On Handling Business Process Anomalies through Artifact-based Modeling

Luciano Baresi, Giovanni Meroni, and Pierluigi Plebani

Politecnico di Milano — Dipartimento di Elettronica, Informazione e Bioingegneria
Piazza Leonardo da Vinci, 32 - 20133 Milano, Italy
[firstname].[lastname]@polimi.it,

Abstract. Control flow-based process modeling notations, like BPMN, are good at defining the normal execution flow and the management of foreseen exceptions. When unforeseen situations occur, one cannot detect if the execution is still acceptable with respect to the process definition. In contrast, artifact-centric process modeling notations, like the Guard-Stage-Milestone (GSM), are better suited for this kind of scenarios: they define a process in terms of acceptable states and do not enforce any specific execution flow. This improves flexibility, but hampers the clarity of the defined models. The goal of this paper is to show how an extension of GSM, i.e., E-GSM, can be used to detect deviations from the execution path as modeled in BPMN, while keeping the process execution alive.

Keywords: Process Monitoring, Guard-Stage-Milestone, Artifact-centric processes, Handling Process Exceptions

1 Introduction

The cooperation and communication among different organizations are usually defined through business processes that capture the tasks performed by each organization, their order, and the messages exchanged. To design these processes, one usually adopts imperative modeling languages, like the Business Process Modeling Notation (BPMN). These languages ease the definition of the expected execution flow and of the exceptional flows aimed to manage anomalies foreseen at design time. However, while a process is executed, each organization has full control only on the portion of the process that is under its responsibility. This means that the overall, actual execution flow may differ from the one defined originally, and unforeseen exceptions could materialize. Moreover, one organization can know the activities executed by the others only if they properly distribute information about their initiation and termination.

To alleviate these problems, this paper proposes a solution to monitor the execution of distributed control flow-based process models based on artifact-centric ones. We start from a BPMN process, which is easy to conceive, and we transform it into an equivalent model in E-GSM, an extension to the Guard-Stage-Milestone (GSM) artifact-centric modeling notation [5]. The resulting E-GSM model can be shared by the involved parties to allow them to keep track

In: S. España, M. Ivanović, M. Savić (eds.): Proceedings of the CAiSE'16 Forum at the 28th International Conference on Advanced Information Systems Engineering, Ljubljana, Slovenia, 13-17.6.2016, published at http://ceur-ws.org

of the order in which activities are executed and of the status of each activity. Deviations from the "original" execution flow can easily be detected at run-time during the process enactment.

The paper is structured as follows. Section 2 discusses how we extended GSM into E-GSM to enable a data-artifact driven process monitoring solution. Section 3 gives a brief overview of the rules we defined to translate BPMN to E-GSM. Section 4 shows how E-GSM can be used to monitor a real business process in the domain of logistics. After a comparison with relevant work in the literature in Section 5, Section 6 outlines possible future work.

2 E-GSM

The GSM notation is a declarative language that allows one to model artifact-centric processes by defining conditions that determine the activation and termination of activities, called **Stages**, based on events. Starting from the standard GSM notation and our preliminary work [2], we propose E-GSM, an extension where we distinguish between **Data Flow Guards** (that represent the original Guards in GSM) and **Process Flow Guards**, and we add **Fault Loggers**:

- A **Data Flow Guard** is an Event-Condition-Action (ECA) rule. If true, the associated **Stage** is declared opened. A **Milestone** is another ECA rule. If true, the **Stage** is declared closed. A **Milestone** may also have an *invalidator*: a boolean expression that can invalidate the **Milestone** and reopen the **Stage**.
- A **Process Flow Guard** is a boolean expression that predicates on the activation of the **Data Flow Guards** and **Milestones** used to map the expected control flow. The expression is evaluated once one of the **Data Flow Guards** of the associated **Stage** is triggered, and before the **Stage** becomes opened. If the expression is true, the **Stage** complies with the expected execution, otherwise the **Stage** has been activated without respecting the normal flow.
- A **Fault Logger** is an ECA rule. If true, the associated **Stage** is declared as faulty because something went wrong during the execution of the activity. A faulty **Stage** does not imply its termination, as the termination is only determined by **Milestones**.

