BPMDS 2016

Ljubljana – 13th – 14th June 2016

POLITECNICO DI MILANO





Luciano Baresi, Giovanni Meroni and Pierluigi Plebani

USING THE GUARD-STAGE-MILESTONE NOTATION FOR MONITORING BPMN-BASED PROCESSES

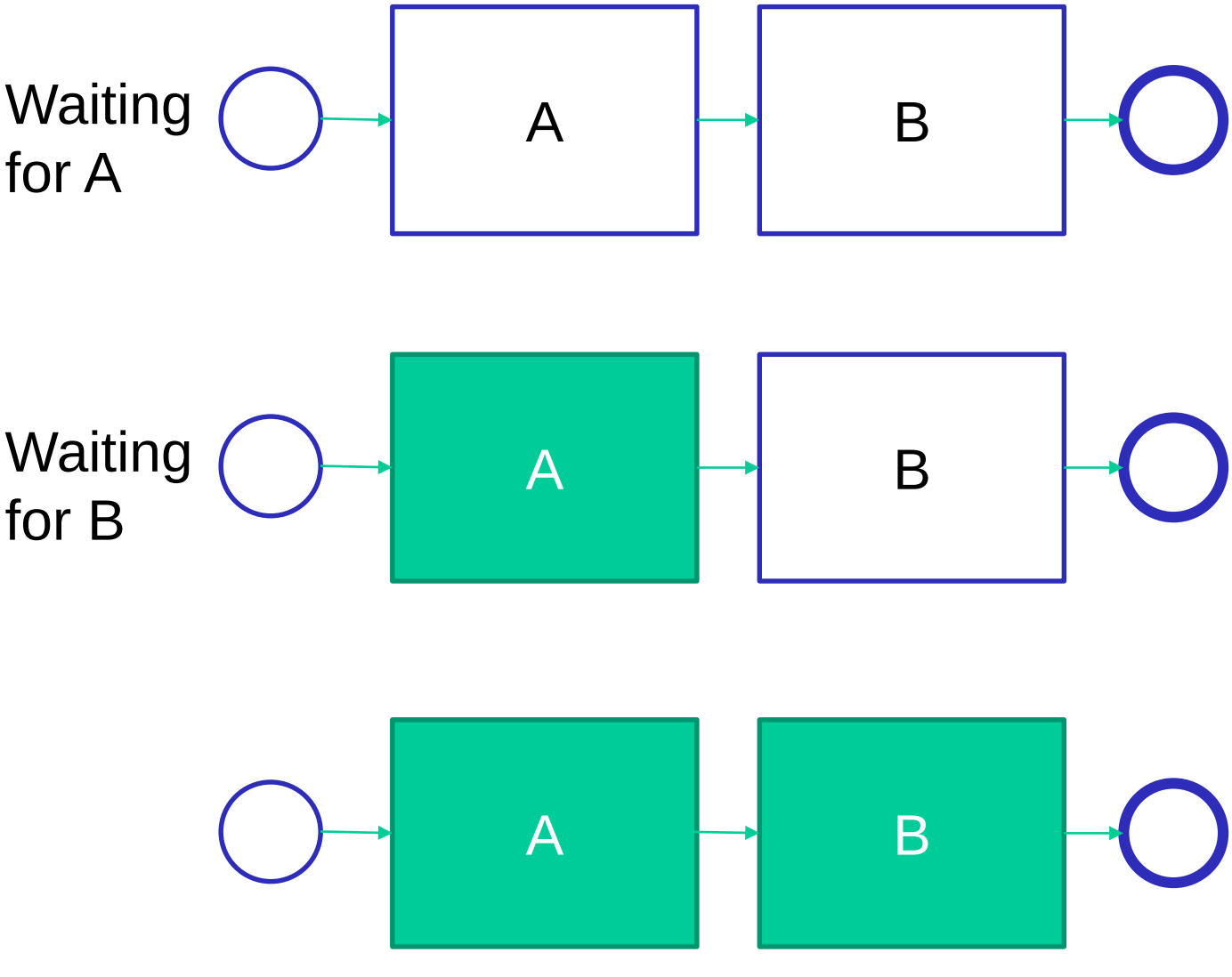
Agenda

- Introduction and problem statement
- Goal of this work
- E-GSM
- From BPMN to E-GSM
- Validation
- Conclusion

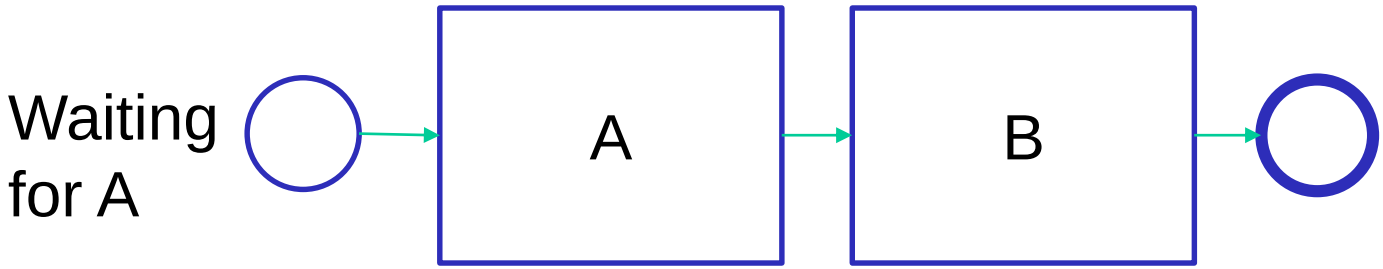
Introduction and problem statement

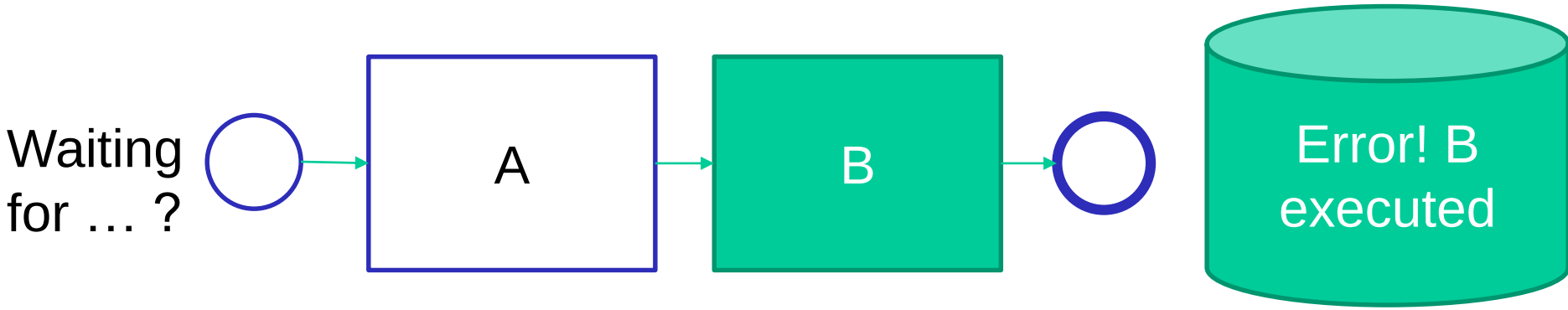
- Using control flow languages to monitor the process execution has the following shortcomings:
  - If activities do not respect their execution order, an exception is raised and the rest of the process cannot be monitored
    - It is desirable to continue monitoring and assess to which extent the whole process violates the model
  - They rely on explicit start and termination messages to understand which activities are running
    - When the process is not automated, such messages are unavailable
  - Processes that are outside the organization that performs monitoring cannot be forced to follow the model.
    - In principle, activities could be actually executed in arbitrary order, regardless on the model

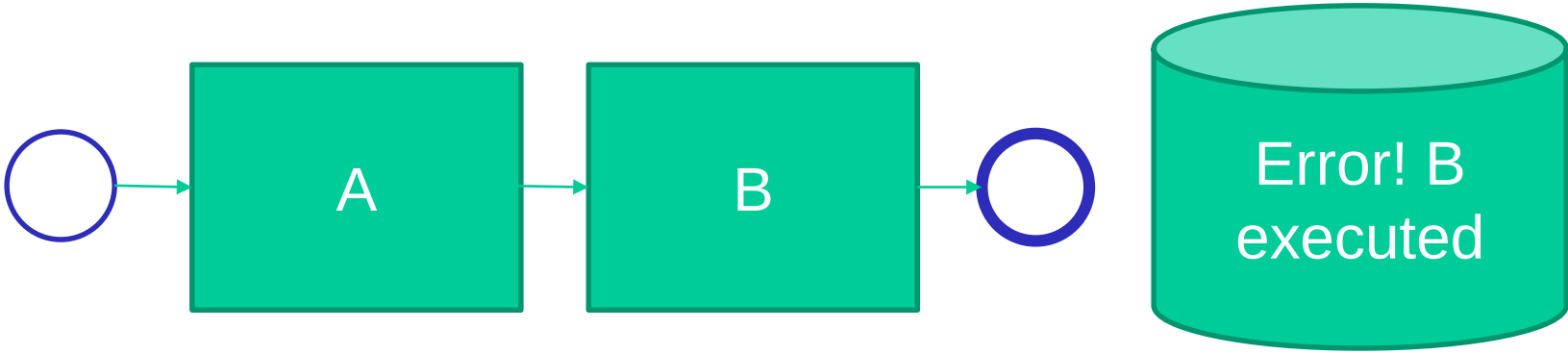
Motivating example



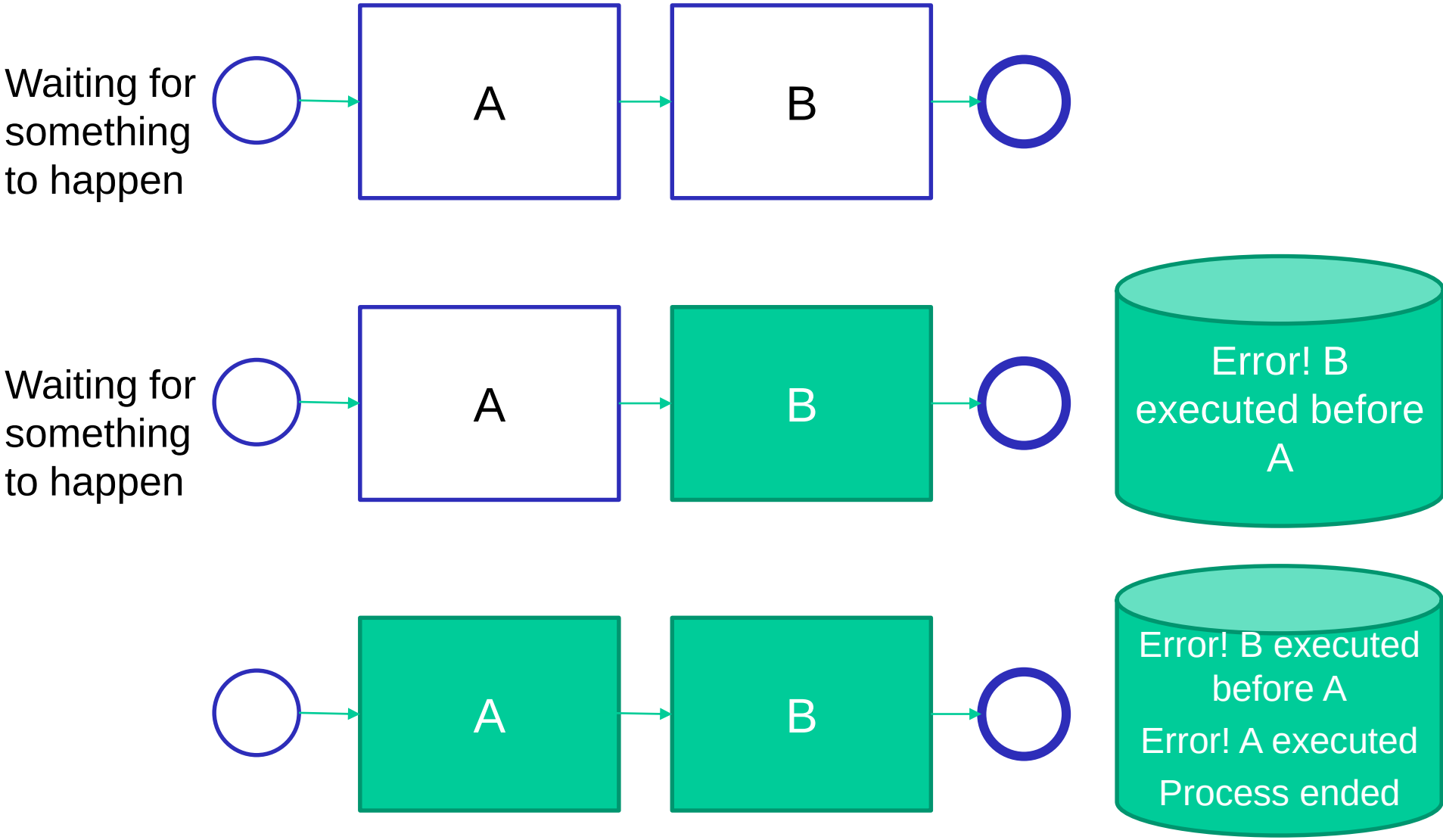
Motivating example







Motivating example



Goal of this work

- The adoption of artifact-centric languages (e.g., GSM) can overcome these limitations
- Declarative languages allow more degrees of freedom
- Declarative languages can be used to passively monitor the execution of processes

Goal of this work

Propose an approach for configuring process monitor by using a declarative model directly derived from a process model that will be executed, which is defined with traditional control-flow languages.

