ is raised when Truck leaves [heathrow,still], and $Truck^e$ is raised when it enters [heathrow,moving].
- The data flow guard (milestone) of a stage is evaluated on Ar^e (Ar^l) for each artifact Ar associated with each input (output) data objects of Ar. The stage is *opened* (*closed*) if the state assumed by all Ar's is the one indicated by the input (output) data objects of the associated activity. For example, `LoadContainer.DFG1` is evaluated when $Container^e$ or $Truck^e$ occur. `LoadContainer` is opened if Container is [unhooked], and Truck is in [heathrow,still].

– Given a BPMN event *E* (e.g., *TU-HQ started*), a stage is produced (e.g., `TU-HQStarted`). One data flow guard and one milestone, both requiring *E* to be raised, are attached to the stage. This way, `TU-HQStarted` is opened and immediately closed when *TU-HQ started* occurs.
– As discussed in detail in [4], the BPMN model is decomposed into nested process blocks identified by *(i)* control-flow patterns (e.g., the loop blocks UK Loop and EU Loop, containing the fragments of the process with a structured loop), *(ii)* subprocess activities (e.g., TU-HQ). Each block B is translated into a stage B`S` that encloses the inner stages derived from activities, events or process blocks therein. The data flow guard of the block-stage B`S` is the union of the data flow guards of the inner stages, whereas the milestone of B`S` and the process flow guard of the inner stages reflect the control flow pattern expressed by B. For instance, TU-HQ is translated into a stage `TU-HQSeq` containing `TU-HQStarted`, `LoadContainer`, `UKLoop`, `TakeChnTunnel`, `EULoop`, `UnloadContainer` and `TU-HQEnded`. `TU-HQEnded.DFG1` is fulfilled only if the control flow is respected, i.e., `TU-HQEnded` is executed only once and immediately after `UnloadContainer` ends.

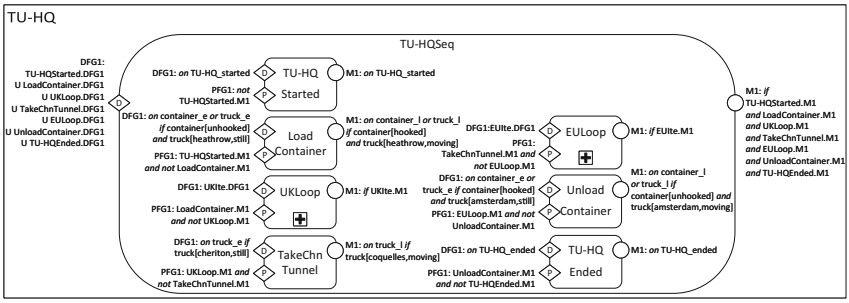

Fig. 3. E-GSM process model derived from the TU-HQ subprocess. For the sake of clarity, stages inside UK Loop and EU Loop are omitted.

Example. Figure 3 shows the E-GSM process model derived from the BPMN process model of Fig. 2. Here, `UnloadContainer.DFG1` is evaluated whenever the artifacts Truck or Container change their state, thus generating events *Truck*^{l} or *Truck*^{e}. To mark `UnloadContainer` as opened (i.e., to represent the fact that the container is currently being unloaded from the truck), `UnloadContainer.DFG1` requires that Truck is in [amsterdam,still], and Container is [hooked]. `UnloadContainer.M1` is evaluated when Truck or Container change their state, thus generating events *Truck*^{l} or *Container*^{l} respectively. To mark `UnloadContainer` as closed (i.e., to signal that the unloading of the container finished), `UnloadContainer.M1` requires that Truck is in [amsterdam,moving], and Container is [unhooked]. Finally, to ensure that `UnloadContainer` is executed at the right time, `UnloadContainer.PFG1` requires that `UnloadContainer` has not already

been executed (thus requiring `UnloadContainer.M1` not to be achieved). Also, `UnloadContainer.PFG1` needs that `EULoop` (directly preceding `UnloadContainer`) has already been executed, hence that `EULoop.M1` was achieved.