Figure 1 sketches the lifecycle of an E-GSM **Stage** organized around three main orthogonal execution perspectives: status, outcome, and compliance[1]:

- The *Execution status* captures the status of a **Stage**: *unopened* (**Data Flow Guards** have never been triggered), *opened* (one of the **Data Flow Guards** triggers) or *closed* (when opened, a **Milestone** is achieved).

[1] In this paper we use the the notation introduced in [5], so we write $\mathtt{S.DFG_i}$, $\mathtt{S.PFG_k}$, $\mathtt{S.FL_l}$ to indicate the activation of a Data Flow Guard, Process Flow Guard, or a Fault Logger associated with Stage `S`, $\mathtt{+S.M_j}$ ($\mathtt{-S.M_j}$) to indicate the achievement (invalidation) of a Milestone $\mathtt{M_j}$, $\mathtt{S.M_j}$ to indicate that Stage `S` is closed and a Milestone $\mathtt{M_j}$ is achieved, and `Active(S)` to indicate that Stage `S` is opened.

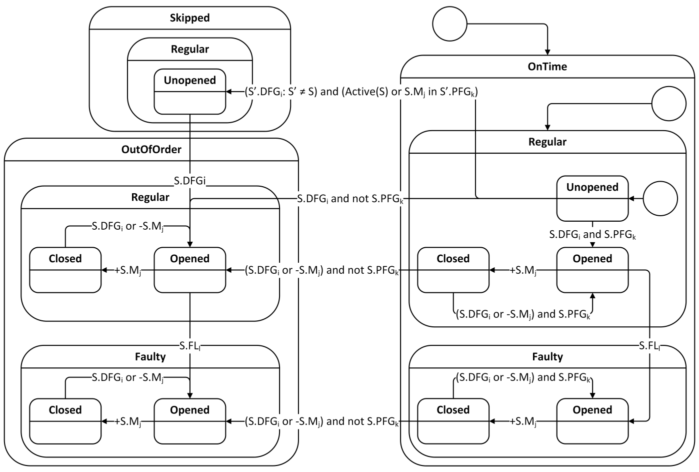
Skipped
Regular
Unopened
(S'.DFG$_i$: S' ≠ S) and (Active(S) or S.M$_j$ in S'.PFG$_k$)
OutOfOrder
S.DFGi
Regular
S.DFG$_i$ and not S.PFG$_k$
S.DFG$_i$ or -S.M$_j$
Closed
+S.M$_j$
Opened
(S.DFG$_i$ or -S.M$_j$) and not S.PFG$_k$
S.FL$_l$
Faulty
S.DFG$_i$ or -S.M$_j$
Closed
+S.M$_j$
Opened
(S.DFG$_i$ or -S.M$_j$) and not S.PFG$_k$
OnTime
Regular
Unopened
S.DFG$_i$ and S.PFG$_k$
Closed
+S.M$_j$
Opened
(S.DFG$_i$ or -S.M$_j$) and S.PFG$_k$
S.FL$_l$
Faulty
(S.DFG$_i$ or -S.M$_j$) and S.PFG$_k$
Closed
+S.M$_j$
Opened

Fig. 1. Lifecycle of an E-GSM Stage `S`.

- The *Execution outcome* captures the situation of a **Stage**, which can be either *regular* (none of its **Fault Loggers** has ever been triggered) or *faulty* (at least one of its **Fault Loggers** has been triggered, $\texttt{A.FL}_\texttt{l}$).
- The *Execution compliance* captures the compliance of each **Stage** with the normal flow. A **Stage** is declared *onTime* by default. It can become *outOfOrder* (according to the normal flow) when one of its **Data Flow Guards** is triggered but none of its **Process Flow Guards** holds, or *skipped* if it does not become active before the Stages that requires its activation.

Summarizing, **Data Flow Guards** drive the change of status. **Fault Loggers** drive the outcome, while **Process Flow Guards** are in charge of the compliance. With respect to Standard GSM, E-GSM interprets reopening a *closed* **Stage** as a new iteration of that process portion. Therefore, once a parent **Stage** is *reopened*, the lifecycle of all its child **Stages** will restart from scratch.