BPMN

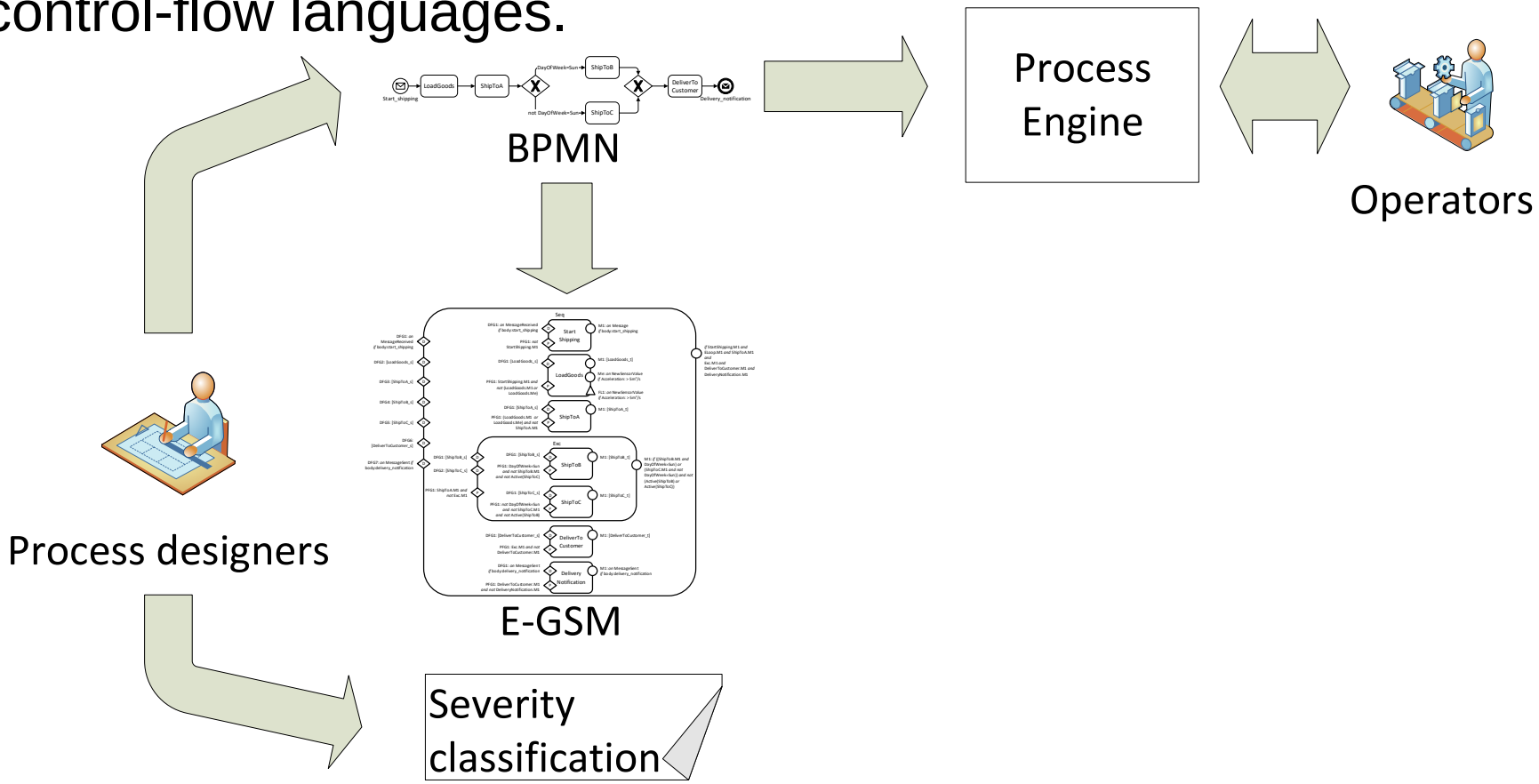
Process Engine

Operators

Process designers

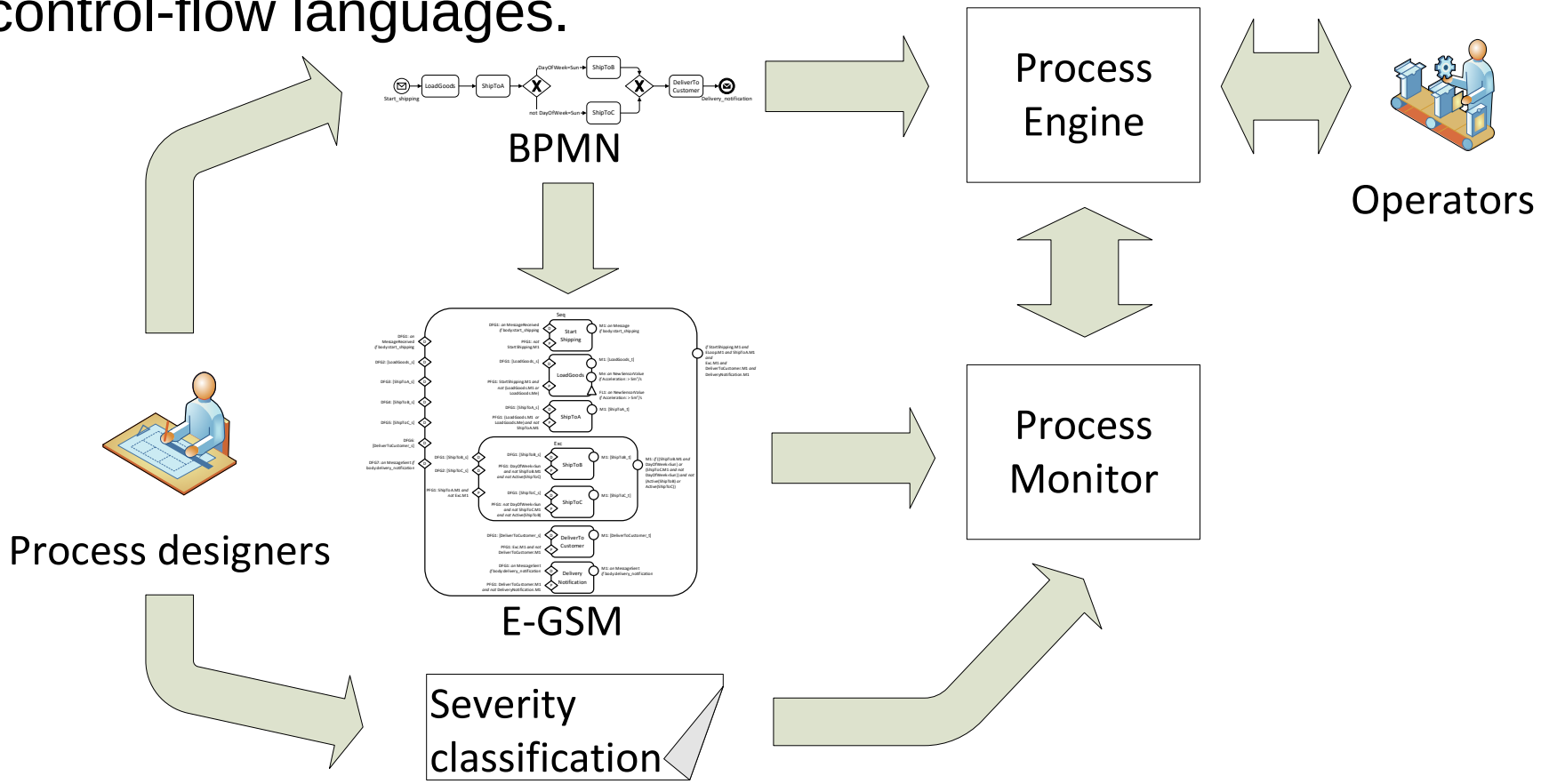
Goal of this work

Propose an approach for configuring process monitor by using a declarative model directly derived from a process model that will be executed, which is defined with traditional control-flow languages.



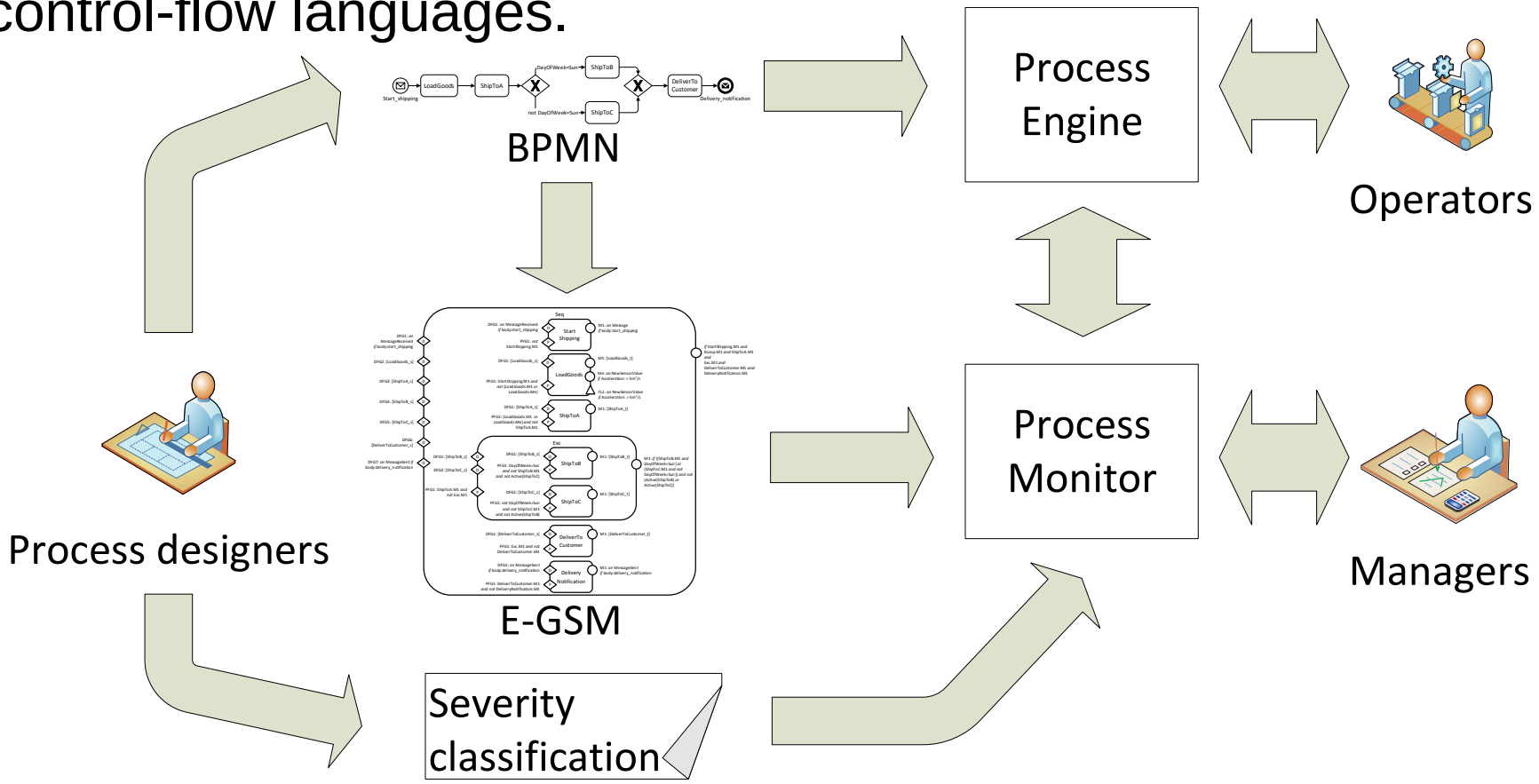
Goal of this work

Propose an approach for configuring process monitor by using a declarative model directly derived from a process model that will be executed, which is defined with traditional control-flow languages.



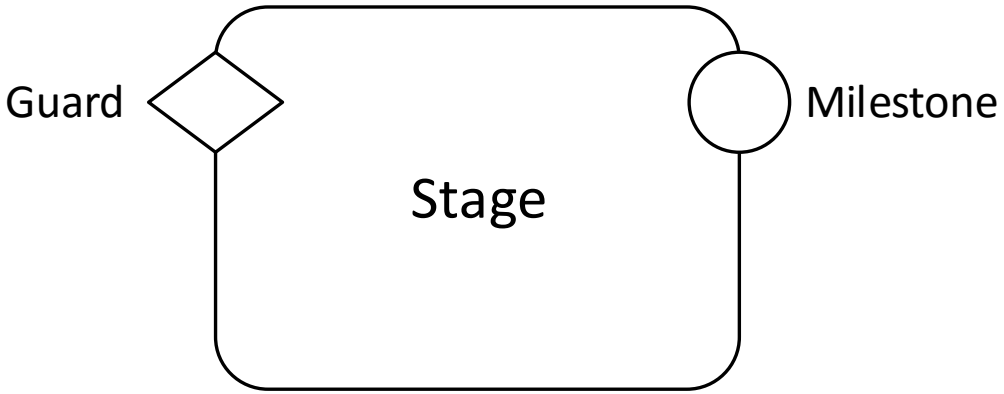
Goal of this work

Propose an approach for configuring process monitor by using a declarative model directly derived from a process model that will be executed, which is defined with traditional control-flow languages.



Guard-Stage-Milestone (GSM) Notation

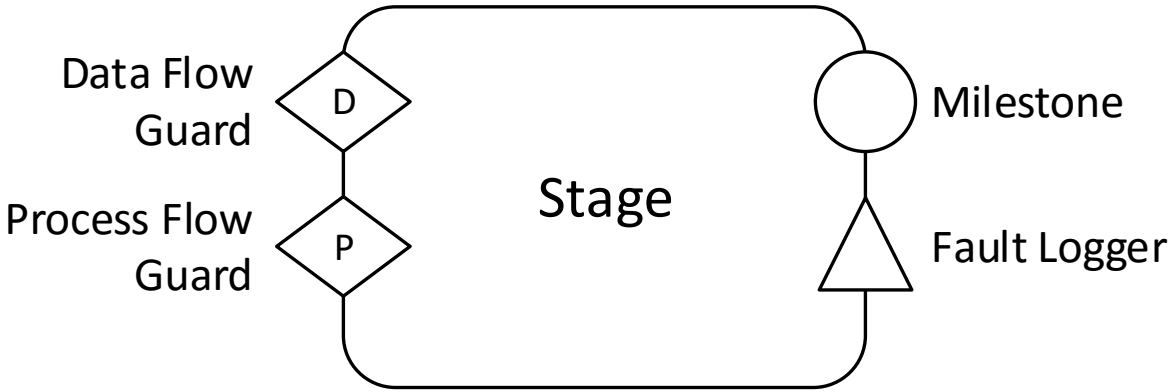
- The Guard-Stage-Milestone (GSM) notation [1] is the ideal candidate for capturing the execution of processes:
  - Guard (G) determines the start of each task based on events
  - Milestone (M) determines the end of each task based on events
  - Events can be internal or external, involving conditions on sensor data, explicit messages, etc.



[1] Hull et al.: Introducing the guard-stage-milestone approach for specifying business entity lifecycles.

