A formal approach to verifying BPMN collaborations

The BProVe way

Francesco Tiezzi

Università degli Studi di Firenze, Italy

Guest Lecture of System Integration
Technical University of Denmark, April 24th, 2024

Goal and Motivations

Goal of the lecture

In previous lectures, you have seen:

GOAL OF THE LECTURE

In previous lectures, you have seen:

- **Formal models**, to specify systems behaviour

Goal of the lecture

In previous lectures, you have seen:

- **Formal models**, to specify systems behaviour
- Techniques and tools, to **verify properties** over models

GOAL OF THE LECTURE

In previous lectures, you have seen:

- **Formal models**, to specify systems behaviour
- Techniques and tools, to **verify properties** over models
- **BPMN models**, to specify Business Processes (BP)

Goal of the lecture

In previous lectures, you have seen:

- **Formal models**, to specify systems behaviour
- Techniques and tools, to **verify properties** over models
- **BPMN models**, to specify Business Processes (BP)

In this lectures, you will see:

Verification of BP-relevant properties over BPMN collaboration models

Motivations

Why do we need to verify Business Processes?

- Nowadays, organizations recognized the importance of having tools to model the different and interrelated aspects governing their behavior
- BPMN collaboration models, in particular, have acquired increasing relevance for this task
    - also in software development, since they shorten the communication gap between domain experts and IT specialists and permit clarifying the characteristics of software systems needed to provide automatic support for the organization activities
- There is the need of precisely checking the satisfaction of behavioral properties, to enhance the quality of the BP and the related software
- Hence, effective formal verification capabilities can foster the full adoption of the BPMN standard

The Ingredients

The “recipe” for formal verification requires the following ingredients:

- **BPMN notation**, for modelling the system to be analysed
- **Formal semantics**, associating to each BPMN model a corresponding formal model (LTS in our case)
- **BP-relevant properties**, defined on top of BPMN models and mapped to logical formuale (LTL in our case)
- **Verification tool** (MC in our case), automating the verification of the formal property over the formal model

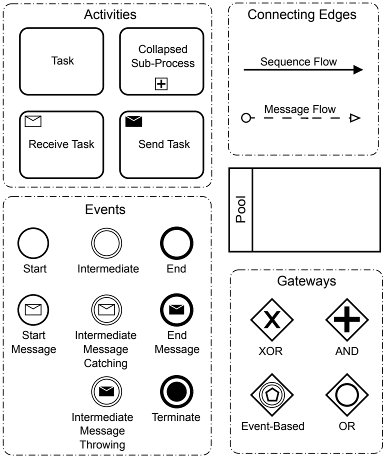
The BPMN notation

BPMN

- BPMN is an OMG standard (version 2.0 is released on 2011)
- BPMN is currently acquiring a clear predominance among the proposed notations to model BP thanks to:
  - its intuitive and graphical notation that is widely accepted by the industry and the academia
  - the support provided by a broad spectrum of modeling and enacting tools
- You already know the BPMN notation, here we just clarify the **elements supported by the BProVe approach**:
  - we selected a subset of BPMN elements following the pragmatic approach of retaining those features most used in practice[1]

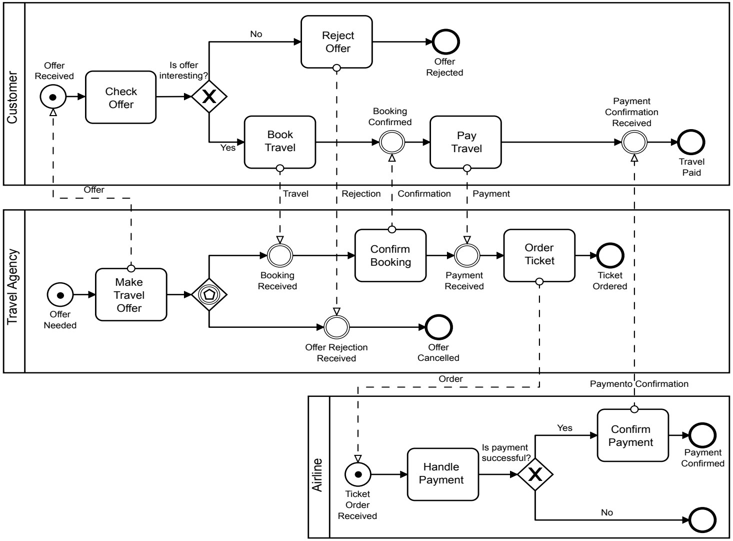
[1] Muehlen, M.Z., Recker, J., 2008. How much language is enough? Theoretical and practical use of the business process modeling notation. In: Advanced Information Systems Engineering. In: LNCS, vol. 5074, Springer, pp. 465–479

BPMN collaboration diagrams
≫Considered elements



We support models with an **arbitrary topology**, i.e. we do not impose any syntactical restriction on the usage of the modeling notation (such as well-structuredness)

