Introduction

- **Business process**: “chain of events, activities and decisions” [A], “a set of activities that are performed in coordination in an organizational and technical environment” [B]

- **Business process monitoring**: tools and techniques to determine:
  - If activities are correctly executed
  - If dependencies among activities are respected

- **Artifact-driven process monitoring**: a novel technique for business process monitoring, that allows to:
  - Autonomously collect information
  - Determine violations at runtime
  - Without human intervention

Agenda

• Motivations
• Idea: artifact-driven process monitoring
• Contributions:
  – E-GSM modeling language
  – Method to configure smart objects
  – Monitorability assessment and improvement
  – SMARTifact monitoring platform
• Dissemination
• Validation
• Conclusion & future work
Context

- Many intra-organizational processes are becoming multi-party:
  - Portions of a process are outsourced to external organizations
  - Companies interact with goods without owning them
- Organizations are interested in monitoring the execution of multi-party processes as a whole
  - No guarantee that outsourced activities are performed as agreed
  - No guarantee that goods given to other companies are manipulated as agreed
Motivating example
Motivating example

Manufacturer

- Manufacturer portion started
- Fill in container
- Truck reached manufacturer
- Attach container
- Container attached to truck
- Manufacturer portion ended

Carrier

- Carrier portion started
- Drive to manufacturer
- Truck reached manufacturer
- Container attached to truck
- Travel on highway
- Take a break
- Midnight
- Container overheated
- Process failed
- Destination reached
- Truck reached inland terminal
- Container delivered
- Carrier portion ended

Inland terminal

- Terminal portion started
- Truck reached inland terminal
- Inspect goods
- Detach container
- Container delivered
- Terminal portion ended
Problem statement

- Monitoring multi-party processes is challenging
  - The process cannot (always) be interrupted
  - Human operators may not provide feedback
Idea: artifact-driven process monitoring
Idea: artifact-driven process monitoring

- Goods participate in multi-party processes
  - Goods belong to a specific organization
  - Goods have visibility on activities
  - The conditions of the goods can be altered by organizations
- Objects participating in a process are named **artifacts**
- Goods can be seen as artifacts
  - For our purposes, goods = artifacts
- Idea: Artifact-driven process monitoring [1]
  - Monitoring is directly performed on the artifacts
  - The artifact “knows” when its conditions change
  - The artifact “knows” when activities are executed

Objectives

- Exploit the Internet of Things to monitor processes
- Make objects aware of the process
- Perform monitoring transparently and autonomously
Contributions

- Extended-GSM (E-GSM), a declarative language to autonomously monitor business processes
- A method to configure smart objects for artifact-driven monitoring
- A technique to formalize, assess and improve the monitorability of a process
- SMARTifact, an artifact-driven monitoring platform prototype
Contributions

Extended-GSM modeling language
Limitations of activity-centric languages

- Activity-centric languages are unsuited for artifact-driven process monitoring:
  - Execution order must strictly adhere to the process definition
  - An orchestrator must explicitly starts or ends activities
- Artifact-centric languages overcome these limitations [2]
  - Only what is explicitly stated in the model is constrained
  - Everything else is allowed
  - Guard-Stage-Milestone is a good starting point

Extended-GSM overview

- GSM provides the following constructs:
  - **Data Flow Guards** to determine task activation
  - **Milestones** to determine task termination
- E-GSM adds these additional constructs:
  - **Process Flow Guards** to define the expected process flow
  - **Fault Loggers** to determine if a task is unsuccessfully executed
E-GSM allows to monitor processes with respect to three orthogonal perspectives:

- **Execution status:**
  - unopened
  - opened
  - closed

- **Execution outcome:**
  - regular
  - faulty

- **Execution compliance:**
  - on time
  - out of order
  - skipped
E-GSM allows to monitor processes with respect to three orthogonal perspectives:

- **Execution status:**
  - unopened
  - opened
  - closed

- **Execution outcome:**
  - regular
  - faulty

- **Execution compliance:**
  - on time
  - out of order
  - skipped
E-GSM allows to monitor processes with respect to three orthogonal perspectives:

- **Execution status:**
  - unopened
  - opened
  - closed

- **Execution outcome:**
  - regular
  - faulty

- **Execution compliance:**
  - on time
  - out of order
  - skipped
E-GSM allows to monitor processes with respect to three orthogonal perspectives:

- **Execution status:**
  - unopened
  - opened
  - closed

- **Execution outcome:**
  - regular
  - faulty

- **Execution compliance:**
  - on time
  - out of order
  - skipped
Contributions

Method for configuring smart objects
Configuring smart objects for artifact-driven monitoring

- The adoption of E-GSM is not straightforward
  - Artifact-centric languages are difficult to model
  - Processes may already be modeled in BPMN
  - Modelers don’t want to do the same task twice
- Multiple Smart Objects required to monitor the process
- Smart Objects may participate in a portion of the process
  - Information exchanged before/after that portion is useless
- We propose a method to easily configure smart objects [3] [4] [5]

Configuring smart objects for artifact-driven monitoring

- **Input**: BPMN collaboration diagram
- **Output**:
  - rules to dynamically bind and unbind smart objects
  - E-GSM models to monitor:
    - When activities composing the process are performed
    - If the conditions of the smart object evolve as expected

![Diagram showing the process flow](image)
Back to the motivating example

Manufacturer

Carrier

Inland terminal

Manufacturer portion started

Fill in container

Attach container

Drive to inland terminal

Carrier portion started

Drive to manufacturer

Truck reached manufacturer

Container attached to truck

Travel on highway

Take a break

Destination reached

Truck reached Inland terminal

Container delivered

Carrier portion ended

Inspect goods

Detach container

Container delivered

Truck reached inland terminal

Terminal portion ended

Container overheated

Process failed

Midnight
Step 1 – Enrich BPMN collaboration diagram

Manufacturer portion started

Manufacturer

Fill in container

Truck reached manufacturer

Attach container

Container attached to truck

Manufacturer portion ended

Carrier portion started

Drive to manufacturer

Truck reached manufacturer

Carrier portion ended

Inland terminal portion started

Inspect goods

Detach container

Container delivered

Inland terminal portion ended

Carrier portion

Drill to inland terminal

Truck reached manufacturer

Container attached to truck

Travel on highway

Take a break

Midnight

Destination reached

Truck reached Inland terminal

Container delivered

Carrier portion ended

Manufacturer

[garage,moving]

[opened,unhooked]

[closed,unhooked]

[opened,hooked]

[closed,hooked]

[overheated]

[highway,moving]

[highway,still]

[inlandterminal,moving]

[inlandterminal,still]
Step 1 – Enrich BPMN collaboration diagram

Manufacturer
- Manufacturer portion started
- Drive to inland terminal
- Fill in container
- Container attached to truck
- Container delivered

Carrier
- Carrier portion started
- Drive to manufacturer
- Truck reached manufacturer
- Container delivered

Inland terminal
- Terminal portion started
- Inspect goods
- Container delivered
- Container [opened,unhooked]
- Container [closed,unhooked]

Process failed
- Container overheated
- Container [overheated]

© Giovanni Meroni 2018

POLITECNICO MILANO 1863
Step 2 – Derive BPMN process view

© Giovanni Meroni 2018
Step 3a – Derive the E-GSM process model

Drive to inland terminal - BPMN

Drive To Inland Terminal – E-GSM
Step 3b – Derive E-GSM artifact lifecycle model

Container

DDFG1: if (container[o,u] or container[c,u] or container[c,h] or container[o,h] or container[t])

PFG1: false
Step 3c – Derive mapping criteria

```
<LocalArtifact name="Container"/>
<Mapping>
  <Artifact name="Truck">
    <BindingEvent id="Carrier_portion_started"/>
    <UnbindingEvent id="Carrier_portion_ended"/>
    <UnbindingEvent id="Process_failed"/>
  </Artifact>
</Mapping>
```
Contributions