E-GSM

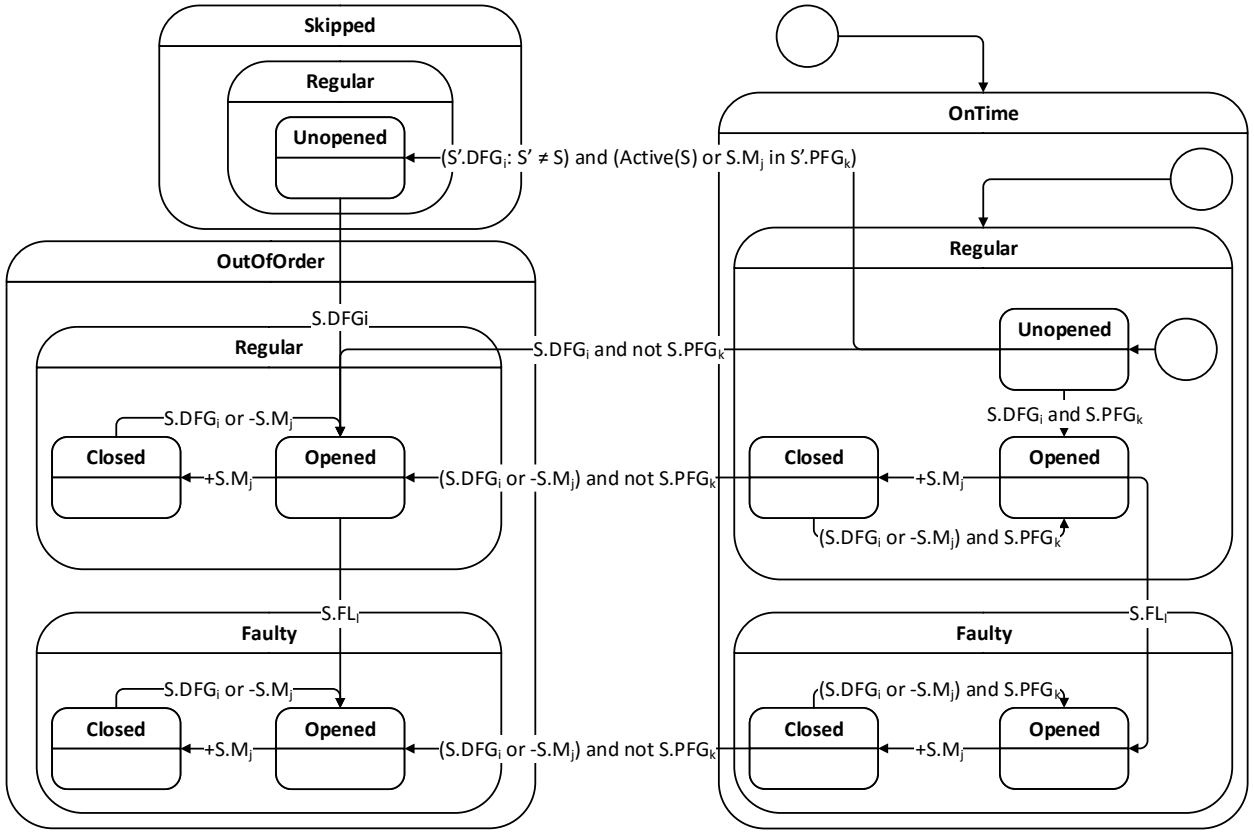
- E-GSM, our extension of GSM developed in a previous work [2], introduces the following changes:
  - Guard distinguished in Data Flow Guard and Process Flow Guard:
    - Data Flow Guard (DFG) determines task activation
    - Process Flow Guard (PFG) defines the expected process flow
  - Fault Logger (FL) annotation introduced:
    - Conditions that identify a violation of the task's constraints and invalidate it
    - If a task is invalidated, it is not terminated



[2] Baresi et al.: A GSM-based approach for monitoring cross organization business processes using smart objects