BPMN collaboration diagrams
≫Travel agency example



BPMN collaboration diagrams
≫Informal semantics

- The execution of BPMN models is based on the notion of **tokens**, graphically denoted as black dots labeling BPMN elements
- In the example, the presence of such tokens over the start events enables the execution of the three processes (representing the collaboration's initial status)
- Tokens traverse the sequence edges of processes and pass through their elements enabling their execution
- The notation element's specific characteristics define the rules to follow to move, consume and generate tokens

BPMN formalization

Formalization

- **Goal**: providing BPMN with a **structural operational semantics (SOS)**
  - *SOS style*: we define the behavior of a model in terms of the behavior of its elements, in a syntax-oriented and inductive way
  - The behavior is formalized as a *Labeled Transition System* (LTS)
- **Issue**: The syntax of BPMN 2.0 is given in the standard document by a metamodel in classical UML-style
  - Such syntax is not suitable for providing an operational semantics
  - Thus, we provide an alternative **syntax in BNF-style**

BNF syntax is defined by grammar productions $N ::= A_1 \mid \ldots \mid A_n$ where N is a non-terminal symbol and alternatives $A_1, \ldots, A_n$ are compositions of terminal and non-terminal symbols

BNF Syntax

The BPMN formal syntax is defined by a grammar whose

- *non-terminal* symbols, C and P, represent *Collaborations* and *Processes*
- *terminal* symbols are the typical graphical elements of a BPMN model, i.e. pools, events, tasks, gateways, and edges

To obtain a **compositional** definition, each (message/sequence) edge is divided in two parts:

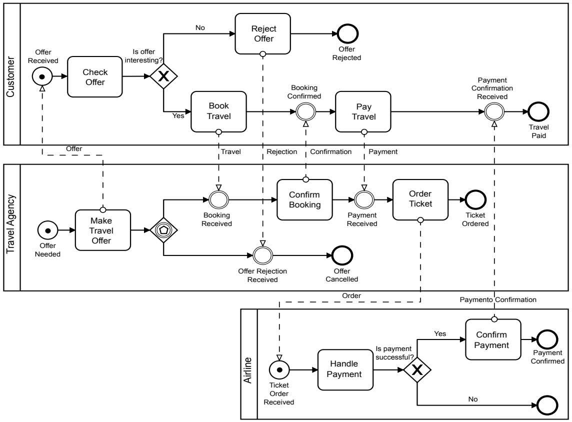
- the part outgoing from the source element
- the part incoming into the target element

A term of the syntax can be straightforwardly obtained from a BPMN model by decomposing:

- the collaboration in a collection of pools
- processes in collection of nodes
- edges in two parts

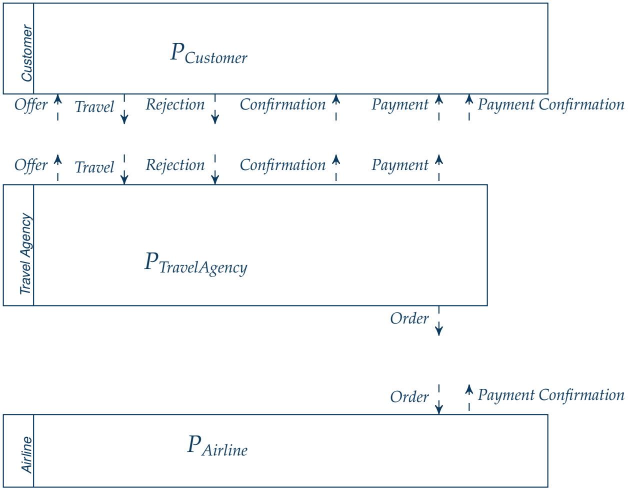
BNF Syntax
≫Model decomposition

Our travel agency BPMN collaboration model...



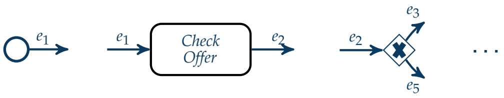
BNF Syntax
≫Model decomposition

...is decomposed as



BNF Syntax
≫Model decomposition

...where (an excerpt of) process $P_{Customer}$ is defined as follows:

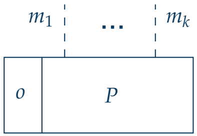


and processes $P_{TravelAgency}$ and $P_{Airline}$ are defined in a similar way

BNF Syntax
≫Grammar

A BPMN collaboration is rendered as a collection of pools where message edges can connect different pools and each pool contains a process

$(Collaborations) \quad C \quad ::=$



$(Pool,\ k \geqslant 0)$

$(Pool\ collection) \quad | \quad C_1 \quad C_2$

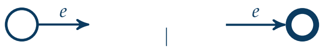
Names: organizations (o), messages (m), sequence edges (e), and tasks (t). We use edges of the form $\overset{m}{\text{- - - - -}}$ to denote message edges of the form $\overset{m}{\dashrightarrow}$ either incoming into or outgoing from pools/nodes. We identify *collaborations* (resp. *processes*) up to commutativity and associativity of *pool* (resp. *node*) *collection*