3 E-GSM generation

Starting from a process modeled using BPMN – that we aim to monitor – the generation of the corresponding E-GSM – that feeds the monitoring system – adheres to a set of transformation rules. These rules are applicable to every BPMN process model that complies with a workflow net [1], that is, the process has only one start event and only one end event, and it always terminates (soundness).

Basic transformation rules are shown in Figure 2:

(a) BPMN Activities are translated into E-GSM Stages.
(b) BPMN Start, End or Intermediate Events are translated into E-GSM Stages where their Data Flow Guards and the Milestones are associated to the occurrence of the event.

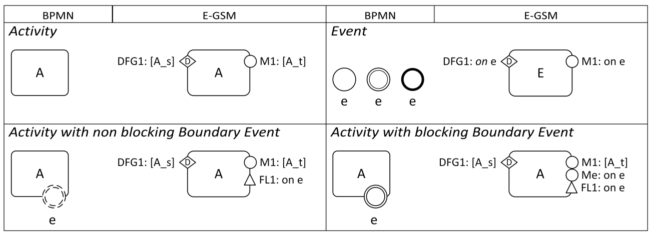

Fig. 2. BPMN to E-GSM transformation rules for basic elements.

(c) BPMN Activities with a non-interrupting Boundary Event attached to them are translated into E-GSM Stages as in case (a), with a Fault Logger having the occurrence of the event as condition.

(d) BPMN Activities `A` with an interrupting Boundary Event attached to them are translated into E-GSM Stages as in case (a), with an additional Milestone and Fault Logger having the occurrence of the event as condition.

These transformation rules allow any well-structured BPMN process model to be translated into E-GSM. A BPMN to E-GSM prototype translator implemented in ATL (ATLAS Transformation Language [6]), has been developed to support the translation activities.[2]

The combination of the above rules for basic elements allows one to translate well-structured business process models. In particular, we focus on five types of blocks, defined starting from the classical control flow patterns [12]: *sequence block*, *parallel block*, *conditional exclusive block*, *conditional inclusive block*, *loop block*. For each of these blocks a transformation rule have been defined and a graphical representation of some of them is reported in Figure 3. In [9], the complete set of these transformation rules is discussed in details.

For instance, a sequence block corresponds to a Stage `Seq` that includes $\mathtt{S_x}$ inner Stages obtained by applying the transformation rules to all the elements (i.e., Activities, Events, inner blocks) that belong to the block. Moreover, in addition to the existing Process Flow Guards, each inner stage has $\mathtt{S_x.PFG1}$ to state that none of its Milestones is achieved, and at least one of the Milestones of the element that directly precedes it (if present) is achieved. This way, inner Stages are expected to be opened only once, and only after their direct predecessor is closed. Finally, `Seq` has a set `Seq.DFG` that includes all $\mathtt{S_x.DFG_i}$, and a Milestone `Seq.M1` that requires that, for all $\mathtt{S_x}$, at least one $\mathtt{S_x.M_j}$ be achieved. This way, `Seq` is opened when at least one of its $\mathtt{S_x}$ is opened too, and – as achieving a Milestone is enough to close a Stage – `Seq` is closed when all $\mathtt{S_x}$ are closed.

4 E-GSM in action

To better clarify how E-GSM models can be used to monitor the execution of complex (distributed) processes, here we concentrate on an example taken from

[2] The tool is publicly available at `https://bitbucket.org/polimiisgroup/bpmn2egsm`.