Monitorability assessment and improvement
Monitorability of a process

• Not all smart objects are suited to monitor a process
• The **monitorability** of a process indicates how many activities in a process can be monitored by smart objects [6]
• The capabilities of the Smart Objects affect monitorability
  – The execution of activities is determined by the state of the smart objects
  – The state of a smart object is inferred from its physical properties
  – The physical properties of a smart object are measured by sensors

We exploit ontologies to formalize:
- Sensor data provided by smart objects
- Rules to derive the state of an artifact from sensor data

We then query the ontologies to:
- Compute the monitorability of activities
- Derive the monitorability of the process
- Suggest modifications to improve monitorability:
  - Which sensors should be added to smart objects
  - Which rules can be easily altered to exploit other sensor data
Contributions

SMARTifact monitoring platform prototype
SMARTifact – An artifact-driven monitoring platform

SMARTifact – An artifact-driven monitoring platform

Stage: LoadContainer
State: closed
Status: regular
Compliance: onTime

Data guard: LoadContainer_dfg1
Value: false
Sentry: ((GSM.IsInfoModel("Truck","status","LhrStill"))) && GSM.IsEventOccurring("Truck_e")

Process guard: LoadContainer_pfg
Value: false
Sentry: !((GSM.IsMilestoneAchieved("LoadContainer_m1")) && GSM.IsMilestoneAchieved("process_started_m1")

Milestone: LoadContainer_m1
Value: true
Sentry: !((GSM.IsInfoModel("Truck","status","LhrMoving"))) && GSM.IsEventOccurring("Truck_l")

Stage: TakeChannelTunnel
State: unopened
Status: regular
Compliance: skipped
Validation
Validating SMARTifact – Simulated environment

- Eight shipment processes provided by a large European logistics company [8]
- Two datasets related to 77 shipments
  - Dataset 1: position and speed of trucks (19966 entries)
  - Dataset 2: activation and termination of activities in shipment processes, manually notified by truck drivers (815 entries)
- Dataset 1 was replayed on SMARTifact
- The results of the monitoring were compared with Dataset 2
  - Over 93% of the shipments were correctly monitored
  - SMARTifact detected more activities than manual notifications
  - Detection delay was less than 5 minutes w.r.t. 533 minutes uptime

Validating SMARTifact – Field evaluation

- Equipped my briefcase with an Intel Galileo SBC and a GPS receiver
- Monitored for 4 months the process of going to work and back home
- Almost 95% of the process instances were correctly identified
- The median detection delay was less than 2 minutes, while the processes lasted on average 102 minutes
Dissemination
Publications

Conclusion & future work
Conclusion

• Artifact-driven process monitoring can effectively monitor inter-organizational processes
  – The IoT makes physical objects smart
  – Operators no longer have to send notifications
  – Violations can be autonomously detected
  – Monitoring is continuous, without human intervention
  – The SMARTifact platform proved the applicability of this approach

• The information required for artifact-driven monitoring can be derived from BPMN models
  – An E-GSM model, to introduce flexibility
  – Criteria to bind and unbind smart objects to running processes
Future Work

- Integrating artifact-driven process monitoring with blockchain to achieve trusted monitoring [9]
- Introducing mechanisms to make monitoring robust when network communications are unreliable [10]
  - Exploring the capabilities of 5G mobile networks
  - Implementing corrective actions to compensate loss and delay
  - Investigating on distributed consistency among smart objects

Thanks for your attention

This PhD has been funded by the Italian Project ITS2020 under the Technological National Clusters program
Backup slides
E-GSM Stage lifecycle

- E-GSM allows to monitor processes with respect to three orthogonal dimensions:
  - Execution status:
    - Unopened
    - Opened
    - Closed
  - Execution outcome:
    - Regular
    - Faulty
  - Execution compliance:
    - OnTime
    - OutOfOrder
    - Skipped