BNF Syntax
≫Grammar (an excerpt)

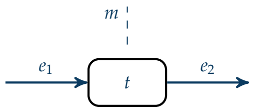
A BPMN process is rendered as a collection of nodes (events, tasks and gateways), each one with incoming and/or outgoing sequence edges

(Processes) P $::=$

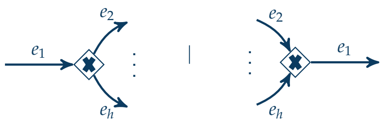
(Start/End event)



(Sending/receiving message task) $|$



(Split/join XOR gateway, $h > 2$) $|$



$\ldots$

(Node collection) $|$ $P_1 \quad P_2$

Operational semantics

The semantics of BPMN collaborations requires at the same time:

- dealing with **workflow aspects** requires synchronising the process elements in order to model the internal execution flow of an organisation
- dealing with **collaborative aspects** requires managing asynchronous message exchanges between different organisations
- dealing with **killing effects** of the terminate end event, which must be confined within a single process (not propagated to other organisations)

To overcome these issues we defined the operational semantics by separating process and collaboration layers

- **process layer** deals with internal interactions among process elements, including the effects of the terminate end event
- **collaboration layer** only focusses on inter-communication between organisations

Operational semantics

We give the semantics of BPMN in terms of **marked collaborations**, i.e. collaboration enriched with message edges, sequence edges, events and tasks marked by (possibly multiple) tokens

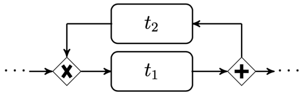
- A **marking** is a distribution of **tokens** over pool message edges and process elements
- This resembles the notions of token and marking in Petri Nets
- Tokens move along the syntax constructs, acting as program counters
- A single token is denoted by $\bullet$, while multiple tokens labelling a message edge m (resp. sequence edge e) are denoted by $m.n$ (resp. $e.n$), where $n \in \mathbb{N}$ is the token multiplicity

OPERATIONAL SEMANTICS

We give the semantics of BPMN in terms of **marked collaborations**, i.e. collaboration enriched with message edges, sequence edges, events and tasks marked by (possibly multiple) tokens

- A **marking** is a distribution of **tokens** over pool message edges and process elements
- This resembles the notions of token and marking in Petri Nets
- Tokens move along the syntax constructs, acting as program counters
- A single token is denoted by $\bullet$, while multiple tokens labelling a message edge m (resp. sequence edge e) are denoted by $m.n$ (resp. $e.n$), where $n \in \mathbb{N}$ is the token multiplicity

The use of tokens with multiplicity is due to the arbitrary topology of processes



Operational semantics
≫Process layer

The labeled transition relation of the LTS defining the **semantics of marked processes** is induced by a set of inference rules

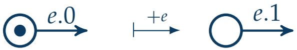
- Transitions have the form $P \overset{\alpha}{\longmapsto} P'$
- The labels are generated by:

$$
\begin{array}{lllllllll}
(\mathit{Actions}) & \alpha & ::= & \tau & | & !m & | & ?m & \\
(\mathit{Internal\ actions}) & \tau & ::= & \mathit{running}\ t \mid & & \mathit{completed}\ t \mid & & (-\tilde{e_1}, +\tilde{e_2}) \mid & \mathit{kill}
\end{array}
$$

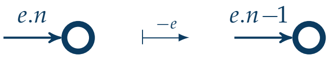
where $(-\tilde{e_1}, +\tilde{e_2})$ denotes movement of workflow tokens in the process

Operational semantics
≫Process layer

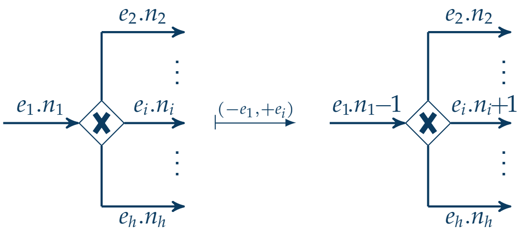
Some operational rules:



(*E-Start*)

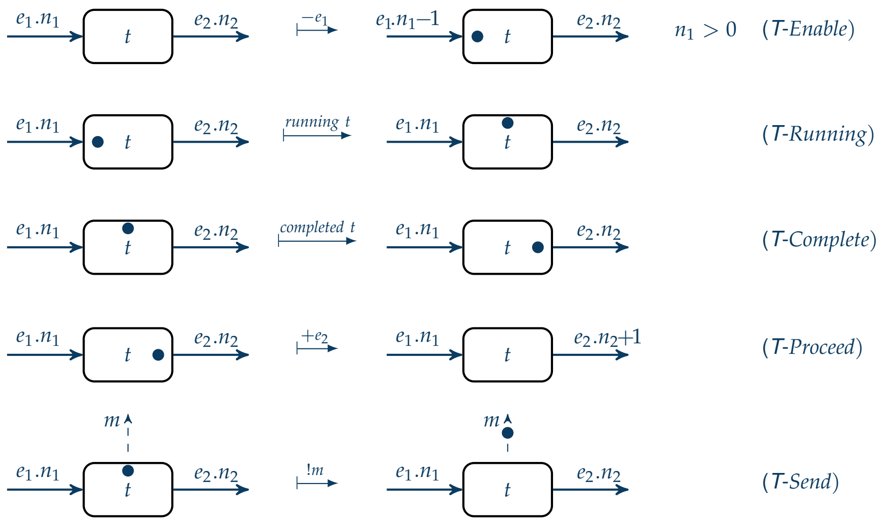


$n > 0$ (*E-End*)