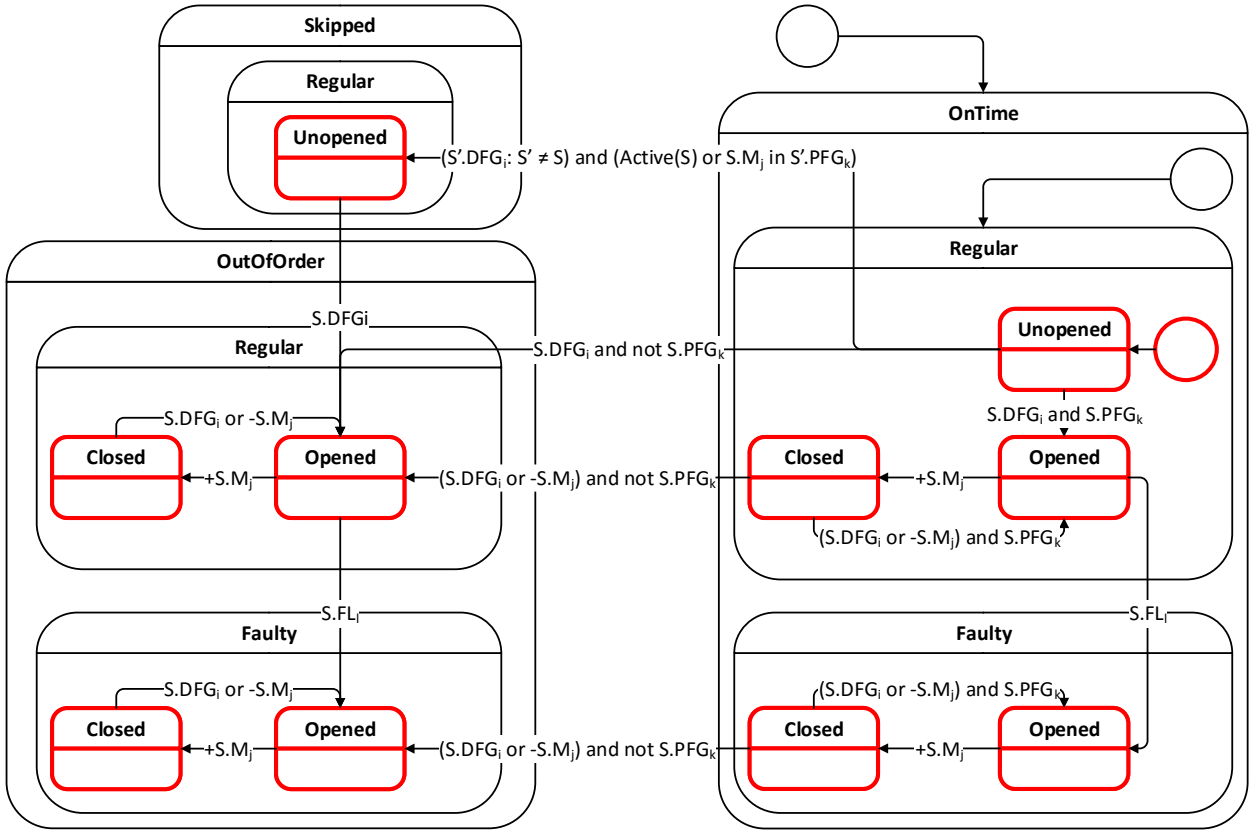
E-GSM Stage lifecycle

- 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



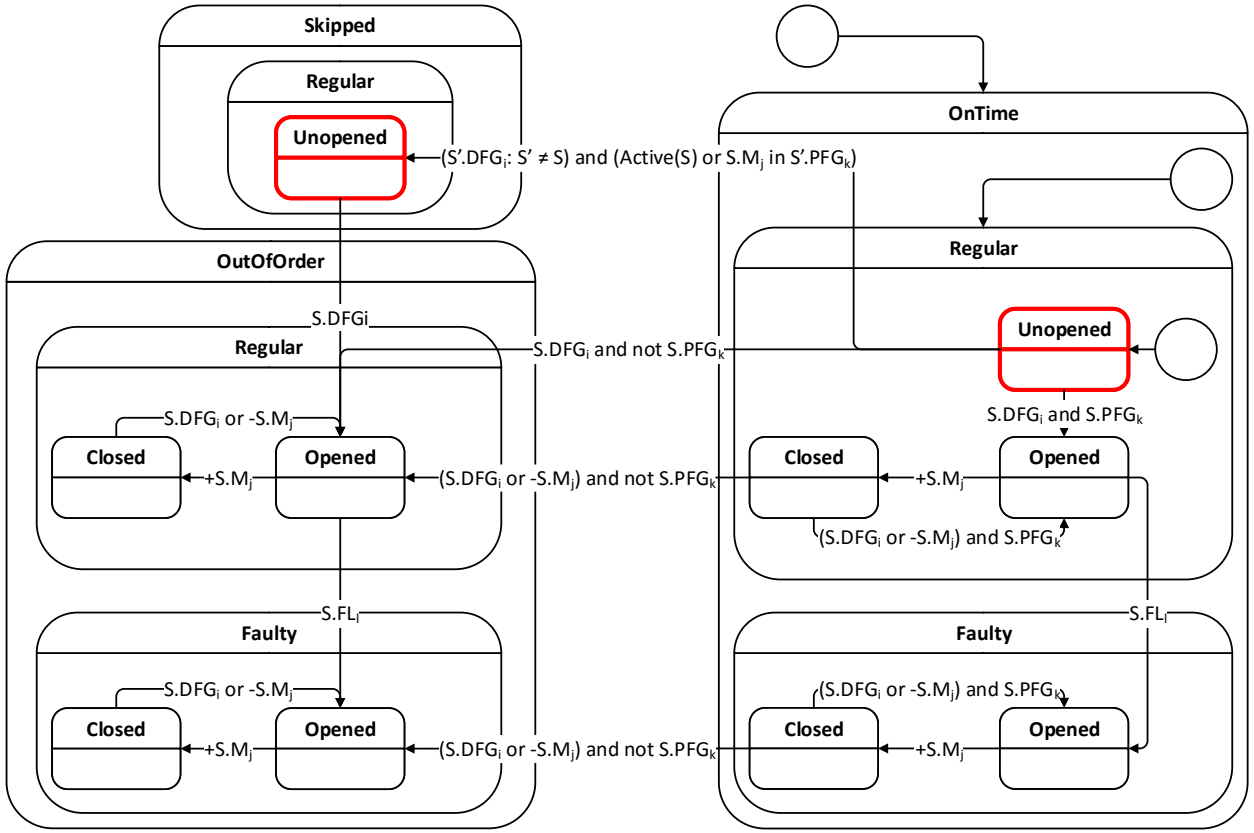
E-GSM Stage lifecycle

- 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



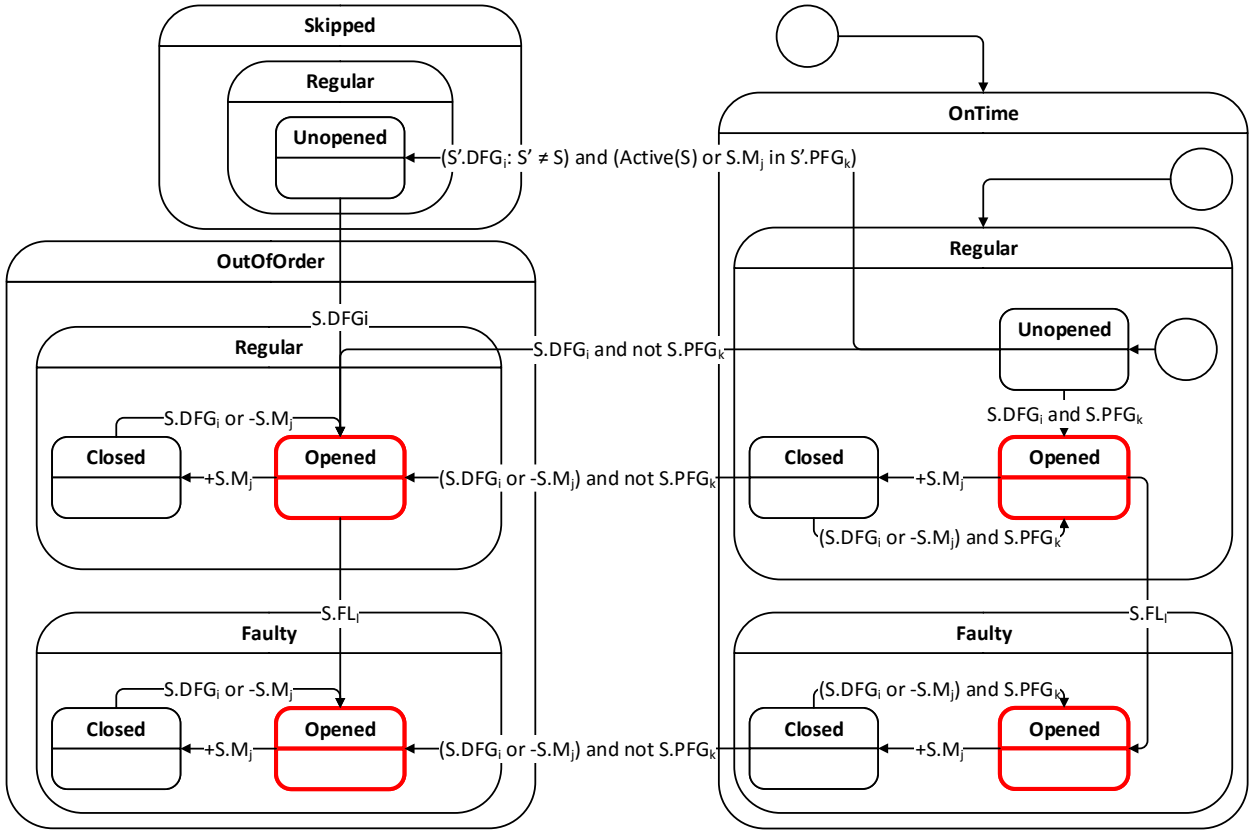
E-GSM Stage lifecycle

- 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



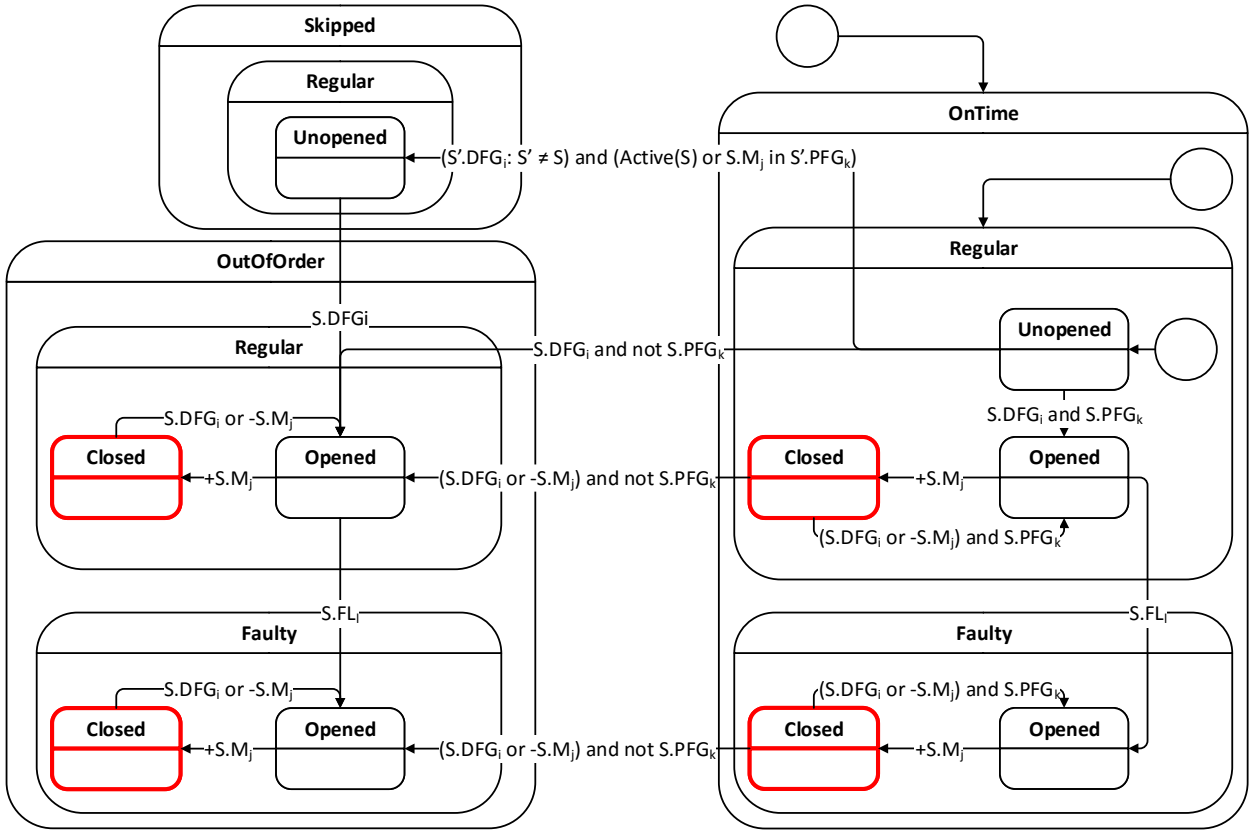
E-GSM Stage lifecycle

- 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



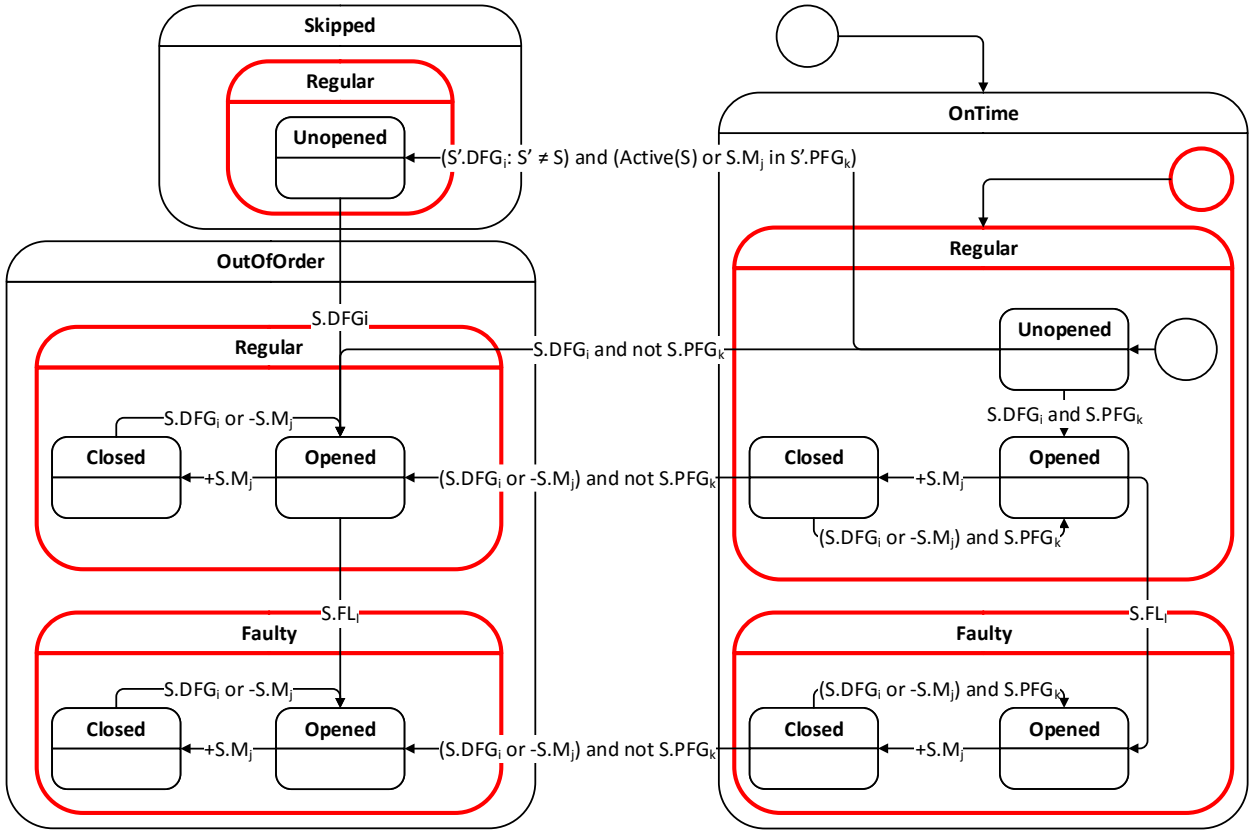
E-GSM Stage lifecycle

- 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



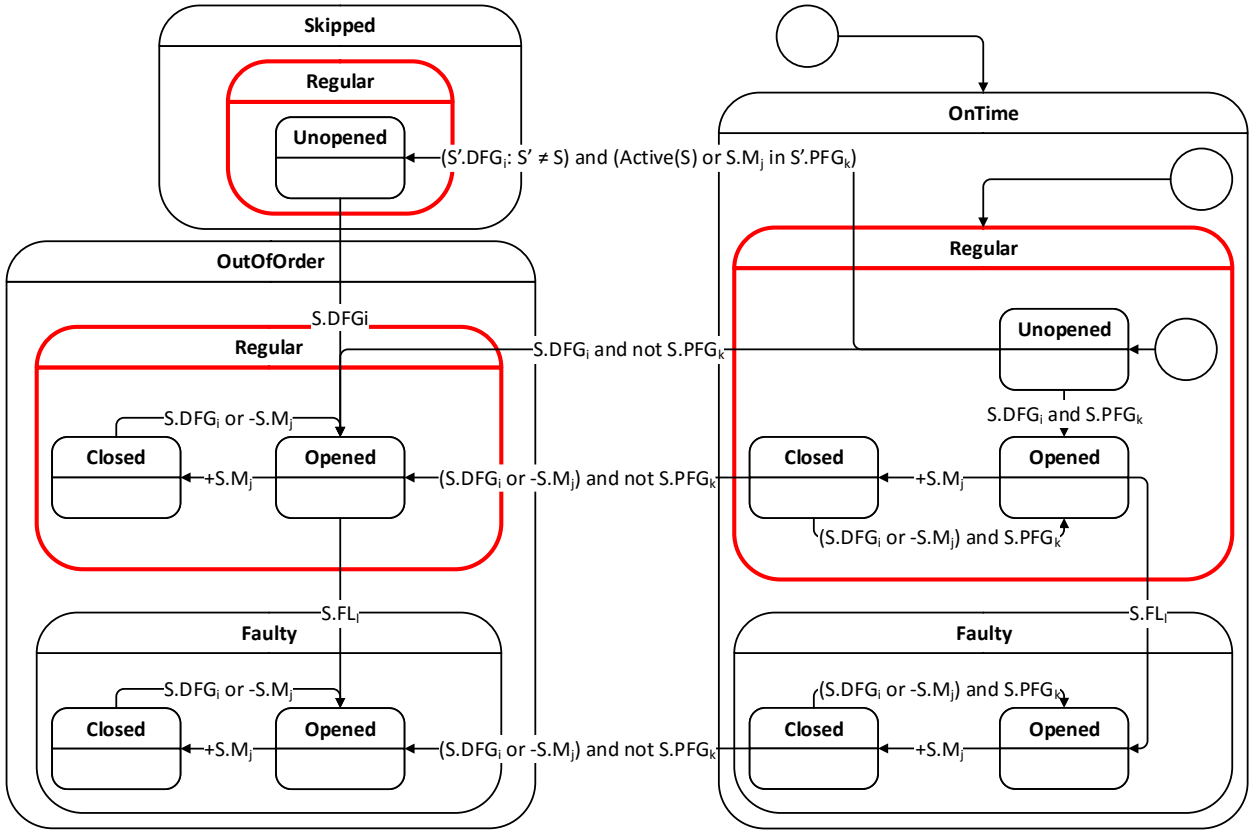
E-GSM Stage lifecycle

- 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



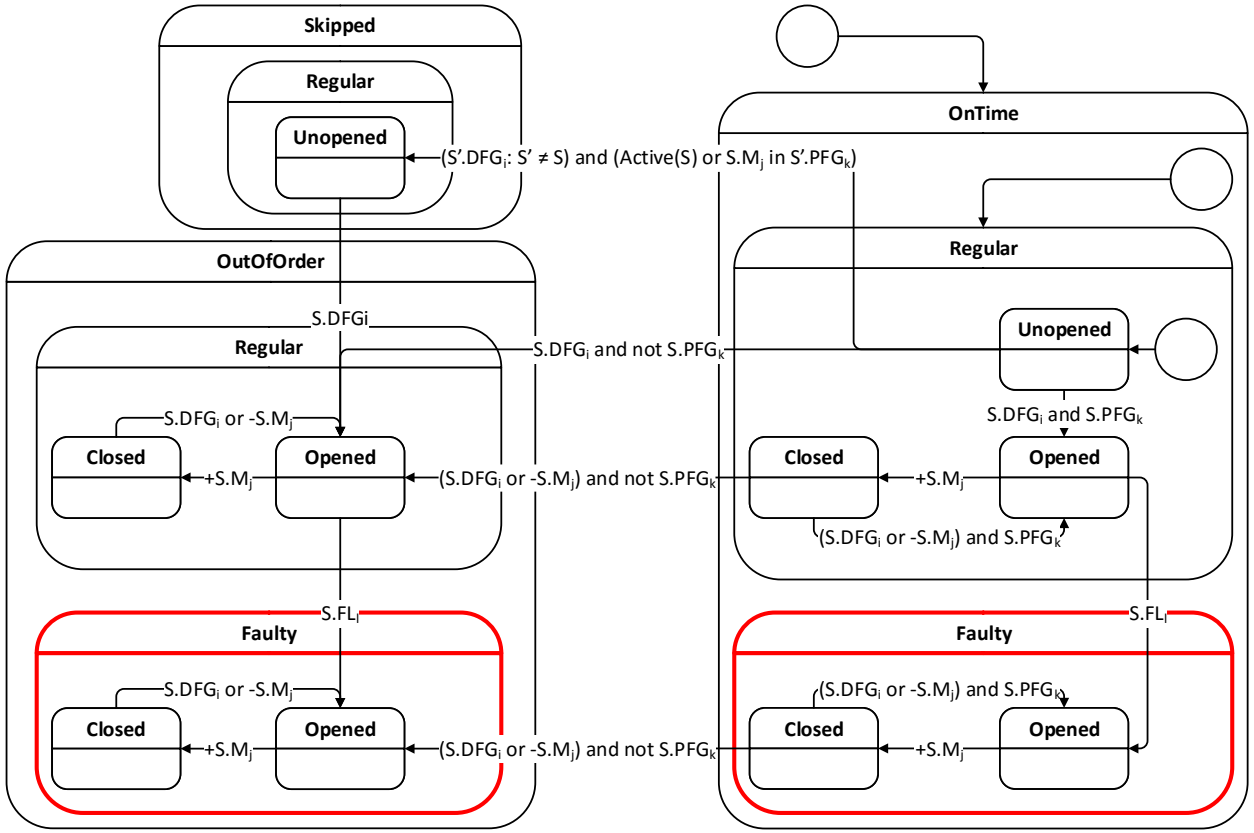
E-GSM Stage lifecycle

- 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



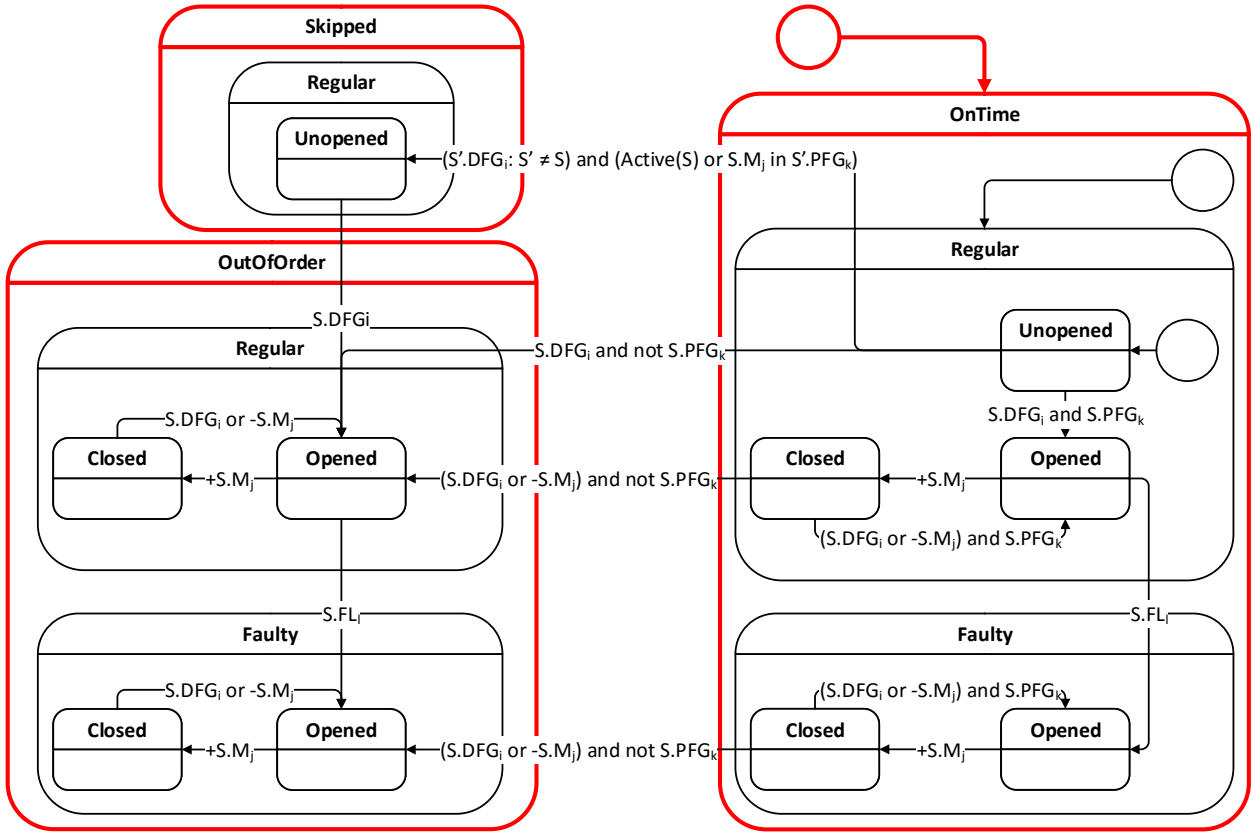
E-GSM Stage lifecycle

- 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



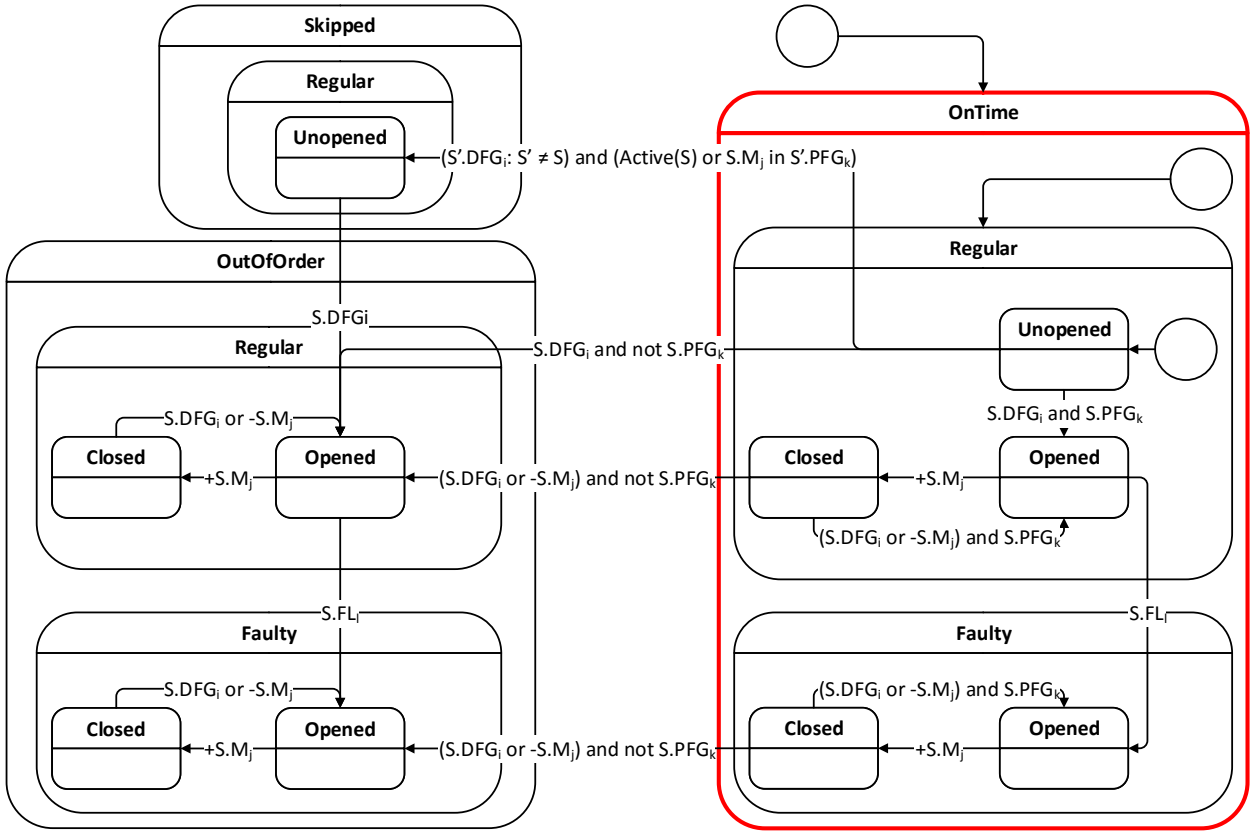
E-GSM Stage lifecycle

- 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



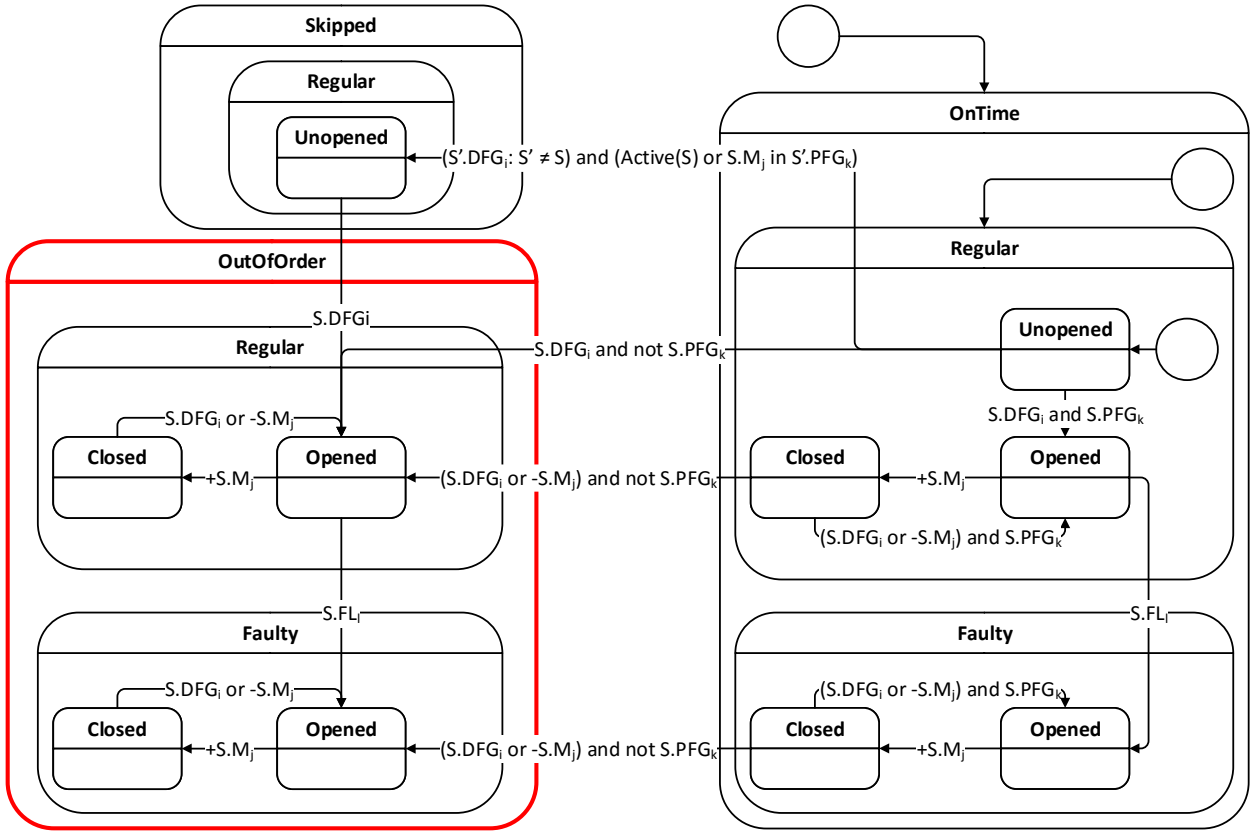
E-GSM Stage lifecycle

- 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



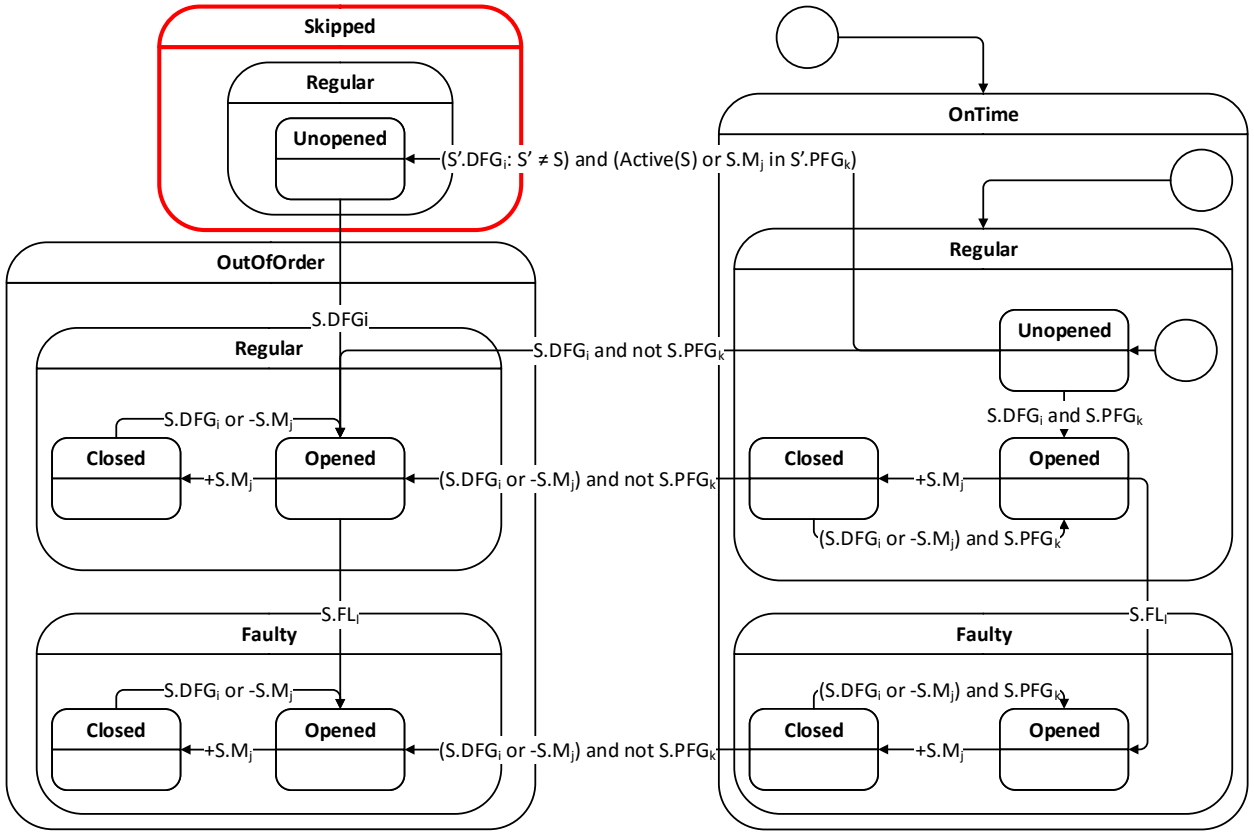
E-GSM Stage lifecycle