3.3 Generation of the Artifact-to-object Mapping Criteria

The E-GSM model generated in the previous step allows us to detect when activities are executed based on the state of the artifacts participating to the process. However, the E-GSM model does not indicate which real-world object will impersonate each artifact (e.g., the artifact Truck is impersonated by the physical truck having license plate "AB123XY"). We capture the mapping criteria among artifacts and objects in a separate document. Such a choice allows us to decouple the process logic from the artifact instantiation logic, which significantly improves the scalability of the platform. Starting from the enriched BPMN process model obtained in the first step, the criteria to map real-world objects to the artifacts can be applied in an automated way. To do so, the following rules are applied:

- Each data association between a BPMN start event and a data object is translated to a mapping criterion. The criterion states that, whenever the event is detected, the artifact represented by the data object is bound to the object identified in the payload of the event. Should the artifact be already bound to a different object, the new binding would replace the existing one. For instance, when the event *TU-HQ started* occurs, Truck is bound to the physical truck whose license plate is specified in the payload of *TU-HQ started*.
- Each data association between a data object and a BPMN end event is translated into a mapping criterion. The criterion states that, whenever the event is detected and the artifact represented by the data object is bound to an object, it becomes unbound. If the artifact is already unbound, no action is taken. For instance, when the event *TU-HQ ended* occurs, no truck is bound to Truck.

Example. Figure 4 shows the artifact-to-object mapping criteria derived from the BPMN process model of Fig. 2. Because the Container artifact interacts with the whole process, the binding is expected to occur when the process starts, and the unbinding to occur once the process finishes. Therefore, to bind a physical container to Container, event *Shipment started* should occur. Once *Shipment started*

```
<Mapping>
  <Artifact name="Container">
    <BindingEvent id="shipment_started"/><UnbindingEvent id="shipment_ended"/>
  </Artifact>
  <Artifact name="Truck">
    <BindingEvent id="M-TU_started"/><UnbindingEvent id="M-TU_ended"/>
    <BindingEvent id="TU-HQ_started"/><UnbindingEvent id="TU-HQ_ended"/>
    <BindingEvent id="HQ-TG_started"/><UnbindingEvent id="HQ-TG_ended"/>
    <BindingEvent id="TG-C_started"/><UnbindingEvent id="TG-C_ended"/>
  </Artifact>
</Mapping>
```

Fig. 4. Artifact-to-object mapping criteria.

is detected, Container is bound to the container whose unique identifier (e.g., its serial number) is equal to the one specified in the payload of *Shipment started*. To unbind Container, *shipment ended* should occur. The Truck artifact, on the other hand, interacts when each subprocess is running. Therefore, to bind a physical truck to Truck, any of the events *M-TU started*, *TU-HQ started*, *HQ-TG started*, or *TG-C started* should occur. Similarly, to unbind Truck, *M-TU ended*, *TU-HQ ended*, *HQ-TG ended*, or *TG-C ended* should occur.

4 Architecture and Implementation

Figure 5 shows the architecture of the monitoring service we developed to support inter-organizational processes. To completely automate the monitoring, we assume that the real-world objects embodying the artifacts can autonomously infer their state and submit such an information to the service. This is a feasible assumption in the context of a Wireless Sensor Network (WSN) [1] or the Internet of Things (IoT) [2], where environmental data can be collected by the objects, which can then infer their own state.

To allow the objects to communicate with the service, a *Message Queue Telemetry Transport (MQTT) Broker* is used. MQTT[1] is a queue-based publish/subscribe protocol, which is especially suited for applications where computing power and bandwidth are constrained. The MQTT Broker contains as many topics (i.e., queues) as the objects that can participate to the process. Each of these topics adheres to the following naming convention: `/{artifact_type}/{object_id}`, where `artifact_type` is the artifact represented by the object (e.g., a truck), and `object_id` is the unique identifier of the object (e.g., the license plate of the truck). Whenever the object changes its state, it publishes the updated state on its own topic. The MQTT Broker also contains as many topics as the process instances that are currently being carried out. Each of these topics adheres to the following naming convention:

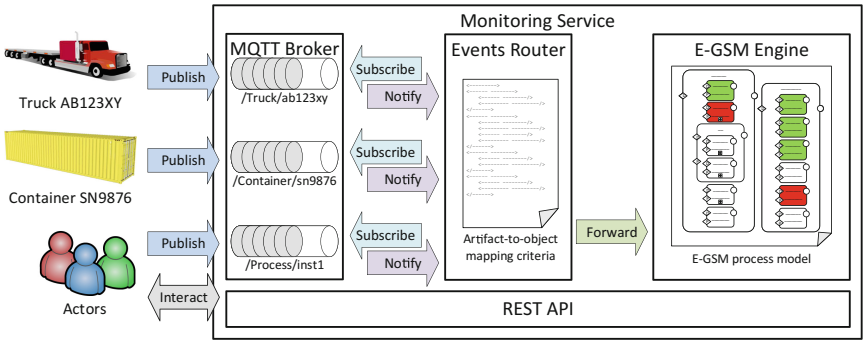

Fig. 5. Architecture of the monitoring service.

[1] http://mqtt.org/.

/{process_name}/{instance_id}, where process_name is the name of process model to be monitored (i.e., the shipment from M to C, henceforth MtoCProcess), and instance_id is the unique identifier of the process instance (i.e., the actual execution of the process) that is being run. These topics are used by the organizations to send events related to the running processes, but not related to the state of the artifacts (i.e., when a subprocess starts or ends).

The *E-GSM Engine*[2] is the component responsible for monitoring the execution of each process instance. This component takes as input the E-GSM models produced according to Sect. 3.2. Whenever a new execution of the process starts, the E-GSM Engine creates a new model instance, whose identifier instance_id is the same as the one of the running process instance. For each model instance, the E-GSM Engine *(i)* keeps track of which activities are ongoing, *(ii)* detects whether they follow the execution flow defined in the model and, if not, *(iii)* marks them as not compliant.

To support late binding and unbinding among objects and artifacts referenced by the process, the *Events Router* component is introduced.[3] By receiving as input the artifact-to-object mapping criteria produced according to Sect. 3.3, the Events Router forwards to each E-GSM model instance only the events produced by the objects that effectively take part in that process execution. Note that, by keeping the binding logic separate from the process logic, the E-GSM instance receives only events coming from those objects that are bound to the running processes. This way, the scalability of the E-GSM engine is affected only by *(i)* the number of processes being run, and *(ii)* the number of objects interacting with those processes, which is way lower than the total number of objects under observation. To do so, the Events Router subscribes to all the /{process_name}/{instance_id} topics (e.g., /MtoCProcess/inst1). Whenever a new event is published (e.g., process_started), the Events Router checks if a mapping criterion is defined for that event. If no mapping criterion exists, the Events Router forwards the event to the E-GSM instance whose identifier is instance_id (e.g., inst1). If a binding criterion exists, the Events Router subscribes to topic /{artifact_type}/{object_id}, where object_id is the object specified in the payload of the event (e.g., /Container/sn9876), and associates to that topic the instance_id (e.g., inst1). From that point on, whenever a new change of state is published in /{artifact_type}/{object_id}, the Events Router forwards it to the E-GSM model instance whose identifier is instance_id. For example, if the truck having license plate AB123XY publishes on /Truck/AB123XY that its state changed to [heathrow,moving], the Events Router will notify that Truck is in [heathrow,moving], together with the raising of $Truck^{l}$ and $Truck^{e}$ events, to the E-GSM instance inst1. If an unbinding criterion exists, the Events Router unsubscribes to topic /{artifact_type}/{object_id}, where object_id is the object specified in the payload of the event.

Finally, the *Representational State Transfer (REST)* [22] *API* offers an interface for the organizations and the service providers to interact with the

[2] Source code at https://bitbucket.org/polimiisgroup/egsmengine.
[3] Source code at https://bitbucket.org/polimiisgroup/eventsrouter.