$n_1 > 0$
$2 \leq i \leq h$ (*G-XorSplit*)

Operational semantics
≫Process layer


$e_1.n_1$ t $e_2.n_2$ $\xmapsto{-e_1}$ $e_1.n_1{-}1$ t $e_2.n_2$ $n_1 > 0$ (*T-Enable*)
$e_1.n_1$ t $e_2.n_2$ $\xmapsto{running\ t}$ $e_1.n_1$ t $e_2.n_2$ (*T-Running*)
$e_1.n_1$ t $e_2.n_2$ $\xmapsto{completed\ t}$ $e_1.n_1$ t $e_2.n_2$ (*T-Complete*)
$e_1.n_1$ t $e_2.n_2$ $\xmapsto{+e_2}$ $e_1.n_1$ t $e_2.n_2{+}1$ (*T-Proceed*)
m $e_1.n_1$ t $e_2.n_2$ $\xmapsto{!m}$ m $e_1.n_1$ t $e_2.n_2$ (*T-Send*)

Operational semantics
≫Process layer

$$\frac{P_1 \xmapsto{(-\tilde{e_1},+\tilde{e_2})} P_1'}{P_1 \quad P_2 \xmapsto{(-\tilde{e_1},+\tilde{e_2})} P_1' \quad P_2 \cdot (-\tilde{e_1},+\tilde{e_2})} \quad (\textit{N-MarkingUpd})$$

$$\frac{P_1 \xmapsto{\mathit{kill}} P_1'}{P_1 \quad P_2 \xmapsto{\mathit{kill}} P_1' \quad P_2 \dagger} \quad (\textit{N-Kill})$$

$$\frac{P_1 \xmapsto{\alpha} P_1' \qquad \alpha \notin \{(-\tilde{e_1},+\tilde{e_2}), \mathit{kill}\}}{P_1 \quad P_2 \xmapsto{\alpha} P_1' \quad P_2} \quad (\textit{N-Interleaving})$$

The *marking updating function* $P \cdot (-\tilde{e_1},+\tilde{e_2})$ returns a process obtained from P by unmarking (resp. marking) edges in $\tilde{e_1}$ (resp. $\tilde{e_2}$). The *killing function* $P\dagger$ returns a process obtained from P by completely unmarking it.

Operational semantics
≫Collaboration layer

The labeled transition relation of the LTS defining the **semantics of marked collaborations** is induced by a set of inference rules

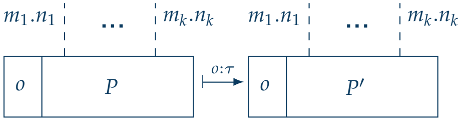
- Transitions have the form $C \xmapsto{l} C'$
- The labels are generated by:

$$(Labels) \qquad l \ ::= \ o : \tau \quad | \quad o :?m \quad | \quad o_1 \to o_2 : m$$

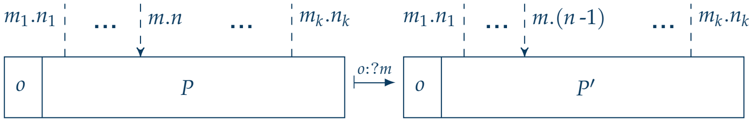
where internal/receiving/sending actions take place within/between organizations