- 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



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**

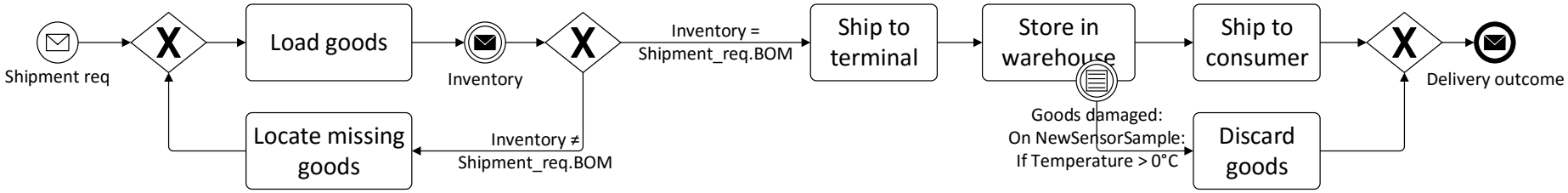


Severity classification

- By predicating on the execution state, outcome and compliance of each stage, designers can define rules to let the process monitoring assess how severely the execution of the whole process is affected.
- An example of severity classification is shown below

Severity	Outcome (S_y.o)	Compliance (S_z.c)	Status (S_x.s)
None	$\forall S_y$: S_y.o = regular	$\forall S_z$: S_z.c = onTime	$\forall S_x$: S_x.s = unopened $\vee$ S_x.s = opened $\vee$ S_x.s = closed
Low	$\forall S_y$: S_y.o = regular	$\exists S_z$: S_z.c = outOfOrder	$\forall S_x$: S_x.s = unopened $\vee$ S_x.s = closed
Medium-low	$\exists S_y$: S_y.o = faulty	$\forall S_z$: S_z.c = onTime	$\forall S_x$: S_x.s = unopened $\vee$ S_x.s = opened $\vee$ S_x.s = closed
Medium	$\forall S_y$: S_y.o = regular	$\exists S_z$: S_z.c = outOfOrder $\vee$ S_z.c = skipped	$\exists S_x$: S_x.s = opened
Medium-high	$\forall S_y$: S_y.o = regular	$\exists S_z$: S_z.c = skipped	$\forall S_x$: S_x.s = unopened $\vee$ S_x.s = closed
High	$\exists S_y$: S_y.o = faulty	$\exists S_z$: S_z.c = outOfOrder $\vee$ S_z.c = skipped	$\forall S_x$: S_x.s = unopened $\vee$ S_x.s = opened $\vee$ S_x.s = closed