(S.DFG; S' ≠ S) and (Active(S) or S.Mj in S'.PFGk)
S.DFG and not S.PFGk
(S.DFG, or -S.Mj) and not S.PFGk
(S.DFG, or -S.Mj) and S.PFGk
(S.DFG, or -S.Mj) and S.PFGk
S.DFGi
S.DFGi
S.DFGi
S.DFGi
S.DFGi
S.DFGi
S.DFGi
S.DFGi
S.DFGi
S.DFGi
S.DFGij
S.DFGij
S.DFGij
S.DFGij
S.DFGij
S.DFGij
S.DFGij
S.DFGij
S.DFGij
S.DFGij
Basic translation rules

- Each activity is translated into a Stage with one or more DFG and one or more M.
- Each event is translated into a Stage with a DFG and a M capturing the occurrence of the event.
- Each non-interrupting boundary event is translated into a FL.
- Each interrupting boundary event is translated into a FL and a M.

<table>
<thead>
<tr>
<th>BPMN</th>
<th>E-GSM</th>
<th>BPMN</th>
<th>E-GSM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rule 1: Activity</strong></td>
<td></td>
<td><strong>Rule 2: Event</strong></td>
<td></td>
</tr>
<tr>
<td><img src="image1" alt="Diagram" /></td>
<td>DFG1: [A_s]</td>
<td><img src="image2" alt="Diagram" /></td>
<td>DFG1: on e</td>
</tr>
<tr>
<td></td>
<td><img src="image3" alt="Diagram" /></td>
<td>M1: [A_t]</td>
<td>M1: on e</td>
</tr>
<tr>
<td></td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td><strong>Rule 3: Activity with non blocking Boundary Event</strong></td>
<td></td>
<td><strong>Rule 4: Activity with blocking Boundary Event</strong></td>
<td></td>
</tr>
<tr>
<td><img src="image4" alt="Diagram" /></td>
<td>DFG1: [A_s]</td>
<td><img src="image5" alt="Diagram" /></td>
<td>DFG1: [A_s]</td>
</tr>
<tr>
<td></td>
<td><img src="image6" alt="Diagram" /></td>
<td>M1: [A_t]</td>
<td>M1: [A_t]</td>
</tr>
<tr>
<td></td>
<td><img src="image7" alt="Diagram" /></td>
<td>FL1: on e</td>
<td>Me: on e</td>
</tr>
<tr>
<td></td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
</tbody>
</table>
• Identification of process blocks
• Single inbound and single outbound control flows
• Can be nested
• Five process blocks: Sequence, Parallel, Conditional exclusive, Conditional exclusive, Loop
• Each block is translated into an E-GSM stage
• DFG, PFG an M depend on the nature of the block
• Inner blocks become inner stages
Translating the exceptional flow

- Exceptional flow is alternative to normal flow
  - Originates from interrupting boundary events
  - Can go either in the same or in the opposite direction wrt normal flow
  - Must be merged with normal flow with an exclusive merge gateway.

- Exceptional flow runs in parallel with the normal flow
  - Originates from non-interrupting boundary events
  - Must be merged with normal flow with an inclusive merge gateway
Translating the exceptional flow

- **Three exceptional blocks definable**
- **Forward exception handling**
- **Backward exception handling**
- **Non-interrupting exception handling**
- **They behave similarly to Conditional exclusive, Loop and Conditional inclusive blocks, respectively**
Translation by Example
Non-boundary Events

DFG1: on MessageSent
if body:inventory

Inventory

M1: on MessageSent
if body:inventory
Translation by Example
Forward Exception Handling Block

BPMN

E-SEGM

DFG1: 
[ShipToConsumer_s]
D

M1: 
[ShipToConsumer_t]
D

EExc
ShipTo
Consumer
M1: ... 
and not Active(DiscardGoods)
PPFG1: StoreInWarehouse.Me and 
not DiscardGoods.M1 
and not Active(ShipToConsumer)

DFG2: 
[DiscardGoods_s]