Operational semantics
≫Collaboration layer

$$\frac{P \xmapsto{\tau} P'}{\quad}\ (C\text{-}Internal)$$

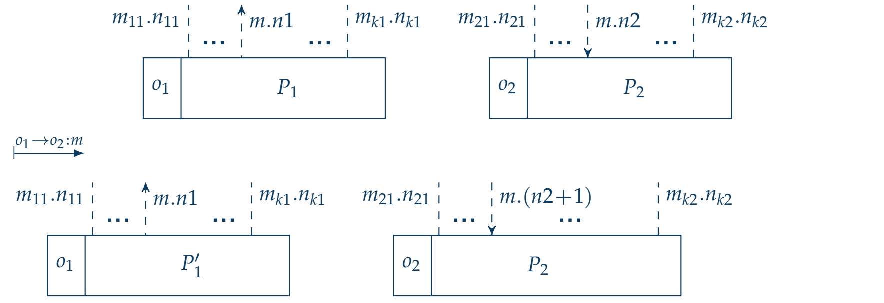


$$\frac{P \xmapsto{?m} P' \qquad n > 0}{\quad}\ (C\text{-}Receive)$$



Operational semantics
≫Collaboration layer

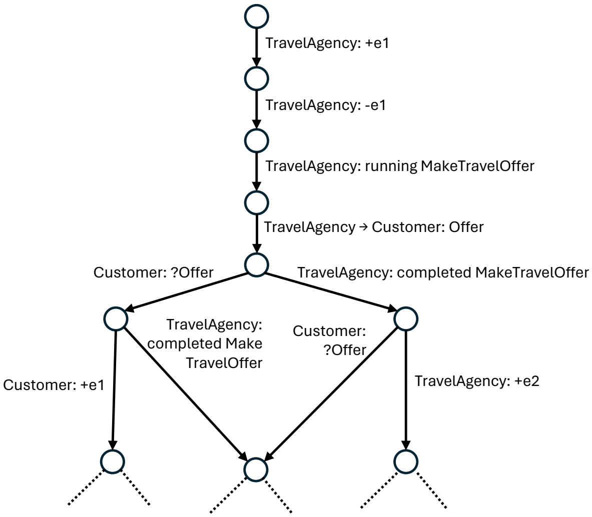
$$\frac{P_1 \xmapsto{!m} P_1'}{\quad} \qquad (\textit{C-Deliver})$$



$m_{11}.n_{11}$ ... $m.n1$ ... $m_{k1}.n_{k1}$
o_1 P_1
$m_{21}.n_{21}$ ... $m.n2$ ... $m_{k2}.n_{k2}$
o_2 P_2
$\xmapsto{o_1 \rightarrow o_2 : m}$
$m_{11}.n_{11}$ ... $m.n1$ ... $m_{k1}.n_{k1}$
o_1 P_1'
$m_{21}.n_{21}$ ... $m.(n2+1)$ ... $m_{k2}.n_{k2}$
o_2 P_2

$$\frac{C_1 \xmapsto{l} C_1'}{C_1 \quad C_2 \xmapsto{l} C_1' \quad C_2} \qquad (\textit{C-Interleaving})$$

Operational semantics
≫Travel agency example: LTS


TravelAgency: +e1
TravelAgency: -e1
TravelAgency: running MakeTravelOffer
TravelAgency → Customer: Offer
Customer: ?Offer
TravelAgency: completed MakeTravelOffer
TravelAgency: completed Make TravelOffer
Customer: ?Offer
Customer: +e1
TravelAgency: +e2

BP-relevant properties

PROPERTIES

- There is a rich literature about properties of process models
- We consider both the internal characteristics of a single process in a collaboration and the whole collaboration
- We refer to well-established in the business process domain: **safeness** and **soundness**
- We also support **ad-hoc properties** specifically defined for given application scenarios

Properties
≫Safeness

A BPMN process model is **safe** if during its execution no more than one token occurs along the same sequence edge

- The satisfaction of this property is generally considered a minimum guarantee to avoid unexpected behaviours
- This definition is inspired by the Petri Net formalism, where safeness means that a Petri Net does not have more than one token at each place in all reachable markings

Safeness of processes scales to collaborations, saying that no more than one token occurs on the same sequence edge during a collaboration execution

Properties
≫Soundness

Soundness can be described as the combination of three basic properties concerning the behaviour of a BPMN process model:

(i) *Option to Complete*: the running process must eventually complete

(ii) *Proper Completion*: at the moment of completion, each token of the process must be in a different end event

(iii) *No dead activities*: any activity can be executed in at least one process execution

Soundness is extended to process collaborations, involving the whole collaboration execution and requiring that all sent messages are properly received

Properties
≫Ad-hoc properties

Besides system-independent properties, we also consider **application-dependent properties** specifically defined for the given application scenario

Examples concerning the travel agency scenario.

(P1) Does the start of the *Confirm Booking* task in the *TravelAgency* pool implies that the same task will sooner or later complete?

(P2) Does the completion of a specified task in the *Airline* pool, say *HandlePayment*, implies the completion of another task in the same pool, say *ConfirmPayment*?

(P3) If the *Customer* has sent the payment to the *TravelAgency*, may it happen that the corresponding confirmation, by the *Airline*, is never received?

PROPERTIES
≫FORMALISATION

We formalize properties in terms of **Linear Temporal Logic (LTL)** formulae

Soundness and safeness properties in LTL:

Property	LTL formula
Soundness (i): Option to Complete	`[] processStart(orgName) -> <>processCompletion(orgName)`
Soundness (ii): Proper Completion	`[] ~ noProperCompletion(orgName)`
Soundness (iii): No Dead Activities	`[] ~ aTaskRunning(taskName)`
Safeness	`[] safeState(orgName)`

LTL operators:

- `[]` ϕ holds if ϕ *globally* holds;
- `<>` ϕ holds if ϕ *eventually* holds;
- ϕ `->` φ corresponds to the *implication*
- $\sim\phi$ corresponds to the *negation*