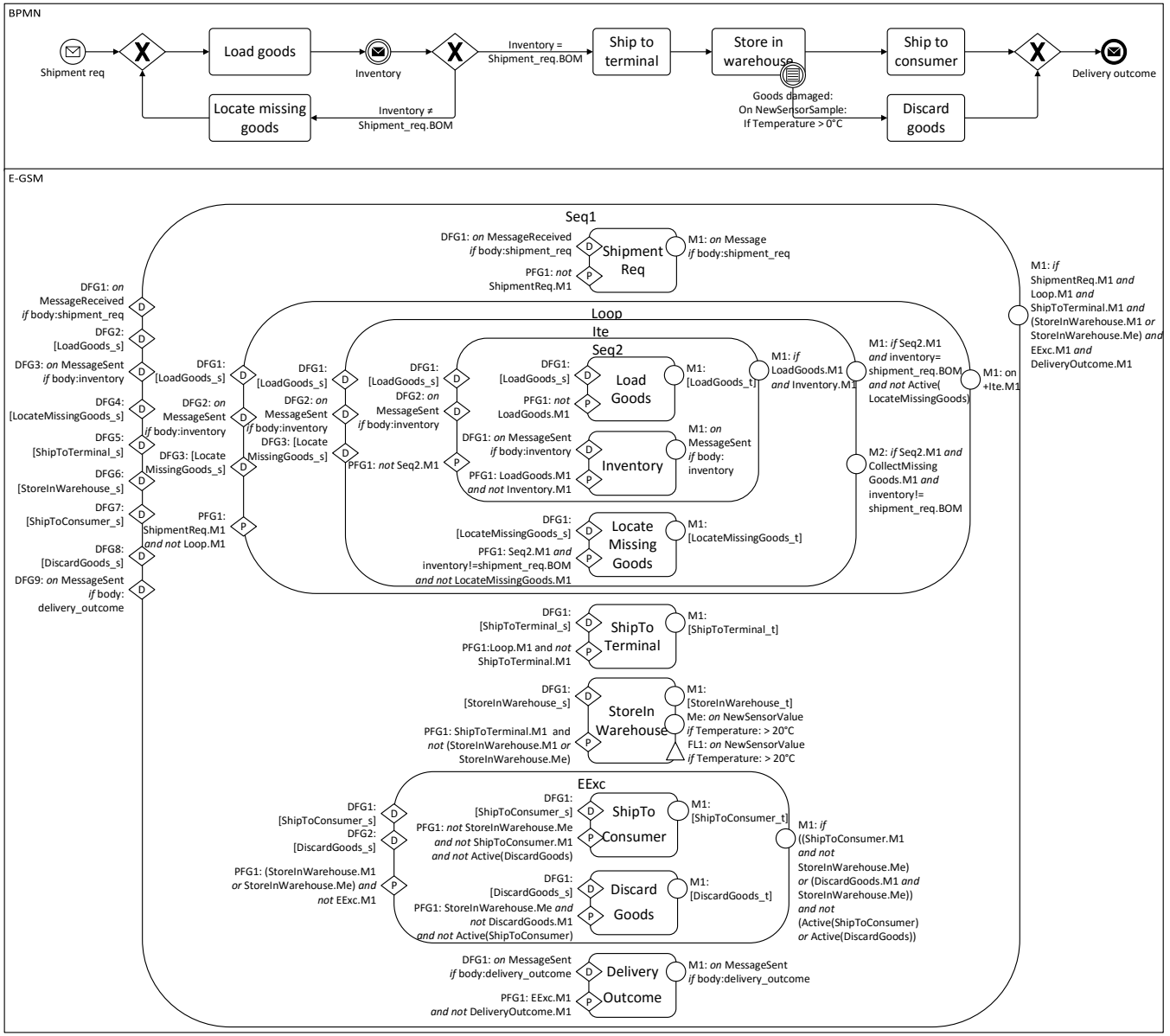
Running example



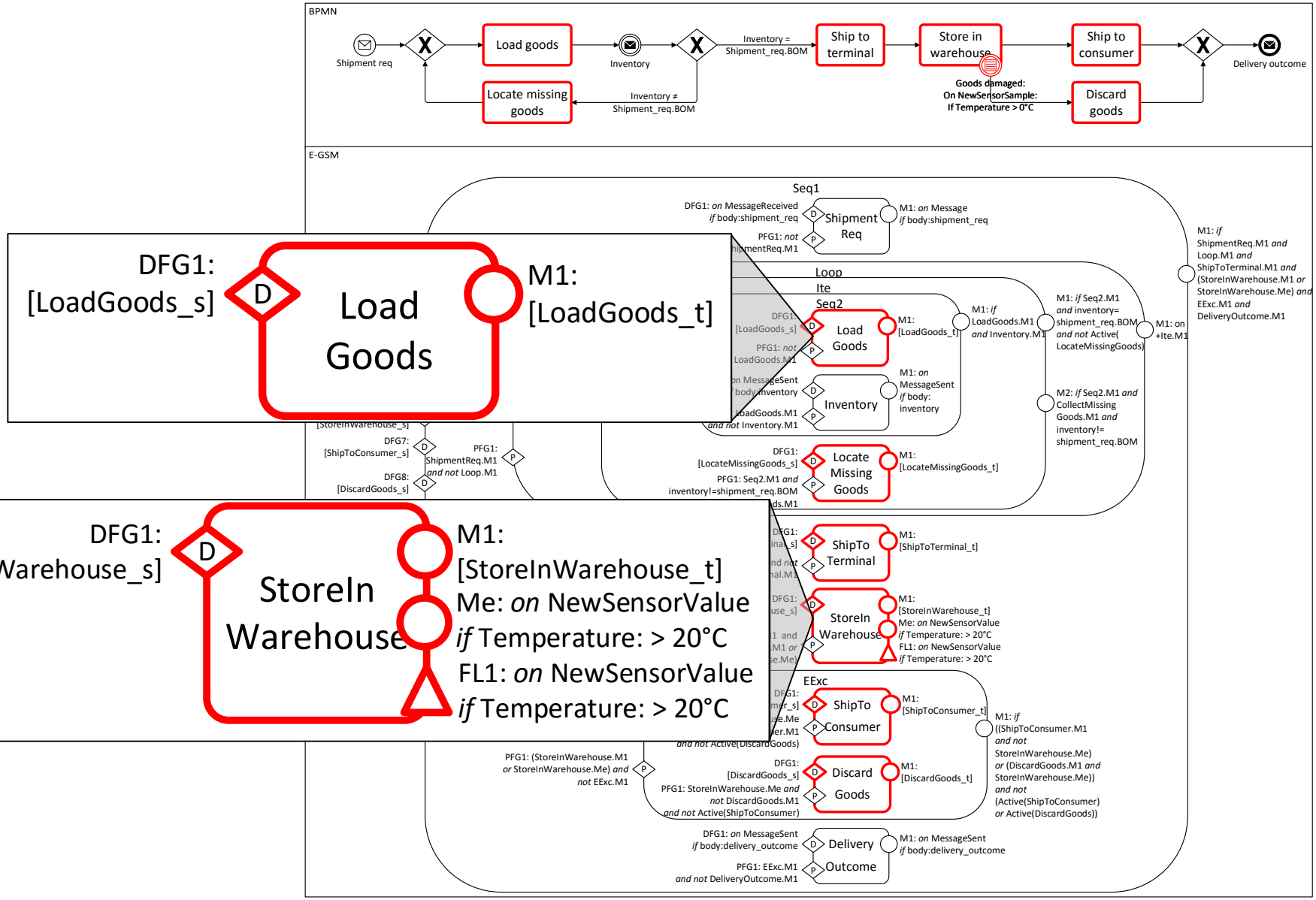
- Simplified shipping process of temperature-sensitive goods
  - Goods are loaded by shipper R into thermally insulated containers
  - R ships the container to inland terminal I
  - I temporarily stores the container in warehouse
  - Shipper T picks the terminal up from I and delivers to the customer
  - All the goods requested by the customer must be shipped at once
  - If the goods are exposed to high temperature, they must be discarded

Translation by Example

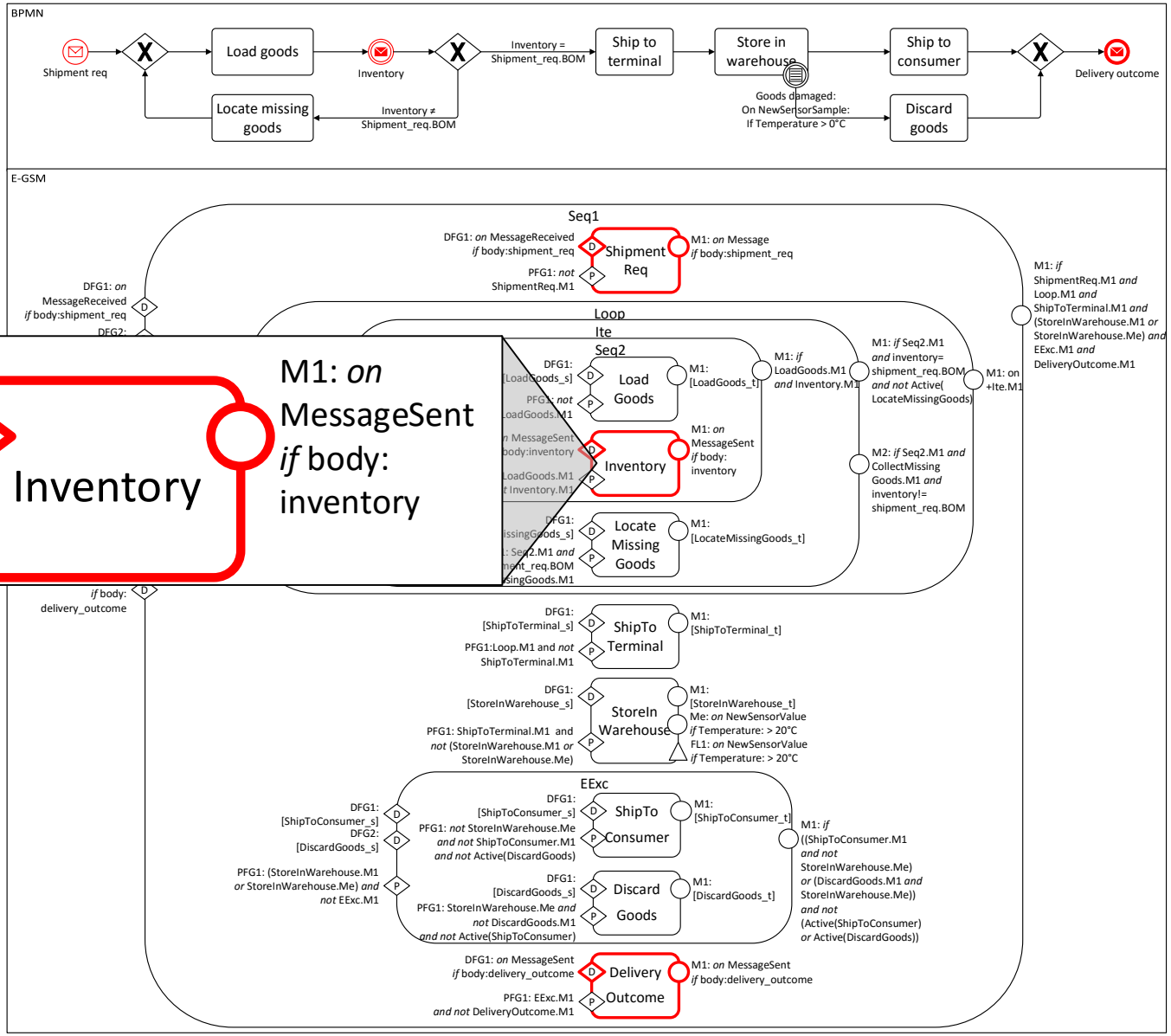
POLITECNICO DI MILANO



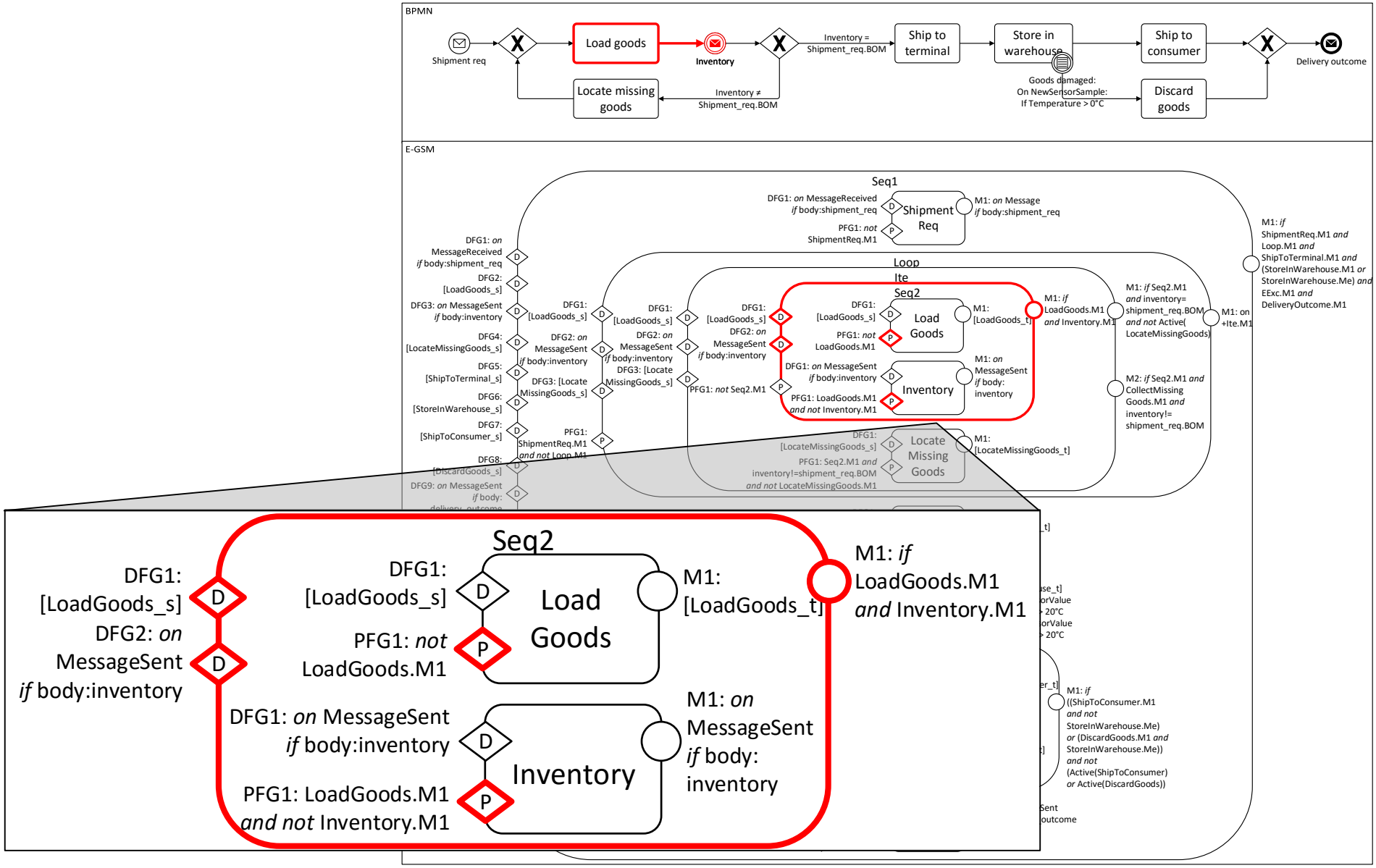
Translation by Example Activities



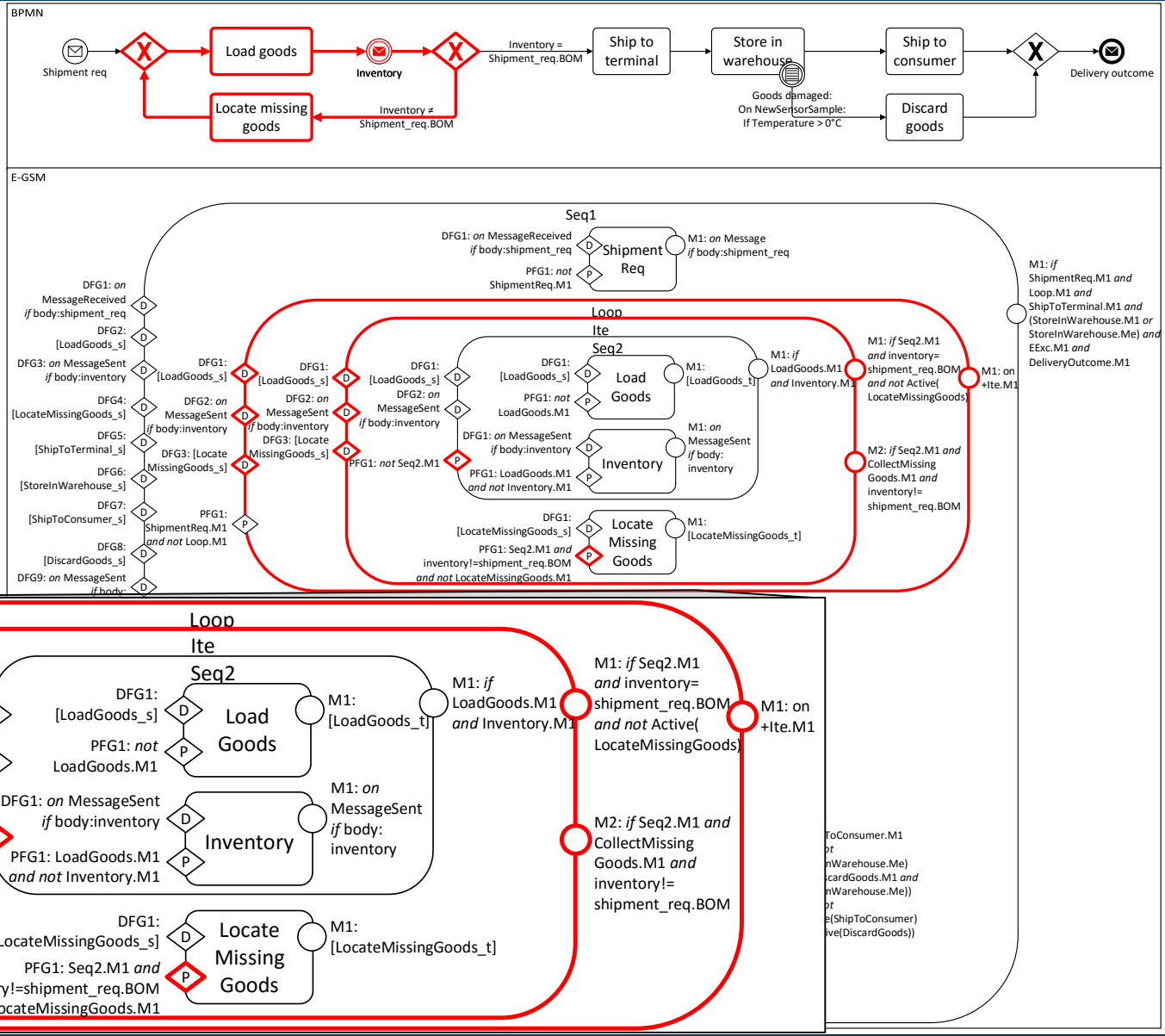
Translation by Example
Non-boundary Events