monitoring service. It allows *(i)* the E-GSM Engine to be provided with the E-GSM model, *(ii)* the Events Router to be instructed with the artifact-to-object mapping criteria, and *(iii)* the organizations and the service providers to determine if the processes are correctly executed. In addition to that, it is responsible for the management of the communication channels between the organizations and service providers, and the monitoring instances: Whenever a new process execution takes place, the REST API instructs the MQTT Broker to create a new `/{process_name}/{instance_id}` topic. Then, the REST API instructs *(i)* the Events Router to listen to that topic for evaluating the mapping criteria, and *(ii)* the E-GSM Engine to create a new model instance whose identifier is the same as `instance_id`. Finally, it forwards the `instance_id` to the involved service providers, to specify the topic on which they should publish the events related to the running process. For instance, when a new shipment from M to C takes place, a new instance id (e.g., `inst1`) is defined, the MQTT topic `/MtoCProcess/inst1` is created, a new E-GSM instance is run, and the notification that `inst1` is up is sent to all involved parties. The organizations can then use `/MtoCProcess/inst1` to send events concerning that shipment.

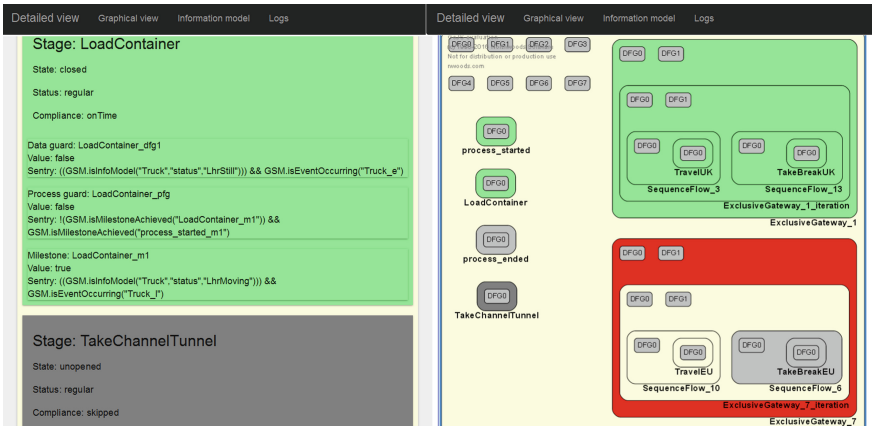

Fig. 6. Screenshot of our service showing a non compliant execution of the TU-HQ leg.

Figure 6 shows a screenshot of the monitoring service displaying a non-compliant execution of the TU-HQ leg. In this case, the truck took a ferry instead of the Channel tunnel. Therefore, our service marks stage `TakeChannelTunnel` as *skipped* (dark gray). Since `TU-HQStarted`, `LoadContainer`, `TravelInUK` and `TakeBreakInUK` were executed in compliance with the control flow, they are marked as *on track* (green). Since the truck has not yet taken a break in the European continent, and the end event has not yet been received, `TakeBreakInEU` and `TU-HQEnded` are not executed yet (light gray). As the truck is traveling in the European continent, stage `TravelInEU` is still being executed (yellow). Note that, although a compliance violation occurred, the monitoring is still running.

5 Validation

To demonstrate the applicability and efficacy of our approach on a real-world case, we have conducted an experiment with truck shipments data provided by a European logistics company. [4] This provided material consisted of *(i)* a dataset with the registered positions and speed of trucks involved in the shipments, captured by on-board AIS/GPS systems and henceforth indicated as *GPS log*, and *(ii)* a dataset indicating the shipments' activities start and completion times, manually triggered by the truck drivers and hereinafter denoted as *activity log*. We replayed the GPS log within our platform and checked whether the start and completion events detected by our platform matched with the manually inserted information in the activity log. This way, we could compare our fully-automated approach with a traditional one relying on human intervention. We focused on routes connecting the premises in Amsterdam (AMS) to four other major European airports, namely the London Heathrow airport (LHR), Brussels (BRU), Paris Charles de Gaulle (CDG), and Frankfurt (FRA). For every route, we considered both inbound and outbound routes from/to Amsterdam.