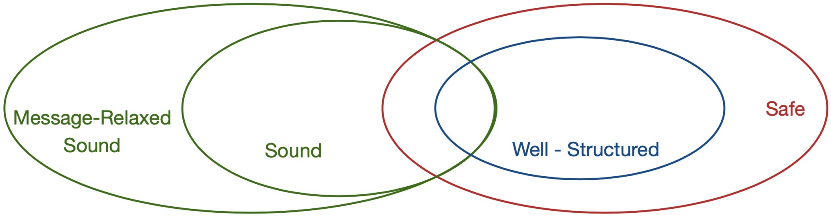
To ease the definition of properties we have defined some recurrent predicates representing high-level BPMN-related concepts

Using operators and predicates we can “cook” ad-hoc properties

PROPERTIES

≫CLASSIFICATION OF BPMN COLLABORATIONS

System-independent properties permitted to define this **classification of BPMN collaborations** [2]



[2] Corradini, F., Morichetta, A., Muzi, C., Re, B., Tiezzi, F. Well-structuredness, safeness and soundness: A formal classification of BPMN collaborations. Journal of Logical and Algebraic Methods in Programming 119, 2021

From theory to practice

MODEL CHECKING

Property Verification enables to check the properties property of interest detecting behavioural issues of the BPMN collaboration model

- This check, named **Model Checking** (MC) is based on:
  the model behaviour $\rightarrow$ its semantics $\rightarrow$ its LTS
- MC consists of systematically exploring the LTS to establish if a logical formula formally holds
- This exploration is error-prone and unfeasible with a manual approach
  an automated approach is needed $\rightarrow$ a **model checker tool**

MODEL CHECKING

Property Verification enables to check the properties property of interest detecting behavioural issues of the BPMN collaboration model

- This check, named **Model Checking** (MC) is based on:
  the model behaviour $\rightarrow$ its semantics $\rightarrow$ its LTS
- MC consists of systematically exploring the LTS to establish if a logical formula formally holds
- This exploration is error-prone and unfeasible with a manual approach
  an automated approach is needed $\rightarrow$ a **model checker tool**

To support property verification of BPMN collaborations we developed

BProVe

Towards automatic verification

To enable automatic verification of BPMN collaborations, as a first step we have **implemented the BPMN formal semantics** in the form of an executable interpreter in MAUDE

Why MAUDE as language for implementing our BPMN semantics?

- MAUDE permits to code rules of the operational semantics simply as rewriting rules with a one-to-one correspondence
- This ensures the faithfulness of the MAUDE implementation of the semantics with respect to its formal definition
- MAUDE comes equipped with a LTL model checker

TOWARDS AUTOMATIC VERIFICATION
≫BPMN SYNTAX IN MAUDE

Each BPMN element is declared as a MAUDE operation

- For example, the task element is rendered as:

```
op task(_,_,_,_) : Status Edge Edge TaskName -> ProcElement .
```

A sketch of the Travekl Agency collaboration is:

```
collaboration(
 pool( "Travel Agency" ,
  proc( start( enabled, "e1".0 ) |
   taskSnd ( disabled, "e1".0, "e2".0, "Offer".msg 0, "Make Travel Offer" ) |
   ...
  ) , in: ... , out: "Offer" .msg 0 ... ) |

 pool( "Customer" ,
  proc(  startRcv ( enabled , " e2 " . 0 , "Offer".msg 0) |
    ...
  ) , in: "Offer" .msg 0 andmsg IMsgSet , out: OMsgSet ) |

 pool( "Airline" ,  proc( ...) , in: ... , out: ... )
).
```

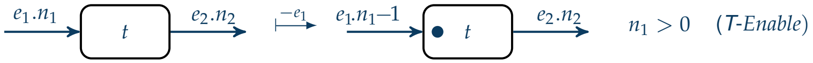
TOWARDS AUTOMATIC VERIFICATION
≫BPMN SEMANTICS IN MAUDE

Each rule of the BPMN semantics is rendered as a MAUDE rewriting rule

- Rewriting rules are exhaustively applied by pattern matching on each generated state, starting from the initial one, until no new states can be generated
- A rewriting rule has the following form:

```
crl [Label] : Term-1 => Term-2 if Condition(s) .
```

- Example: rule for enabling a task



```
crl [T-Enable] :
task( disabled , IEName . IEToken , OEName . OEToken , TName )
=>
{tUpd(IEName . IEToken , emptyEdgeSet)}
task( enabled , IEName . decreaseToken( IEToken ) , OEName . OEToken , TName )
if IEToken > 0 .
```

VERIFICATION

The verification featured by BProVe is based on the state-space exploration capabilities offered by our BPMN interpreter implemented in the MAUDE language

- Given a state of the collaboration under analysis, the interpreter permits to compute its set of one-step next states, i.e. all states reachable from the given state in one execution step, by applying the rewriting rules defining the BPMN semantics
- This allows the MAUDE interpreter to generate the LTS of the collaboration model
- The MAUDE LTL model checker derives and explores on-the-fly the state-space of the LTS

Verification
≫Statistical model checking

The MAUDE LTL model checker allows for an *exhaustive exploration* of the model reachable states and provides a reliable response in the case of systems with a finite state space