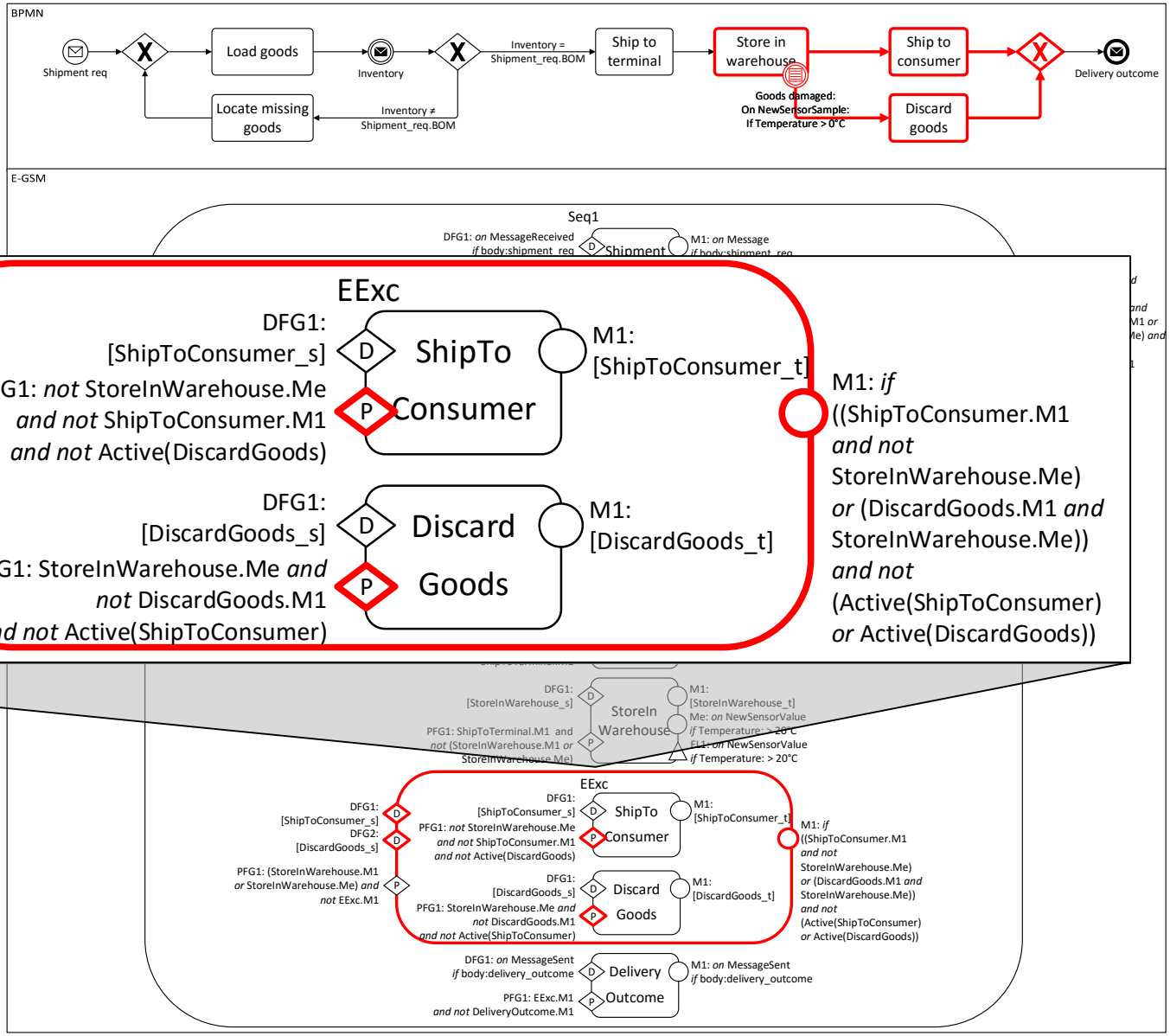
Translation by Example
Inner Sequence Block



Translation by Example
Loop Block

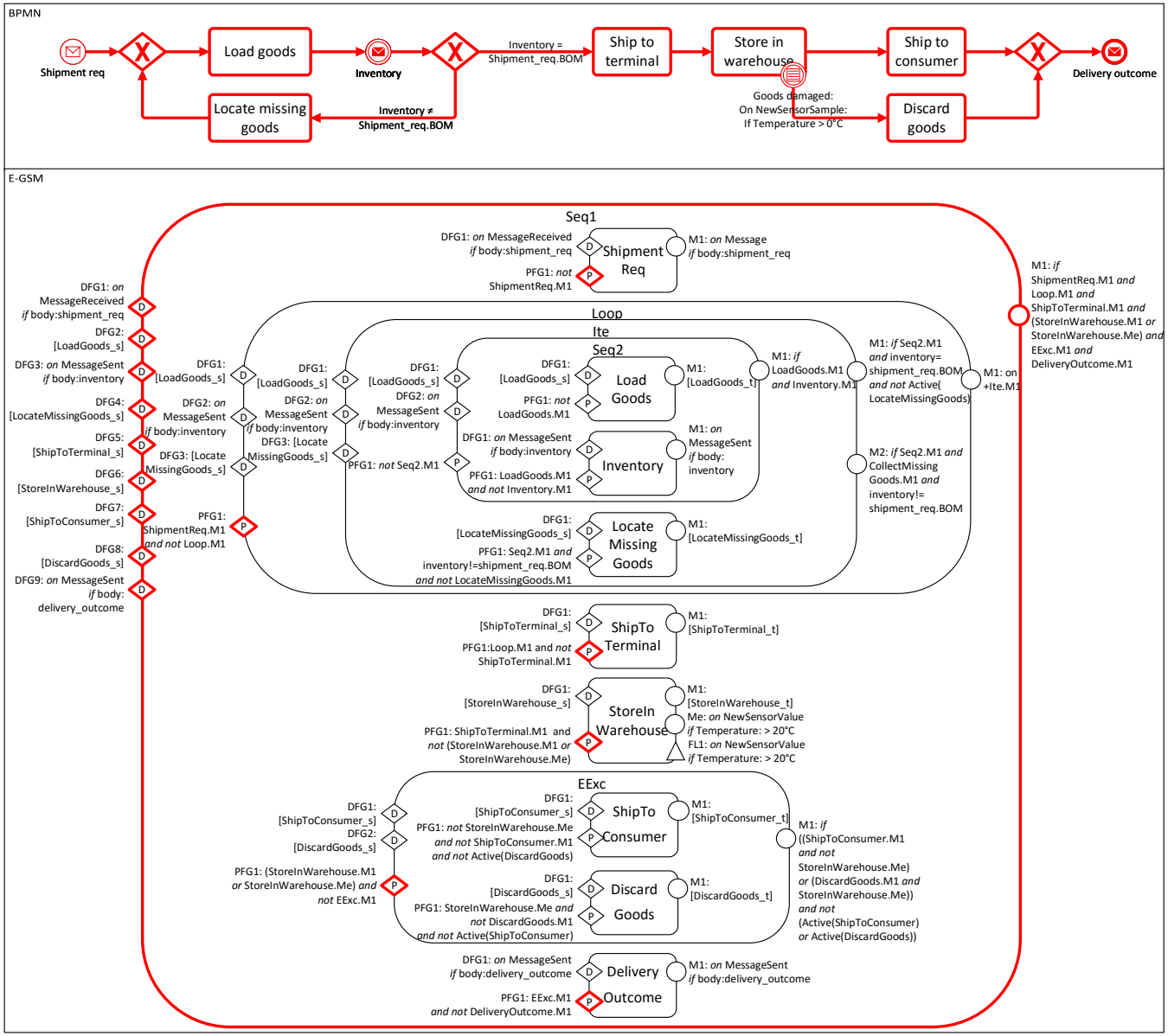


Translation by Example
Forward Exception Handling Block



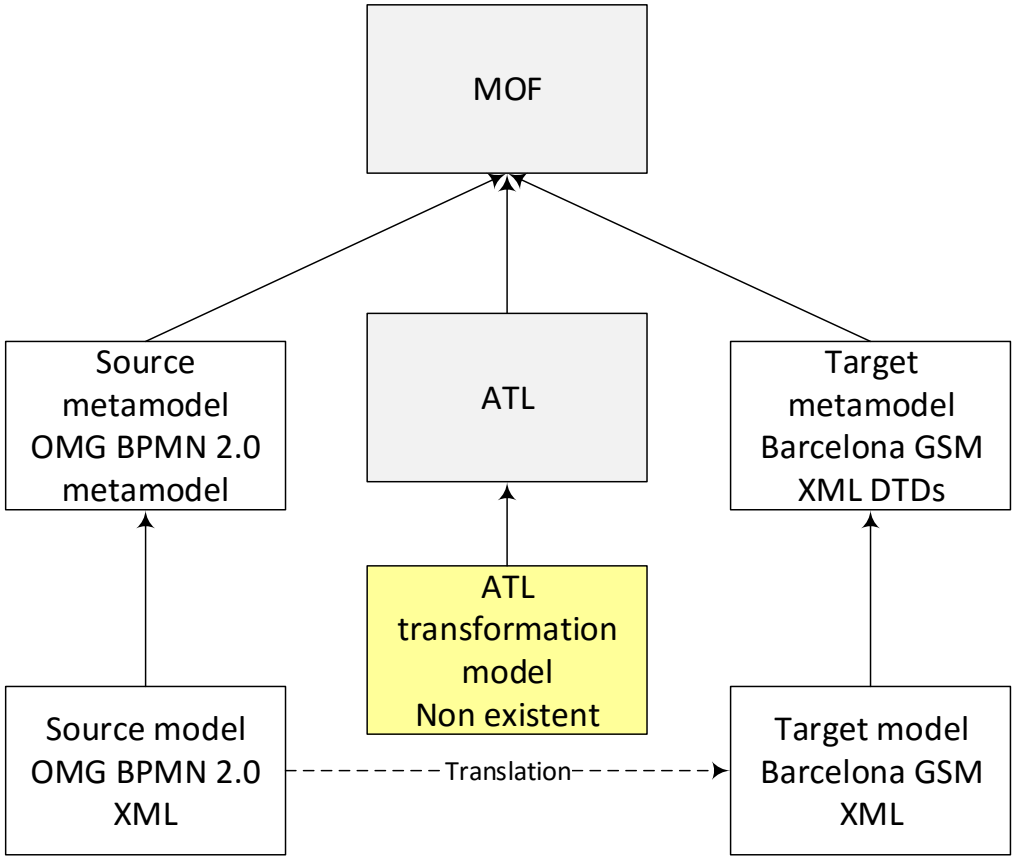
Translation by Example
Outer Sequence Block

POLITECNICO DI MILANO



From BPMN to E-GSM
Automating the translation

- BPMN to E-GSM translator implemented in ATLAS Transformation Language (ATL) [3]
  - Using OMG BPMN 2.0 specifications as source model
  - Using XML-based GSM definitions from Barcelona [4] as target model

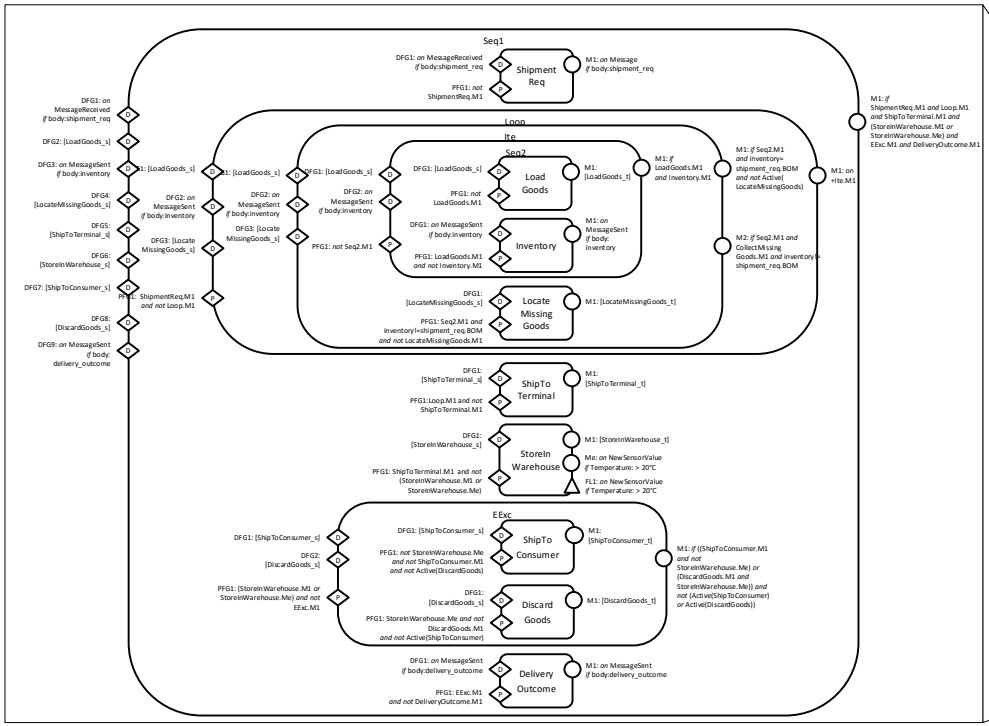


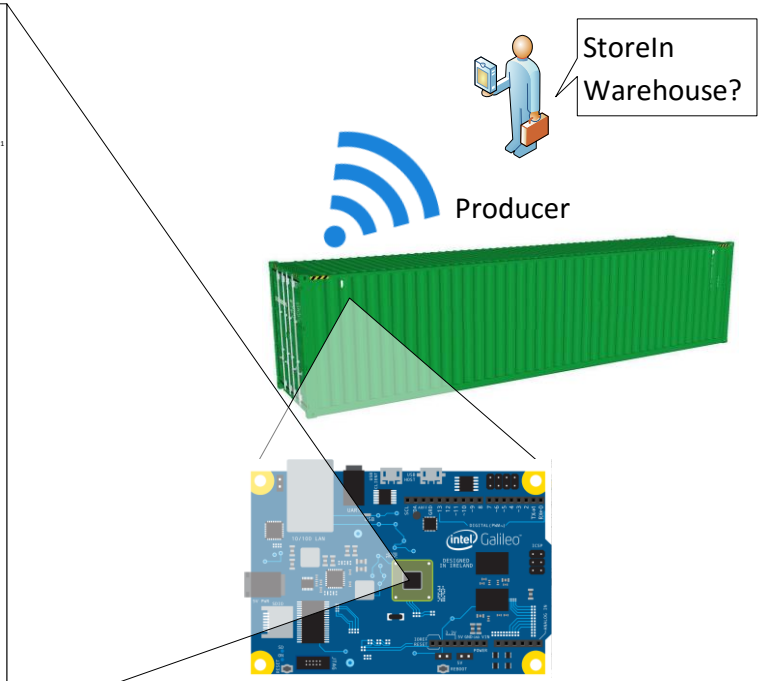
[3] Jouault et al.: ATL: a QVT-like transformation language

[4] Heath et al.: Barcelona: A design and runtime environment for declarative artifact-centric BPM.