The GPS log and the activity log contained 19966 and 815 entries, respectively, distributed over 77 shipments. The reported shipments took on average 533 min, ranging from less than 3 to more than 27 hours. By analyzing the activity log, we built a BPMN process for the routes, structured similarly to the legs described in Sect. 2. We identified the possible discrete states that each truck can assume through the inspection of the GPS log. Then, we followed the approach described in Sect. 3: First, we enriched each BPMN model with artifacts representing the truck and its states. Then, we generated the E-GSM models and the artifact-to-object mapping criteria. This output was then used to instruct the monitoring platform on which processes to monitor. After that, we used the WSO2 Complex Event Processing platform[5] to replay the GPS log, let our system detect when the truck changed state, and forward such changes to the monitoring service. Finally, we compared the results of the monitoring platform with the activity log. Table 1 shows the results of our experiment.

The monitoring service was able to correctly determine the actual execution of a process for 93.13% of the total instances. For the remaining 6.87%, the issues lay in the determination of when activity Load container was executed. For example, during one shipment of the BRU-AMS route, Load container was not identified as completed, even though it was. This has to be imputed to the limited information available to determine the state of trucks: Our system had only access to their speed and position, thus anomalous slow progressions due to congestions at the logistic platform and along the road caused the misinterpretation of their state.

Moreover, the monitoring service detected activities to be started or ended more often than what had been notified by the truck drivers. The matching

[4] The (anonymized) dataset is available at http://purl.org/polimi/martifact/logisticsds-anon (password: GM-CDC-JM-dataset).

[5] See http://wso2.com/products/complex-event-processor/.

Table 1. Results of the validation.

Shipment	AMS-LHR	LHR-AMS	AMS-BRU	BRU-AMS	AMS-CDG	CDG-AMS	AMS-FRA	FRA-AMS	Global
Instances	12	15	9	11	8	10	4	8	77
Median duration [min]	806.28	720.05	306.67	256.30	813.48	483.69	481.32	396.30	533.01
Min. duration [min]	338.47	138.02	153.00	159.62	387.57	353.00	396.10	279.32	138.02
Max. duration [min]	1328.56	1622.03	519.12	388.30	1583.52	723.25	567.47	357.32	1622.03
Correctness [%]	91.67%	100.00%	100.00%	90.91%	100.00%	100.00%	75.00%	87.50%	93.13%
Completeness [%]	58.33%	53.33%	77.78%	90.91%	87.50%	60.00%	100.00%	62.50%	73.79%
Median detection delay [min]	2.73	−0.50	5.33	1.09	14.79	0.80	7.10	2.44	4.22
Median absolute d. delay [min]	12.53	4.57	7.10	5.17	16.57	4.18	8.87	4.88	7.98

cases amounted to 73.79%. Whether the missing entries in the activity log were due to an omission of the driver, or rather due to a wrong detection of the system, is debatable and needs further investigation. However, e.g., whenever the monitoring service notified that activity Travel in EU was ended, and no notification was sent by the truck driver, we inspected the GPS log and noticed that the truck had reached Europe and its speed had amounted to zero for more than a quarter of an hour, which suggests the first hypothesis to be more likely.

To assess the time gain for the detection of the status changes in the process, we computed the delay between when each activity was started or ended, as reported by the manual entries of the activity log, and when the monitoring platform detected it, based on the GPS log. We will henceforth name such time difference as *detection delay*. On average, the median of the detection delays amounted to 4.22 min (7.98 considering the absolute values of the delays), which is negligible for processes that last on average 533 min.

6 Related Work

In this section we briefly report on related work about *(i)* the monitoring of business processes by their interaction with physical objects, and *(ii)* techniques to coordinate inter-organizational processes.

In [14], BPMN data objects are adopted to model information on the artifacts manipulated by the activities composing a process. With respect to our work, [14] expects information on the artifacts to be stored in a relational database. Also, binding mechanisms are implemented as an extension of the BPMN syntax, while our work relies solely on BPMN 2.0 OMG standard constructs. [18] proposes a platform to monitor a process based on its interactions with

