Computing and Informatics, Vol. 40, 2021, 1080–1107, doi: 10.31577/cai_2021_5_1080

FORMALIZATION AND MODEL CHECKING OF BPMN COLLABORATION DIAGRAMS WITH DD-LOTOS

Toufik Messaoud Maarouk, Mohammed El Habib Souidi
Nadia Hoggas

Faculty of Science and Technology, Université Abbès Laghrour Khenchela
ICOSI Lab, BP 1252 El Houria
40004 Khenchela, Algeria
e-mail: {maarouk.toufik, souidi.mohammed}@univ-khenchela.dz

Abstract. Business Process Model and Notation (BPMN) is a standard graphical notation for modeling complex business processes. Given the importance of business processes, the modeling analysis and validation stage for BPMN is essential. In recent years, BPMN notation has become a widespread practice in business process modeling because of these intuitive diagrams. BPMN diagrams are built from basic elements. The major challenge of BPMN diagrams is the lack of formal semantics, which leads to several interpretations of the concerned diagrams. Hence, this work aims to propose an approach for checking BPMN collaboration diagrams to guarantee some properties of smooth functioning of systems modeled by BPMN notation. The verification approach used in this work is based on model checking techniques. The approach proposes as a first step a formal semantics of the collaboration diagrams in terms of the formal language DD-LOTOS, i.e., a phase of the transformation of collaboration diagrams into DD-LOTOS. This transformation is guided by applying the inference rules of the formal semantics of the DD-LOTOS formal language, and we then use the UPPAAL model checker to check the absence of deadlock, safety properties, and liveness properties.

Keywords: BPMN, business process, DD-LOTOS, C-DATA, temporal logic, model checking

Mathematics Subject Classification 2010: 68-N30, 68-Q60

1 INTRODUCTION

In recent years, Business Process Model Notation (BPMN) [1] has been the main language for modeling business processes [2]. The modeling of business processes in BPMN is based on the use of graphic elements to build diagrams. These diagrams describe the behavior of the system to be modeled. BPMN 2.0 defines three diagrams to support all aspects of business processes. The elements of BPMN are tasks, subprocesses, events, sequences, and gateways [3].

The major problem with BPMN models is the lack of formal semantics to interpret BPMN diagrams. This renders the crucial step of analyzing and validating BPMN models impossible. Consequently, we cannot show that the system contains no design errors and therefore cannot ensure the smooth functioning of the model. This issue has opened an area of research, and several works have tried to provide a solution to this insufficiency.

In general, the proposed approaches are based on integrating formal methods into the design process for complex systems. Beginning with the expression of needs and continuing through the final production, thus making it possible to rigorously design the system and guarantee a smooth functioning and reliability of the developed system. The proposed approaches include transforming BPMN models to formal models such as Petri nets [4, 5, 6, 7, 8], process algebras [9, 10], and rewriting logic (Maude Language) [11, 12].

In our previous work [13], we proposed a formal semantics for BPMN models based on the formal language Distributed Durational LOTOS (DD-LOTOS). DD-LOTOS is a formal specification language that can model the behavior of concurrent systems, and it is defined with a semantic model of true concurrency [14]. This language allows distributed systems to be described with temporal constraints, and the specification is then translated into a semantic model called Communicating Durational Action Timed Automaton (C-DATA) [15], to be verified by formal verification tools.

In this paper, we focus on the formal verification of BPMN models. We propose a model verification approach (model checking) that consists of checking the properties of smooth functioning, such as the deadlock property, for the developed systems. These properties are expressed in Timed Computation Tree Logic (TCTL). The system is described by C-DATA. The C-DATA is defined on true concurrency semantics, which makes it possible to verify two categories of properties: qualitative and quantitative properties. A model checker is used to interpret the properties on the C-DATA semantic model. It returns either that the property is satisfied or not, including a counterexample in the latter case.