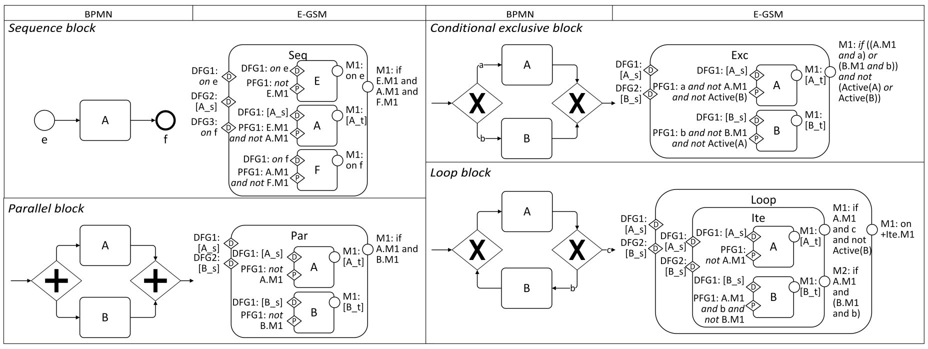
BPMN
E-GSM
Sequence block
Conditional exclusive block
Parallel block
Loop block
Seq
Exc
Par
Loop
Ite
DFG1: on e
PFG1: not E.M1
M1: on e
M1: if E.M1 and A.M1 and F.M1
DFG1: [A_s]
PFG1: E.M1 and not A.M1
M1: [A_t]
DFG1: on f
PFG1: A.M1 and not F.M1
M1: on f
M1: if ((A.M1 and a) or (B.M1 and b)) and not (Active(A) or Active(B))
PFG1: a and not A.M1 and not Active(B)
PFG1: b and not B.M1 and not Active(A)
M1: if A.M1 and B.M1
PFG1: not A.M1
PFG1: not B.M1
M1: if A.M1 and c and not Active(B)
M2: if A.M1 and (B.M1 and b)
M1: on +Ite.M1
PFG1: A.M1 and b and not B.M1

Fig. 3. Excerpt of BPMN to E-GSM transformation rules.

the logistics domain. A manufacturing company M has to ship its goods to one of its customers N and, to do so, it relies on a shipping company S for rail and sea-cargo transportation, and on a shipping company T for truck transportation. The shipping process comprises four main phases: (i) loading goods into a shipping container; (ii) shipping such a container to an intermediate site A by truck; (iii) depending on the day of the week, shipping the goods to either site B by rail, or to site C by sea; (iv) delivering the goods to the customer's site by truck. Furthermore, if part of the goods drops during the loading phase, that activity should stop, all involved actors should be notified of such an accident, and then the loading phase redone. Figure 4 shows the BPMN definition of this process in the upper part, and the derived E-GSM process, which is produced by our translator, in the lower part.

The resulting E-GSM model definition feeds an E-GSM engine that is able to check the evolution of the lifecycle of the **Stages**. We assume that a smart device embeds this engine and travels along with the goods. Using on-board sensors, or via explicit messages sent by an operator, the engine is able to determine when E-GSM elements that decorate each **Stage** are triggered, and to log information on each **Stage** along with the execution status, outcome and compliance perspectives: **Data Flow Guards** allow the parties to realize when activities are started, **Milestones** when they end, **Fault Loggers** if they were not correctly executed, and **Process Flow Guards** if they are compliant with the defined control flow.

Let us suppose that the shipping process is carried out as follows: the process starts on Wednesday, when T carefully loads the goods onto their truck at M's site (when the loading phase is complete, the operator sends a message making `LoadGoods.M1` achieved) and ships them to site A on Thursday. Once the goods arrive at A, `ShipToA.M1` is achieved. Before taking the goods from T, S queries the smart device to verify that the activities `LoadGoods` and `ShipToA` have been correctly performed, and then the goods are exchanged. Being Thursday, S ships the goods to site C by sea. Finally, T delivers the goods to N's site, where N queries the smart device to have information on how the process has been enacted. In this case, since the execution flow has been respected (i.e., every