M1: 
[DiscardGoods_t]

Discard
Goods
M1: ... 
and not Active(DiscardGoods)

PFG1: (StoreInWarehouse.M1 
or StoreInWarehouse.Me) and 
not EExc.M1

PFG1: not StoreInWarehouse.Me 
and not ShipToConsumer.M1 
and not Active(DiscardGoods)

PFG1: StoreInWarehouse.Me and 
not DiscardGoods.M1 
and not Active(ShipToConsumer)

On NewSensorSample:
If Temperature > 0°C
Goods damaged:
On NewSensorSample:
If Temperature > 20°C
Goods damaged:

© Giovanni Meroni 2018
Monitorability of a process

• Not all smart objects are suited to monitor a process
• The **monitorability** of a process indicates how many activities in a process can be monitored by smart objects
• The capabilities of the Smart Objects affect monitorability
  • The execution of activities is determined by the state of the smart objects
  • The state of a smart object is inferred from its physical properties
  • The physical properties of a smart object are measured by sensors
• We propose an ontology-based approach to:
  • Formalize the capabilities of smart objects
  • Estimate the monitorability
  • Provide suggestions to improve the monitorability
Smart object ontology

• Ontology derived from FIESTA-IoT that captures the capabilities of the smart object

Artifact realizesArtifact Smart Object

Truck realizesArtifact Plate AB123XY
• Ontology derived from FIESTA-IoT that captures the capabilities of the smart object
Smart object ontology

• Ontology derived from FIESTA-IoT that captures the capabilities of the smart object

- Artifact realizes Artifact on Platform
- Smart Object realizes Artifact on Platform
- Sensing Device realizes Artifact on Platform
- GPS Receiver has Quantity Kind
- Position captures
- Quantity Kind
- Speed Instantaneous
- Speedometer
Smart object ontology

• Ontology derived from FIESTA-IoT that captures the capabilities of the smart object
State detection rules ontology

- Ontology derived from Physics Domain ontology (Hachem et al. – MDS 2011) that formalizes how sensor data is used to infer a state.
State detection rules ontology

- Ontology derived from Physics Domain ontology (Hachem et al. – MDS 2011) that formalizes how sensor data is used to infer a state.
State detection rules ontology

• Ontology derived from Physics Domain ontology (Hachem et al. – MDS 2011) that formalizes how sensor data is used to infer a state.
State detection rules ontology

- Ontology derived from Physics Domain ontology (Hachem et al. – MDS 2011) that formalizes how sensor data is used to infer a state.
Ontology derived from Physics Domain ontology (Hachem et al. – MDS 2011) that formalizes how sensor data is used to infer a state.
Process monitorability assessment

• For each couple \(<\text{artifact}, \text{state}>\) in the process model, we need to determine how many smart objects \(SSO\) can infer that state based on their capabilities.

• To do so, for each smart object \(SSO\) that embodies \text{artifact}, the ontologies are queried to determine:
  • If a detection rule to infer \text{state} exists
  • Which parameters are required by that rule
  • If the sensors on the smart object provide the required parameters

• Then, the monitorability of \(<\text{artifact}, \text{state}>\) is computed as:

\[
Mon^{ARS}(\langle\text{artifact}, \text{state}\rangle, I) \rightarrow [0, 1] = \left|SSO\right| / |SSO|
\]
• Determine if truck **AB123XY** can infer `<truck, still>`:
  - ✔ If a detection rule to infer **still** exists
  - □ Which parameters are required by that rule
  - □ If the sensors on **AB123XY** provide the required parameters
• Determine if truck **AB123XY** can infer `<truck, still>`:
  • ✔ If a detection rule to infer **still** exists
  • ✔ Which parameters are required by that rule
  • ☐ If the sensors on **AB123XY** provide the required parameters
Process monitorability assessment

• Determine if truck **AB123XY** can infer `<truck, still>`:
  • ✓ If a detection rule to infer `still` exists
  • ✓ Which parameters are required by that rule
  • ✓ If the sensors on **AB123XY** provide the required parameters
Process monitorability assessment