Verification
≫Statistical model checking

The MAUDE LTL model checker allows for an *exhaustive exploration* of the model reachable states and provides a reliable response in the case of systems with a finite state space

However, model checking techniques are known to be affected by the state-space explosion problem

- if a model generates many states, model checking cannot complete in a reasonable amount of time, or may fail due to high memory requirements
- the design of models with large state spaces is not uncommon in the context of business process modelling

VERIFICATION
≫STATISTICAL MODEL CHECKING

The MAUDE LTL model checker allows for an *exhaustive exploration* of the model reachable states and provides a reliable response in the case of systems with a finite state space

However, model checking techniques are known to be affected by the state-space explosion problem

- if a model generates many states, model checking cannot complete in a reasonable amount of time, or may fail due to high memory requirements
- the design of models with large state spaces is not uncommon in the context of business process modelling

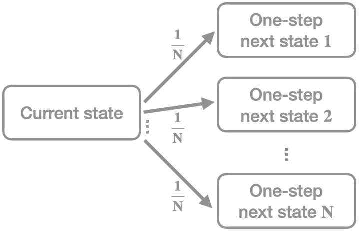
Therefore, to be able to provide a valid response in those cases, we resort to a simulation-based technique known as **statistical model checking** (SMC)

- SMC runs independent executions as long as a required level of statistical accuracy has not been reached, so to provide statistical evidence on the satisfaction or violation of a property

VERIFICATION
≫STATISTICAL MODEL CHECKING

The SMC technique we use is supported by MultiVeStA

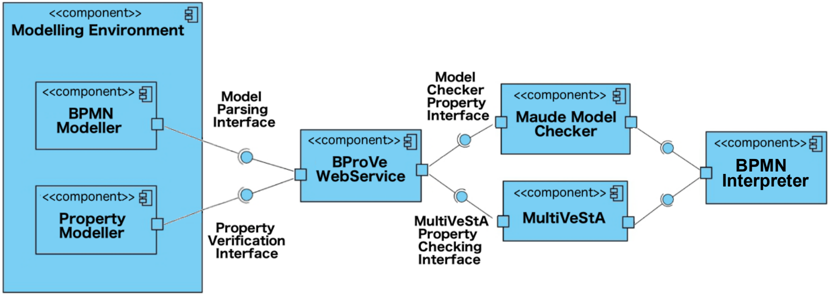
- It associates a discrete uniform probability distribution to the states that can be reached in one step from a specific state, allowing for probabilistic simulations



- This approach to path selection allows to introduce a form of fairness that is not considered when using the MAUDE LTL model checker
- The MultiVeStA's query specification language is MultiQuaTEx

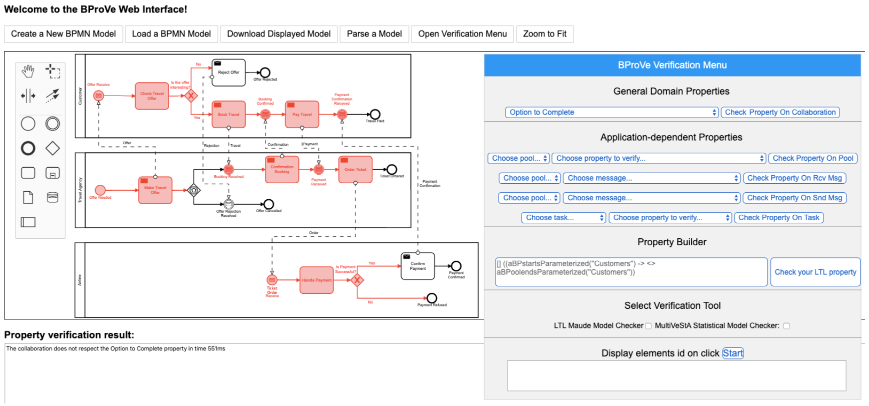
VERIFICATION
≫THE BPROVE TOOL

Component diagram of the BProVe tool-chain:



Verification
≫The BProVe tool

BProVe web interface:



Welcome to the BProVe Web Interface!

Create a New BPMN Model | Load a BPMN Model | Download Displayed Model | Parse a Model | Open Verification Menu | Zoom to Fit

BProVe Verification Menu

General Domain Properties

Option to Complete | Check Property On Collaboration

Application-dependent Properties

Choose pool... | Choose property to verify... | Check Property On Pool

Choose pool... | Choose message... | Check Property On Rcv Msg

Choose pool... | Choose message... | Check Property On Snd Msg

Choose task... | Choose property to verify... | Check Property On Task

Property Builder

[] ((aBPstartsParameterized("Customers") -> <> aBPoolendsParameterized("Customers"))

Check your LTL property

Select Verification Tool

LTL Maude Model Checker ☐ MultiVeStA Statistical Model Checker: ☐

Display elements id on click Start

Property verification result:

The collaboration does not respect the Option to Complete property in time 551ms

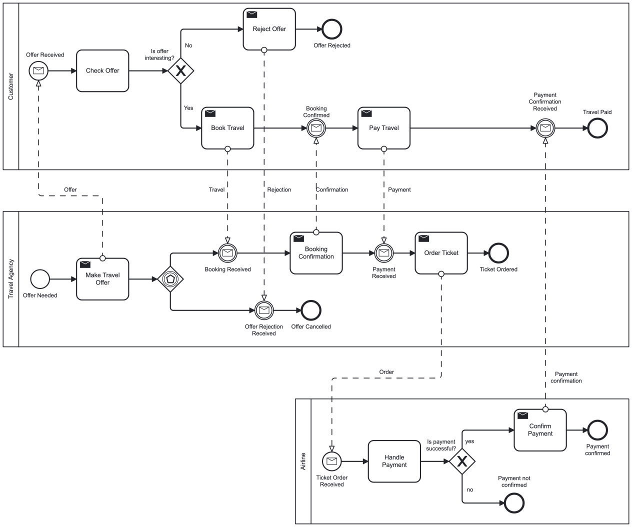
Demo

DEMO

- BProVe web app available at:
  `https://pros.unicam.it/bprove/bprove-web-interface`
- Local installation of BProVe via Docker:
  - Install and run Docker in your system:
    `https://www.docker.com/`
  - Download the BProVe docker image:
    ```
    docker pull proslab/bprove
    ```
  - Create a container:
    ```
    docker run -itd -p 8080:8080 --name bprove proslab/bprove
    ```
  - Run and stop a container:
    ```
    docker start bprove    /    docker stop bprove
    ```
  - Access the app via a browser:
    `http://localhost:8080/BProVe`

Demo

BPMN travel agency model:



Demo

Analysis of the Travel Agency scenario:

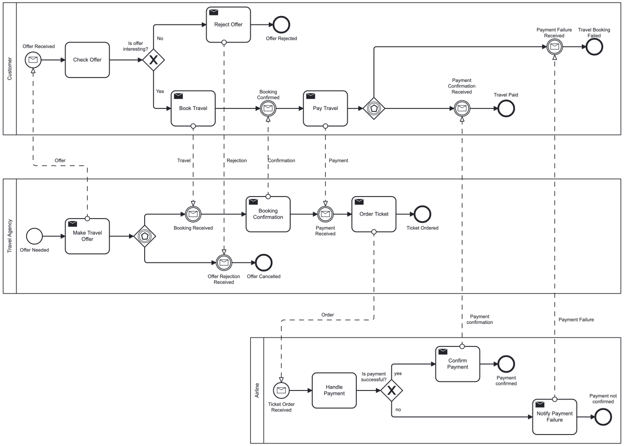
- **Safeness**: TRUE
- **Soundness**: FALSE
  - Option to Complete: FALSE
  - Proper Completion: TRUE
  - No Dead Activities: TRUE

Exercise 1

- Fix the Travel Agency model to make it sound

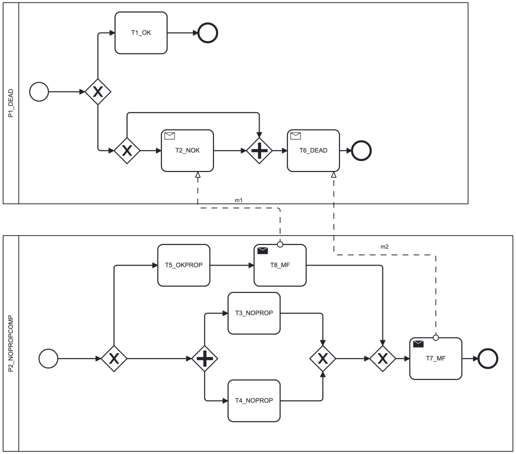
EXERCISE 1

- Fix the Travel Agency model to make it sound
- Solution:



Exercise 2

Consider the BPMN collaboration loaded at the start of BProVe:

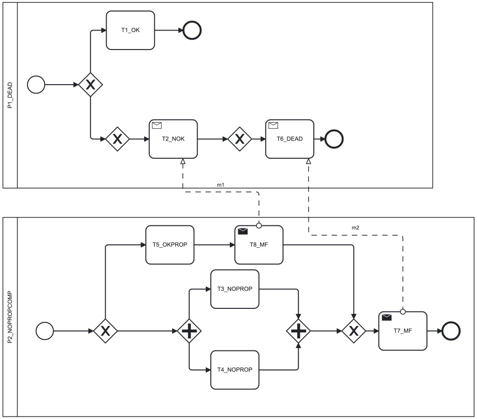


EXERCISE 2

- Check Safety and Soundness
- Fix the model to make it safe (hint: the designer used a wrong gateway)
- Fix the model to make it safe and without dead activities

Exercise 2

- Check Safety and Soundness
- Fix the model to make it safe (hint: the designer used a wrong gateway)
- Fix the model to make it safe and without dead activities
- Solution:



Thank you!

For further details about BProVe, visit
`https://pros.unicam.it/bprove/`