Figure 1 illustrates the proposed approach, which comprises two major steps. First, the system is developed using BPMN collaboration diagrams, and then, based on the collaboration diagrams, we generate a specification in DD-LOTOS code using the BPMN2DDLOTOS tool. At the end of this step, we generate the semantic model C-DATA. The second step consists of passing the C-DATA and the property

to be checked as inputs to the model checker. The latter returns Yes/No, indicating whether the property is satisfied by the C-DATA.

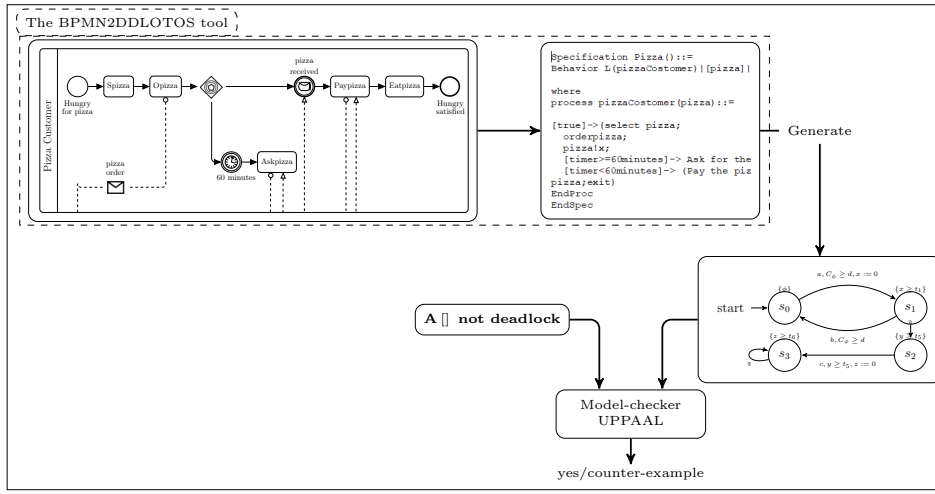

Figure 1. Architecture of the proposed approach

The first step, which consists of transforming the collaboration diagram into a DD-LOTOS specification, is automatically carried out using the BPMN2DDLOTOS tool.

This tool is developed by using the techniques and tools of model-driven engineering. At the beginning, we defined a meta-model for the BPMN collaboration diagram; we have also developed an editor to construct collaboration diagrams based on the defined meta-model. The editor offers a workspace that contains all the BPMN elements necessary for the construction of collaboration diagrams. The editor is developed using the Eclipse Sirius framework. Second, once the diagram is ready, the tool launches the transformation based on the meta-model and a set of transformation rules. These rules are defined from correspondence between BPMN elements and DD-LOTOS behavioral expressions. This step is carried out using the Eclipse Acceleo framework.

The second step concerns the generation of the corresponding C-DATA. This step is performed via the use of a generation tool [16]. The last step is to verify certain properties of smooth functioning by UPPAAL. The choice of UPPAAL compared to the existing model checkers such as SPIN, Tina, ... is motivated by the possibility to take into account the time aspect, either for the specification of temporal properties in particular the bounded liveness property or for the specification of the model to be checked.

The main contribution of our approach in comparison with the related works is to provide formal semantics to BPMN elements in terms of the formal language DD-

LOTOS. DD-LOTOS is an extension of the LOTOS standard; therefore, it inherits all the formal properties of LOTOS. It supports several aspects and characteristics of concurrent systems. In the context of business processes, DD-LOTOS can express the following aspects:

- Distribution of the processes on several sites or localities, this property is expressed explicitly in DD-LOTOS, which allows us to specify the swimlanes in BPMN.
- Time aspect: can be expressed by the temporal constraints through specific operators and the durations of actions through maximality semantics.
- Checking of quantitative and qualitative properties, as an example of quantitative properties, a task can start its execution only if the time elapsed since the sensitization of the system exceeds t units of time, or else a task is proposed to the environment for a determined duration.