• Once $Mon^{ARS}$ has been determined for every couple \langle artifact, state \rangle, the monitorability of the activation and the termination of an activity is determined as:

$$Mon^C(A_i.C_i^{start}, I) \rightarrow [0, 1] = \prod_{ARS_{i,j} \in A_i.C_i^{start}} Mon^{ARS}(ARS_{i,j}, I)$$

(1)

$$Mon^C(A_i.C_i^{stop}, I) \rightarrow [0, 1] = \prod_{ARS_{i,k} \in A_i.C_i^{stop}} Mon^{ARS}(ARS_{i,k}, I)$$

(2)

• Then, the monitorability of an activity is:

$$Mon^A(A_i, I) \rightarrow [0, 1] = \frac{1}{2} \cdot \left( Mon^C(A_i.C_i^{start}, I) + Mon^C(A_i.C_i^{stop}, I) \right)$$

• Finally, the monitorability of the process is:

$$Mon^P(P, I) \rightarrow [0, 1] = \sum_{A_i \in P} \frac{Mon^A(A_i, I)}{|A_i \in P|}$$
To improve monitorability, three types of actions are possible:

- Alter the process model to rely on different artifacts or states to determine when activities are executed
- Improve the state detection rules
- Modify the smart objects introducing new sensors

When altering the process model, for each couple `<artifact, state>` that cannot be monitored, the ontologies can suggest:

- Another state `state'` for `artifact` such that:
  \[ Mon^{ARS}_{\text{state'}}(\langle \text{artifact}, \text{state'} \rangle, I) > 0 \]
- Another artifact `artifact` such that:
  \[ Mon^{ARS}_{\text{state'}}(\langle \text{artifact'}, \text{state} \rangle, I) > 0 \]
To improve the state detection rules, the ontologies can detect smart objects that:

- Cannot detect a state just because their sensors use a data format different from the one required by the detection rule
- Cannot detect a state, but provide sensor data that can be used to indirectly derive that state

By introducing a new detection rule similar to the existing one except for the input parameters, these smart objects can detect that state.

- This positively affects the monitorability of the process

For smart objects that cannot provide sensor data to detect that state, either directly or indirectly, the ontologies can suggest which sensors should be introduced
• Truck CD456WX provides the speed in miles per hour.
• Truck CD456WX provides the speed in miles per hour.
• To detect <truck, still>, rule speed2state requires the speed to be expressed in kilometers per hour.
• Truck CD456WX cannot use speed2state, so it cannot detect <truck, still>.
• Truck CD456WX provides the speed in miles per hour
• To detect <truck, still>, rule speed2state requires the speed to be expressed in kilometers per hour
• Truck CD456WX cannot use speed2state, so it cannot detect <truck, still>
• A new rule speed2state’ can be derived from speed2state by converting the speed from miles per hour to kilometers per hour
• With speed2state’, Truck CD456WX can now detect <truck, still>
• Truck EF789UV provides its own position in decimal degrees coordinates
• Truck EF789UV provides its own position in decimal degrees coordinates
• To detect <truck, still>, rule speed2state requires the speed
• Truck EF789UV cannot use speed2state, so it cannot detect <truck, still>
Process monitorability improvement

- Truck EF789UV provides its own position in decimal degrees coordinates
- To detect <truck, still>, rule speed2state requires the speed
- Truck EF789UV cannot use speed2state, so it cannot detect <truck, still>
- However, speed can be derived from the position by using formula coords2speed
Process monitorability improvement

• Truck EF789UV provides its own position in decimal degrees coordinates
• To detect <truck, still>, rule speed2state requires the speed
• Truck EF789UV cannot use speed2state, so it cannot detect <truck, still>
• However, speed can be derived from the position by using formula coords2speed
• A new rule coords2state can be derived by combining speed2state with coords2speed
• With coords2state, Truck CD456WX can now detect <truck, still>