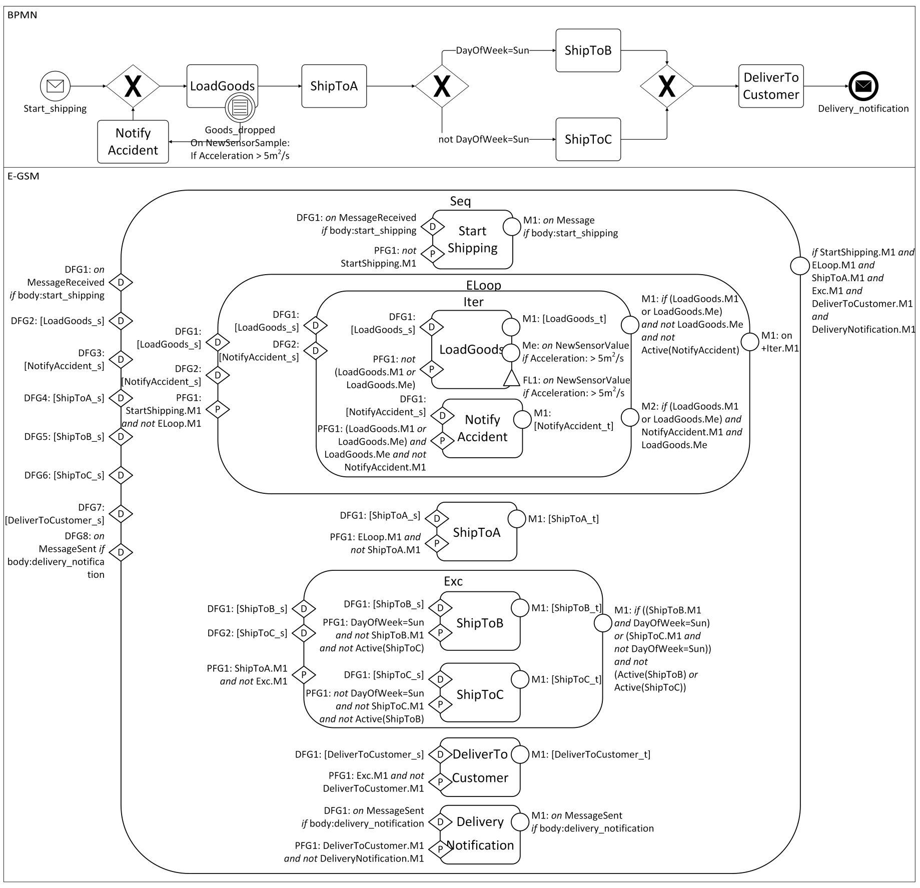

Fig. 4. BPMN and E-GSM models of the example shipping process.

time a Stage was opened, the conditions on its process flow guards have been satisfied), and no Fault Logger has been triggered, N accepts the goods.

Let us now focus on a process execution where the designed control flow is not respected. For instance, during the initial loading, part of the goods is dropped. However, T ignores that accident and the shipping process continues as planned. This cause the verification of the **FaultLogger** `LoadGoods.FL1`, that moves `LoadGoods` to the *faulty* outcome. Once N receives the goods, it discovers that some of them are damaged, and returns them to M. By querying the smart device, M immediately identifies that the goods have been dropped (i.e., the condition on `LoadGoods.FL1` has been satisfied), that `ShipToA` started while `LoadGoods` was still in progress, and that the accident has not been notified (i.e., `NotifyAccident` has not been executed even though `NotifyAccident.PFG1` was satisfied). Thank to these information, M is able to charge T a penalty for the damaged goods, and to impose it to ship the new goods to N for free.

A third example shows how the system can handle unforeseen exceptions. In this case, the shipping to site A takes longer than expected and the goods arrive there on Sunday. Nevertheless, S ships the goods by sea to site C. This causes a significant delay and a waste of resources since the goods are expected to arrive at B and, once their current location is identified, a truck must bring them from site B to C. Again, the information collected by the smart device allows N to identify that `ShipToC` does not comply with the model: `ShipToC` is executed instead of `ShipToB`, and `ShipToC` has been opened even though `ShipToC.PFG1` was not satisfied. Moreover, by knowing that `ShipToC` is currently running, M can blame S as the responsible for this inefficiency. If the smart device is also equipped with a communication interface, it can autonomously send information to all involved parties. This allows T to figure out that `ShipToC` is in progress and be ready to pick up the goods at site C instead of B.

These examples demonstrate how E-GSM can be used to effectively monitor the execution of a distributed process, especially in case of processes where several actors are involved and no central orchestrator is imposed. While an E-GSM engine is not available yet, an extension of the Barcelona GSM engine [4] is currently under development.