The rest of this paper is organized as follows. Section 2 discusses related work. In Section 3, we introduce the basic concepts of BPMN diagrams, the formal language DD-LOTOS, and its semantic model, C-DATA. Section 4 presents our approach to generating C-DATA from BPMN diagrams. Section 5 presents and examines two real case studies. The paper ends with a conclusion and discussion of future work.

2 RELATED WORK

The formalization of BPMN models has been an active area of the research in recent years. Many works have proposed different approaches in order to verify and formally validate BPMN models. The main objective of formalization is to remedy the problem of the lack of formal semantics associated with BPMN models. This lack leads to problems of inconsistency, ambiguity and incompleteness in BPMN models. The majority of the works regarding this research field proposed approaches based on Petri nets, rewriting logic and Maude, and the theory of process algebras.

Corradini et al. [17, 12] proposed in the first work a formalization and formal verification of collaboration diagrams. For the formalization, a formal syntax of collaboration diagrams has been defined and an operational semantics represented by a set of inference rules describing the behavior of a collaboration. The authors proposed to take into account three properties: well-structuredness, safeness, and soundness. These properties are defined formally and studied rigorously within collaboration; several theorems show that the relationship between the three properties have been proven. Finally, the three properties have been checked using the $\mathcal{S}^3$ tool [18]. In the second work, the authors proposed an approach for checking collaboration diagrams known as BProVe. This approach is based on two model checking methods, namely Maude LTL model checker and statistical model checking, through the statistical analyzer MultiVeStA. The formal verification of the properties is effectuated in two stages. The first stage concerns each

process. However, the second one concerns the entire collaboration. For each process, the authors checked two important properties, soundness, and safeness. The soundness property expresses three sub-properties: option complete, proper completion and no dead activities. To analyze the collaboration, the authors used the statistical analyzer MultiVeStA by executing a finite number of finite executions.

Krishna et al. [19] proposed a transformation approach defined on a variety of formal behavioral relations between process models. The approach begins by transforming BPMN processes to a formal labeled transition system. This model is used as an input to the Construction and Analysis of Distributed Processes (CADP) toolbox to verify smooth functioning properties, such as the absence of deadlock.

Kheldoun et al. [20] also proposed an approach for verifying BPMN models. The general idea is to take into account the characteristics of the BPMN models, the cancellation and parallelism of the subprocesses, and handling exceptions. The proposed approach involves three transformations: the first transforms the BPMN model to the XMI format. The second transforms the latter to the recursive Petri nets RECATNets. Finally, the last transforms the RECATNets into a rewriting logic. Then a formal verification step is applied using the Maude LTL model checker tool.

Takemura [4] proposed a transformation correspondence from BPMN diagrams to formal Petri nets models. They show how to use the two properties of coverability and accessibility for formal verification of the BPMN transaction.

Dijkman et al. [5] proposed a mapping from BPMN to a formal Petri nets model. The choice of Petri nets was motivated by the available variety of efficient and reliable static analysis techniques. The properties checked for are the absence of dead tasks and proper task completion.

Martens and Cerioli [6] proposed a transformation from the Business Process Execution Language for Web Services (BPEL4WS) to Petri nets. Furthermore, they checked some proprieties such as the usability property or soundness criterion, the compatibility property, and the equivalence property.

Raedts et al. [7] proposed a transformation of BPMN models to Petri nets and then transformed the latter into a formal language called mCRL. Validation and verification are applied to the Petri nets, and the Yaser simulation tool is used to validate the BPMN model.

Decker and Weske [8] proposed "interaction modeling" as an alternative that avoids many disadvantages of other approaches, with a new extension of BPMN in terms of interaction modeling called *iBPMN*. This extension is then translated to a formal representation of Petri nets, called *interaction Petri nets*.

Cheikhrouhou et al. [21] proposed a BPMN 2.0 compliant extension to take into account the majority of temporal constraints. This extension supports three categories of constraints: intra-activity constraints, inter-activity constraints, inter-process constraints, and temporal constraints correlated with resource/data constraints. For the formal verification part, the authors used the UPPAAL model

checker and checked the absence of deadlock and the respect of deadlines of the processes.