real-world objects and human operators. Additionally, binding relationships are automatically inferred by observing the execution of the process. However, information on when activities are performed must be explicitly sent to the platform. Also, only the occupation of objects and operators (i.e., if the operator is busy or idle) is taken into consideration. [7] focuses on the process execution monitoring based on physical objects' data. To do so, BPMN constructs are extended to define which events produced by a Complex Event Processing (CEP) determine their activation and termination. Similarly, [5,8] propose to annotate activities with constraints on attributes that are monitored when the process is executed. This way, it is possible to report if an activity is not executed as expected as soon as a violation occurs. [9] applies that approach to detect anomalies and diversions in the context of air-freight cargo transportation. [17], on the other hand, relies on artifacts and their lifecycle to monitor all the parameters relevant for the execution of a process. This way, Key Performance Indicators (KPIs) on the overall execution of the process and each single activities are derived. None of these solutions deal with the detection of deviations in the process execution flow. Concerning the generation of GSM models from activity-centric languages, different approaches have been proposed by [11,16]. However, these approaches treat the execution flow in a prescriptive way. Our solution, which extends [4], treats instead execution flow in a descriptive way, thus allowing more flexibility, and uses information on the artifacts to derive guards and milestones.

Traditionally, to monitor process portions carried out by service providers, commitments have been used. Commitments are formal contracts that specify how the interactions between the organization and the service provider should be performed [23]. However, they are mainly focused on the outcome of the outsourced process portion carried out by the service provider, rather than on the activities composing the process. Our work, on the other hand, is better suited whenever the process must strictly adhere to the model, or when a detailed log on how the process was performed is needed. [19], on the other hand, proposes a GSM-based collaboration hub to coordinate logistics processes at the activity level. The hub also adopts GSM to keep track of the execution of the process. However, it relies on explicit notifications to determine when activities are executed. [13] overcomes this limitation by adopting the IoT paradigm: they take advantage of Guards and Milestones to identify when Stages are being executed by predicating on sensor data coming from smart objects. However, the GSM model is expected to be modeled from scratch. Also, both solutions lack mechanisms to detect deviations in the execution of the process with respect to its model.

7 Conclusions and Future Work

This paper presented a monitoring service based on E-GSM to monitor the execution of inter-organizational processes based on the status of the artifacts being manipulated. The paper has also shown how a standard BPMN process model can be used to automatically produce all the information to drive the

monitoring service. Finally, mechanisms to dynamically bind and unbind real-world objects to a process execution were presented.

A limitation of this service is the support for only one-to-one mappings among real-world objects and artifacts. Therefore, we plan to also support one-to-many and many-to-many mappings to support batch processes [21]. Furthermore, we will introduce tool support to check the soundness of the annotated BPMN process model (i.e., if changes in the states of the artifacts during a compliant execution do not contradict the control flow). To improve the accuracy of the automatic artifact state-change determination, it is in our plans to integrate machine-learning techniques such as automated discriminative classifiers, as proposed in [6,8,9]. Additionally, we are going to distribute the monitoring service onto the real-world objects impersonating the artifacts, so as to completely take advantage of the IoT paradigm. An extension of this service to monitor processes involving non-tangible objects (e.g., invoices or purchase orders) is also planned.

Acknowledgments. This work has been partially funded by the Italian Project ITS Italy 2020 under the Technological National Clusters program.