5 Related Work

Modeling business processes with declarative languages is generally more difficult than with imperative ones as proved by Pichler et al. [10]. For this reason, and also because of the need for reusing excerpts of preexisting process models, there have been some attempts to develop methods and solutions to translate imperative languages into declarative ones.

Köpke et al. [7] start from a BPMN process model and produce a GSM equivalent that forces the process to behave exactly as defined in the BPMN model. While we have borrowed the Stage nesting, our rules differ significantly from the ones in [7], mainly due to the fact that the purpose of our work is completely different. In particular, we are interested in identifying control flow violations, and not in forcing the process to rigidly follow a given execution flow, which is what is pursued in this work.

Eshuis et al. [3] define a semi-automated approach to produce GSM models starting from UML Activity Diagrams, while Kumaran et al. [8] propose a language-agnostic algorithm to derive the lifecycle of artifacts based on an imperative process model. With respect to our work, which keeps control flow information in the target process model to assess compliance, [3] and [8] use such information to model the interactions among data artifacts.

Popova et al. [11] define a translator from Petri Nets to GSM to transform the outcome of process mining algorithms, which is often represented as a Petri Net, in a language easier to understand by domain experts.

6 Conclusions and Future Work

In this paper we proposed an extension of the Guard-Stage-Milestone (GSM) notation to embed control flow information in the process model definition to enable the translation from BPMN models into equivalent E-GSM ones. The resulting declarative process can drive the process monitoring to check anomalies during the execution of the process. Future work will concentrate on how to verify the soundness of the derived E-GSM processes and the level of equivalence with respect to the original BPMN process. As E-GSM has been studied to drive a flexible monitoring system, tools for distributing and running E-GSM-driven monitoring on smart devices will be proposed.

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

References

1. Van der Aalst, W.M.: Verification of workflow nets. In: Application and Theory of Petri Nets 1997, pp. 407–426. Springer (1997)
2. Baresi, L., Meroni, G., Plebani, P.: A gsm-based approach for monitoring cross-organization business processes using smart objects (2015), accepted for publication
3. Eshuis, R., Van Gorp, P.: Synthesizing data-centric models from business process models. Computing pp. 1–29 (2015)
4. Heath III, F.T., Boaz, D., Gupta, M., Vaculín, R., Sun, Y., Hull, R., Limonad, L.: Barcelona: A design and runtime environment for declarative artifact-centric bpm. In: Service-Oriented Computing, pp. 705–709. Springer (2013)
5. Hull, R., Damaggio, E., Fournier, F., Gupta, M., Heath, Fenno(Terry), I., Hobson, S., Linehan, M., Maradugu, S., Nigam, A., Sukaviriya, P., Vaculin, R.: Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In: Web Services and Formal Methods, Lecture Notes in Computer Science, vol. 6551, pp. 1–24. Springer (2011)
6. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: Atl: A model transformation tool. Science of computer programming 72(1), 31–39 (2008)
7. Köpke, J., Su, J.: Towards ontology guided translation of activity-centric processes to gsm (2015), accepted for publication
8. Kumaran, S., Liu, R., Wu, F.Y.: On the duality of information-centric and activity-centric models of business processes. In: Advanced Information Systems Engineering. pp. 32–47. Springer (2008)
9. Meroni, G., Baresi, L., Plebani, P.: Translating BPMN to E-GSM: specifications and rules. Tech. rep., Politecnico di Milano (2016), `http://hdl.handle.net/11311/976678`
10. Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J., Reijers, H.A.: Imperative versus declarative process modeling languages: An empirical investigation. In: Business Process Management Workshops. pp. 383–394. Springer (2012)
11. Popova, V., Dumas, M.: From petri nets to guard-stage-milestone models. In: Business Process Management Workshops. pp. 340–351. Springer (2013)
12. Russell, N., Hofstede, A.H.M.T., Mulyar, N.: Workflow controlflow patterns: A revised view. Tech. Rep. BPM-06-22, BPM Center Report, BPMcenter.org (2006)