Validation

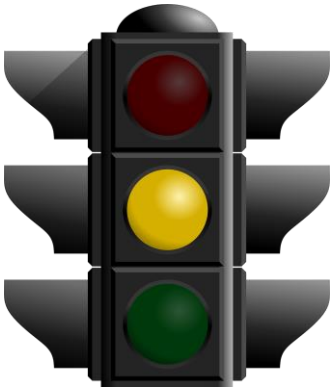
- By exploiting the IoT paradigm, the shipping container can be turned into a smart object running an E-GSM engine
  - The E-GSM model derived from the BPMN process feeds the engine and allows the container to monitor the whole process





Validation
A catastrophic execution

- Cooling system of the warehouse breaks down
- Goods are exposed to high temperature and are spoiled



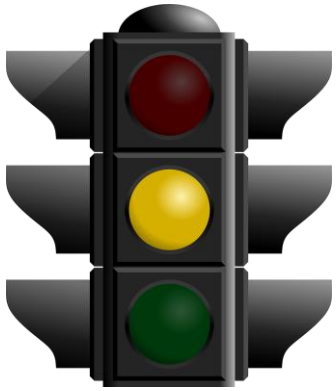
- This causes an economic loss
- This exception is foreseen in the process model
- The inland terminal must discard the goods and terminate the process
- Customer N is unaffected

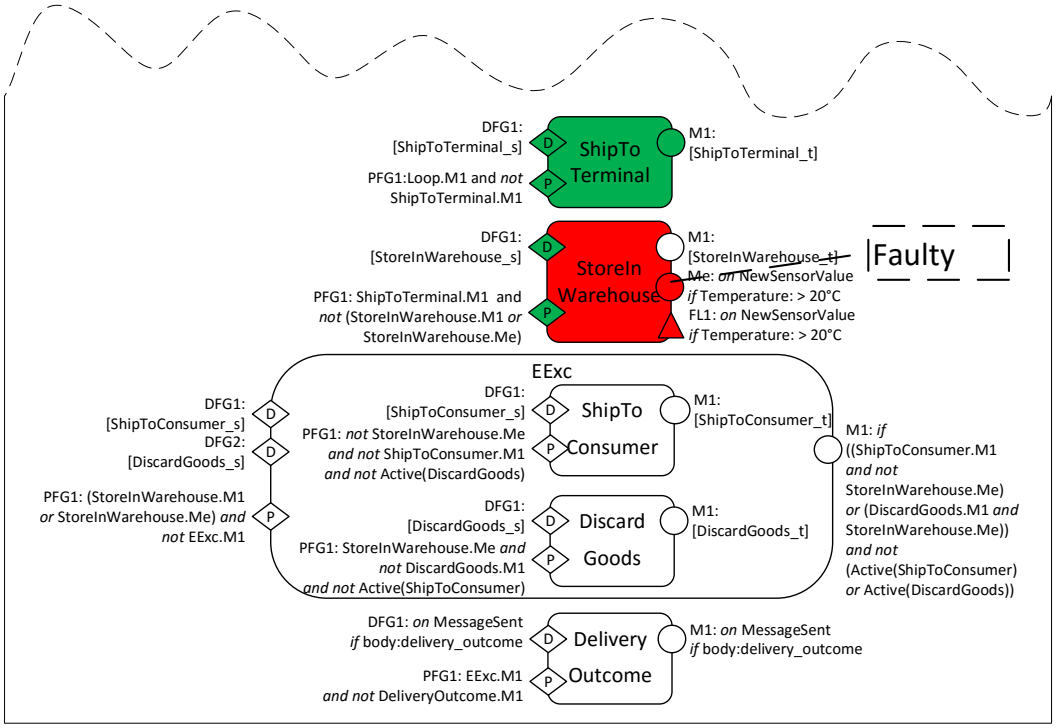
- Up to now, the situation is not so critical

Validation
A catastrophic execution

- Cooling system of the warehouse breaks down
- Goods are exposed to high temperature and are spoiled

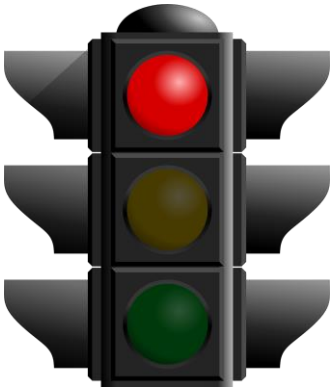
Severity	Outcome (S_y.o)	Compliance (S_z.c)	Status (S_x.s)
Medium-low	$\exists S_y$: S_y.o = faulty	$\forall S_z$: S_z.c = onTime	$\forall S_x$: S_x.s = unopened $\vee$ S_x.s = opened $\vee$ S_x.s = closed





Validation
A catastrophic execution

- Cooling system of the warehouse breaks down
- Goods are exposed to high temperature and are spoiled
- **I ignores the accident and gives the goods to T**
- **T ships the goods**

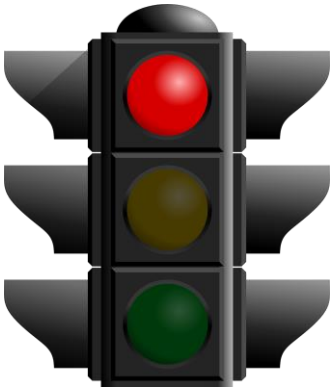


- Customer N receives spoiled goods and gets disappointed
- M must pick the spoiled goods up at N site
- M must plan a new shipping
- An even higher economic loss is produced
- The reputation of M decreases

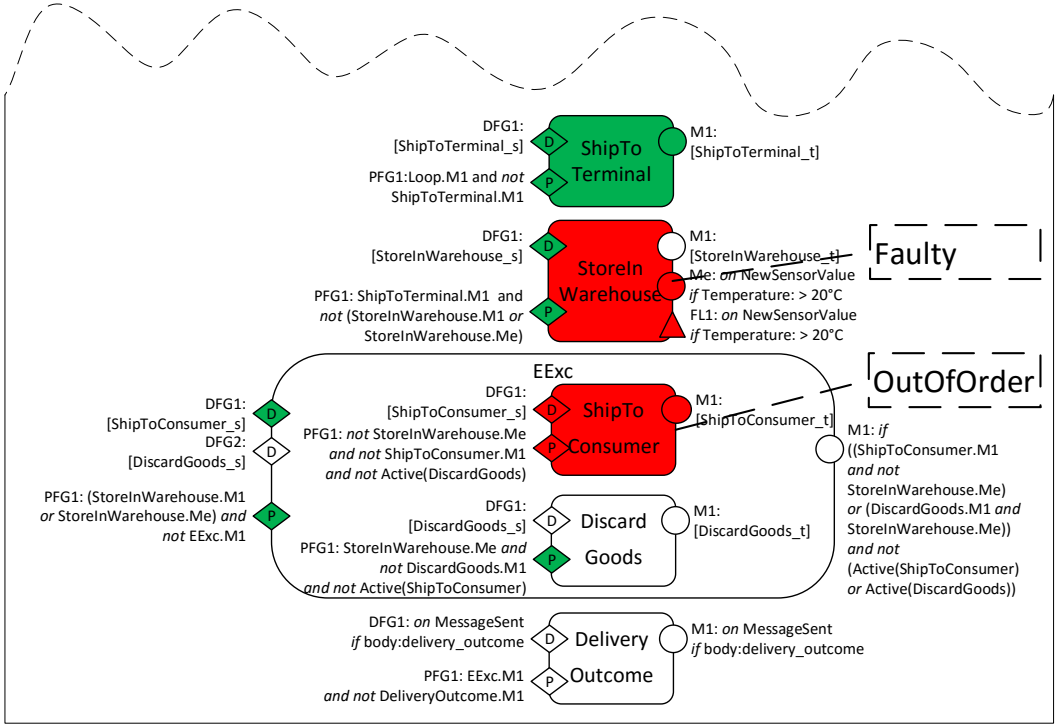
- This execution is a complete catastrophe!

Validation
A catastrophic execution

- Cooling system of the warehouse breaks down
- Goods are exposed to high temperature and are spoiled
- **I ignores the accident and gives the goods to T**
- **T ships the goods**

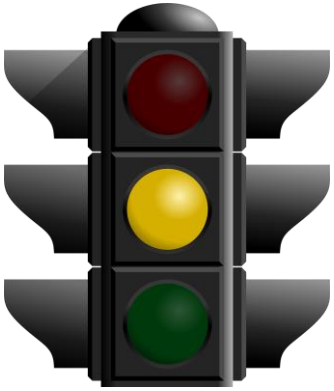


Severity	Outcome (S_y.o)	Compliance (S_z.c)	Status (S_x.s)
High	$\exists S_y$: S_y.o = faulty	$\exists S_z$: S_z.c = outOfOrder ∨ S_z.c = skipped	$\forall S_x$: S_x.s = unopened ∨ S_x.s = opened ∨ S_x.s = closed



Validation
A troublesome yet recoverable execution

- After loading the goods into the container, R begins shipping them
- No inventory is made

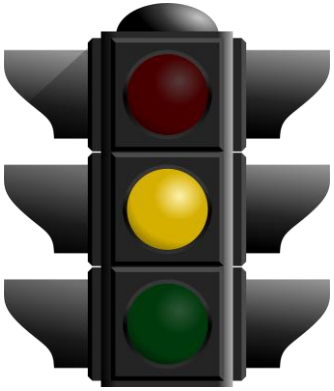


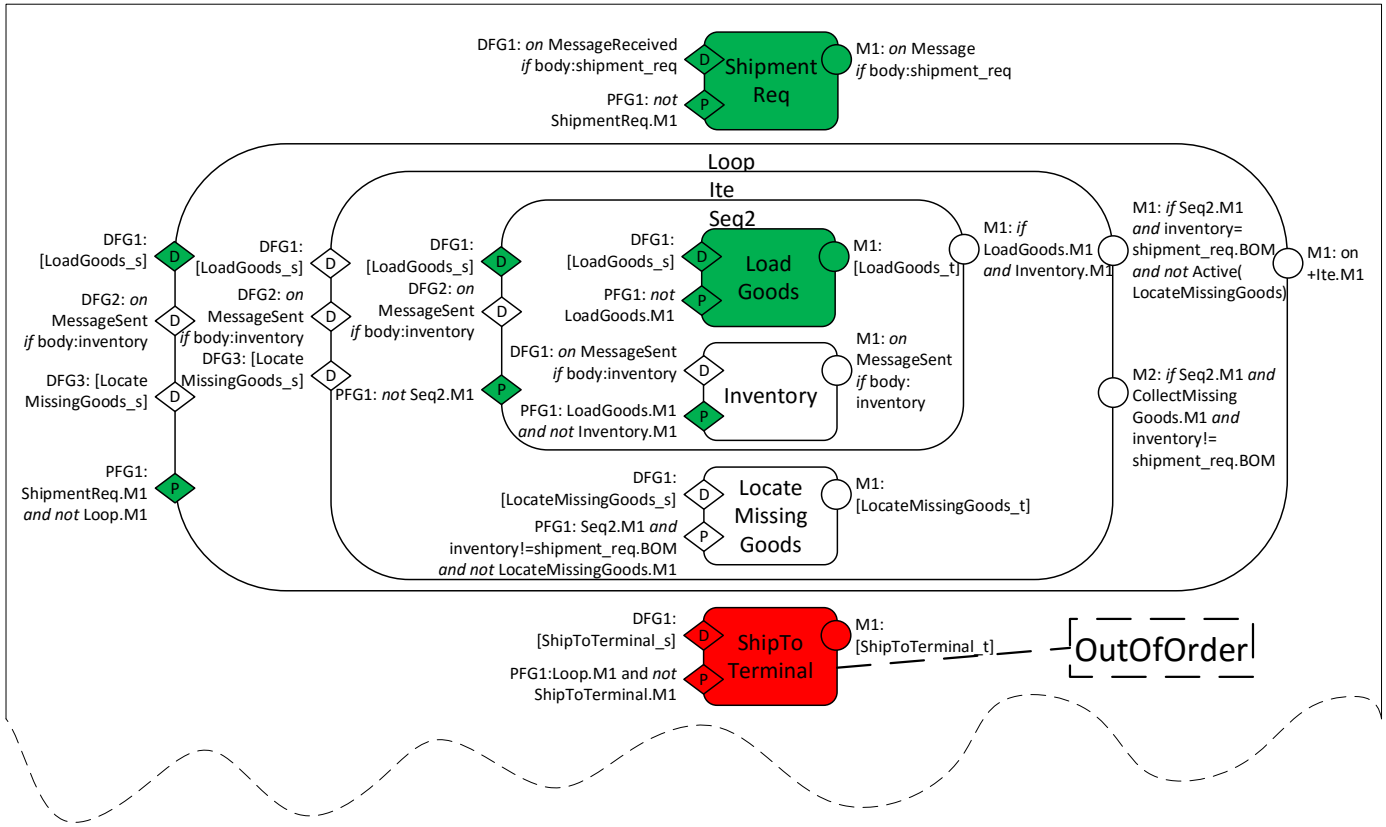
- If all the goods were correctly located and picked up, the process concludes correctly
- If some goods were missing, an additional shipment must be planned
  - This causes a moderate economic loss
  - This slightly decreases the reputation of M
- No goods are lost
- The execution is incorrect, yet recoverable.

Validation
A troublesome yet recoverable execution

- After loading the goods into the container, R begins shipping them
- No inventory is made

Severity	Outcome (S_y.o)	Compliance (S_z.c)	Status (S_x.s)
Medium	$\forall S_y$: S_y.o = regular	$\exists S_z$: S_z.c = outOfOrder $\vee$ S_z.c = skipped	$\exists S_x$: S_x.s = opened





Conclusion

- E-GSM allows monitoring processes with respect to both the control flow and the outcome of each activity
- Based on the violations that occur, it is possible to define metrics to assess how severely the process is affected
- It is possible to automatically translate BPMN process models into E-GSM:
  - BPMN is well-known, easy to use and to understand
  - No need to redesign existing processes from scratch
- The translation can be furtherly improved by considering the type of activities and events, and the associated data objects
- Metrics can be improved by associating weights to each activity

Thanks for your attention

Any question?

A POSTER ON THIS APPROACH WILL BE ALSO PRESENTED AT CAISE FORUM 2016

THIS WORK HAS BEEN PARTIALLY FUNDED BY THE ITALIAN PROJECT ITS2020 UNDER THE TECHNOLOGICAL NATIONAL CLUSTERS PROGRAM

TRASPORTI ITALIA 2020

UNIONE EUROPEA
Fondo europeo di sviluppo regionale

PON Ricerca e Competitività 2007-2013

Ministero dell'Istruzione, dell'Università e della Ricerca