References

1. Akyildiz, I., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: a survey. Comput. Netw. **38**(4), 393–422 (2002)
2. Atzori, L., Iera, A., Morabito, G.: The internet of things: a survey. Comput. Netw. **54**(15), 2787–2805 (2010)
3. Baresi, L., Meroni, G., Plebani, P.: A GSM-based approach for monitoring cross-organization business processes using smart objects. In: Reichert, M., Reijers, H.A. (eds.) BPM 2015. LNBIP, vol. 256, pp. 389–400. Springer, Cham (2016). doi:10.1007/978-3-319-42887-1_32
4. Baresi, L., Meroni, G., Plebani, P.: Using the guard-stage-milestone notation for monitoring BPMN-based processes. In: Schmidt, R., Guédria, W., Bider, I., Guerreiro, S. (eds.) BPMDS/EMMSAD-2016. LNBIP, vol. 248, pp. 18–33. Springer, Cham (2016). doi:10.1007/978-3-319-39429-9_2
5. Baumgraß, A., Botezatu, M., Di Ciccio, C., Dijkman, R., Grefen, P., Hewelt, M., Mendling, J., Meyer, A., Pourmirza, S., Völzer, H.: Towards a methodology for the engineering of event-driven process applications. In: Reichert, M., Reijers, H.A. (eds.) BPM 2015. LNBIP, vol. 256, pp. 501–514. Springer, Cham (2016). doi:10.1007/978-3-319-42887-1_40
6. Baumgrass, A., Cabanillas, C., Di Ciccio, C.: A conceptual architecture for an event-based information aggregation engine in smart logistics. In: EMISA, pp. 109–123. GI (2015)
7. Baumgrass, A., Herzberg, N., Meyer, A., Weske, M.: BPMN extension for business process monitoring. In: EMISA 2014, pp. 85–98. GI (2014)
8. Cabanillas, C., Di Ciccio, C., Mendling, J., Baumgrass, A.: Predictive task monitoring for business processes. In: Sadiq, S., Soffer, P., Völzer, H. (eds.) BPM 2014. LNCS, vol. 8659, pp. 424–432. Springer, Cham (2014). doi:10.1007/978-3-319-10172-9_31
9. Di Ciccio, C., van der Aa, H., Cabanillas, C., Mendling, J., Prescher, J.: Detecting flight trajectory anomalies and predicting diversions in freight transportation. Decis. Support Syst. **88**, 1–17 (2016)

10. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer, Heidelberg (2013)
11. Eshuis, R., Van Gorp, P.: Synthesizing data-centric models from business process models. Computing **98**(4), 1–29 (2015)
12. Gilley, K.M., Rasheed, A.: Making more by doing less: an analysis of outsourcing and its effects on firm performance. J. Manage. **26**(4), 763–790 (2000)
13. Gnimpieba, Z.D.R., Nait-Sidi-Moh, A., Durand, D., Fortin, J.: Using internet of things technologies for a collaborative supply chain: application to tracking of pallets and containers. Procedia Comput. Sci. **56**, 550–557 (2015)
14. Herzberg, N., Meyer, A., Weske, M.: Improving business process intelligence by observing object state transitions. Data Knowl. Eng. **98**, 144–164 (2015)
15. Hull, R., Damaggio, E., Fournier, F., Gupta, M., Heath, F.T., Hobson, S., Linehan, M., Maradugu, S., Nigam, A., Sukaviriya, P., Vaculin, R.: Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In: Bravetti, M., Bultan, T. (eds.) WS-FM 2010. LNCS, vol. 6551, pp. 1–24. Springer, Heidelberg (2011). doi:10.1007/978-3-642-19589-1_1
16. Köpke, J., Su, J.: Towards quality-aware translations of activity-centric processes to guard stage milestone. In: La Rosa, M., Loos, P., Pastor, O. (eds.) BPM 2016. LNCS, vol. 9850, pp. 308–325. Springer, Cham (2016). doi:10.1007/978-3-319-45348-4_18
17. Liu, R., Vaculín, R., Shan, Z., Nigam, A., Wu, F.: Business artifact-centric modeling for real-time performance monitoring. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 265–280. Springer, Heidelberg (2011). doi:10.1007/978-3-642-23059-2_21
18. Maamar, Z., Faci, N., Sellami, M., Boukadi, K., Yahya, F., Barnawi, A., Sakr, S.: On business process monitoring using cross-flow coordination. Serv. Oriented Comput. Appl. **11**(2), 203–215 (2017)
19. Meijler, T.D., Stollberg, M., Winkler, M., Erler, K.: Coordinating variable collaboration processes in logistics. In: MITIP 2011 (2011)
20. Meroni, G., Di Ciccio, C., Mendling, J.: Artifact-driven process monitoring: dynamically binding real-world objects to running processes. In: CAiSE 2017 Forum, pp. 105–112 (2017). CEUR-WS.org
21. Pufahl, L., Weske, M.: Batch processing across multiple business processes based on object life cycles. In: Abramowicz, W., Alt, R., Franczyk, B. (eds.) BIS 2016. LNBIP, vol. 255, pp. 195–208. Springer, Cham (2016). doi:10.1007/978-3-319-39426-8_16
22. Richardson, L., Ruby, S.: RESTful Web Services - Web Services for the Real World. O'Reilly, Sebastopol (2007)
23. 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)