Mallek et al. [22] proposed an approach to specify and verify interoperability in a BPMN collaboration. The approach is based on mapping collaboration diagrams into timed PLC networks. Then the UPPAAL model checker is used to verify the interoperability requirements. These requirements are formalized using temporal logic TCTL. The main limitation of this approach is that the expressive power of the TCTL logic does not allow a precise formalization of many interoperability requirements. Moreover, the response to the requirement is not limited to a yes/no provided by the model checker.

El-Saber and Boronat [11] defined a formal semantics of BPMN models; this semantics is interpreted by the formal language Maude. The BPMN model is defined by a tuple comprising several elements such as flow objects, data objects, activities, events, gateways tasks, subprocesses, etc. The BPMN elements are represented by a formal syntax expressed as an algebraic expression in Maude. In the analysis, the first validated property is the soundness of well-formed BPMN models.

Wong and Gibbons [9, 10] proposed an abstract syntax using the Z notation and the operational semantics of CSP process algebras for a subset of BPMN. In the analysis, some timed and untimed properties are validated, such as untimed invariance, responsiveness, preservation of freedom from deadlock, etc.

Güdemann et al. [23] proposed VerChor for development design and verification of choreographies, an intermediate form of choreography for the model checking, and a library of formal properties that must be checked to ensure the smooth functioning of the specification.

In general, most of the works in the literature proposed approaches that allow the interpretation of BPMN collaboration diagrams as a formal specification model. These models are generally Petri nets and their extensions, rewriting logic, mainly Maude language, and finally process algebras such as CSP. Güdemann's work made the exception by providing the VerChor environment to design and analyze choreographies diagrams.

The advantage of the approaches based on Petri nets is related to the tools part. In other words, there exist a variety of manipulation, simulation, and analysis tools intended for Petri nets. On the other hand, Petri nets suffer from a significant limitation: they cannot support two essential concepts of business processes. The first is the cancellation of an activity or a task, and the second is the OR-joins operator. These two elements pose correctness problems in business processes [24, 25]. To remedy this challenge, Yet Another Workflow Language (YAWL) [24] has been proposed. It has been shown that YAWL is more suited to business processes than Petri nets [24].

In this context, Wang et al. [25] proposed an approach to verify business processes with cancellation and OR-joins; these processes are modeled using YAWL. Four important properties are checked: soundness, weak soundness, irreducible cancellation regions, and immutable OR-joins. The authors confirm that this result can be applied to the BPMN notation. In another work, Wang et al. [26] have de-

fined a set of reduction rules for preserving soundness in the modeling of business processes with cancellation and OR-joins.

Our approach defines an interpretation of BPMN collaboration diagrams in the formal DD-LOTOS language, in which we can specify the temporal constraints and the duration of actions. The main contribution in this context is to check two categories of properties: quantitative properties and qualitative properties. The first category is related to the temporal constraints, the deadlines of the offer and the duration of actions, while the second is related to properties of smooth functioning such as deadlock, accessibility, and starvation.

In the approaches defined on interleaving semantics such as Petri nets, CSP, and CCS process algebras and their temporal extensions, we cannot distinguish between the sequential and parallel activities. This limit is due to the assumption of the structural and temporal atomicity of the actions imposed by the interleaving semantics. It is, therefore, not possible to check the properties relating to the duration of actions and parallel execution.

Our DD-LOTOS language is defined on another semantic model known as true concurrency semantics. In this semantics, we can distinguish between sequential behavior and parallel behavior.

The other contribution of our approach is that the DD-LOTOS language and its semantic model C-DATA take into account the distribution of activities and processes, which makes it able to specify BPMN swimlanes.

3 PRELIMINARIES

3.1 BPMN

The Object Management Group (OMG) [1] developed the standard BPMN language. The BPMN language is intended for the modeling of business processes. BPMN modeling relies on a set of diagrams. These diagrams are made up of several simple elements that describe the activities performed by the business process. However, BPMN does not have formal behavioral semantics.

Table 1 provides the essential BPMN elements for business process modeling, such as tasks, sub-process, gateways, and events.
The basic categories of elements are:

- Flow objects, such as *tasks*, *events*, and *gateways*;
- Connecting objects, such as *sequence flows*, *message flows*, and *associations*;
- Swimlanes, such as *pools* and *lanes*;
- Artifacts, such as *data objects*, *text annotations*, and *groups*.

BPMN defines the organization of these elements through either orchestration (which happens within processes) or choreography (communications between processes). BPMN models have three elementary types of sub-models [1]:

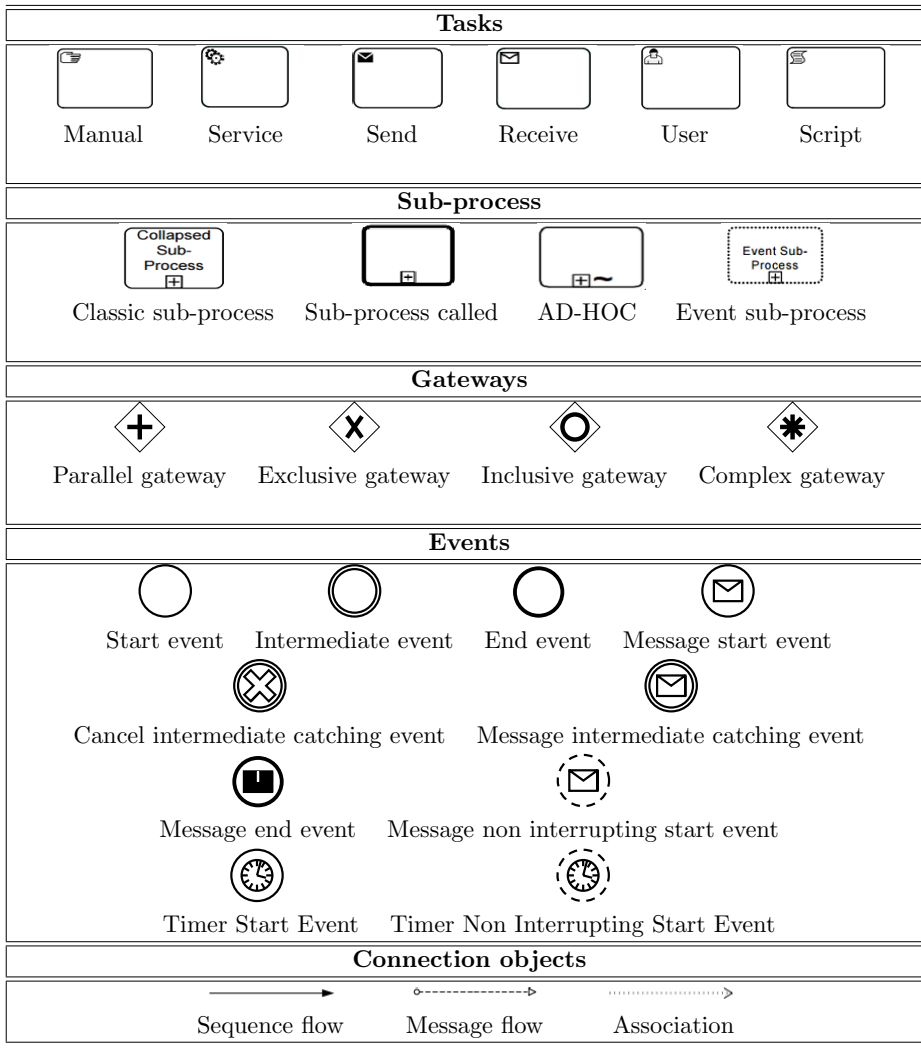

Table 1. Basic BPMN elements

1. Orchestration,
2. Choreographies,
3. Collaborations.

The collaboration diagram shows the relationship and interactions between two or more participants, and it describes a global view. Pools represent the participants in the collaboration, and the messages exchanged are represented by message flows.

In a choreography diagram, we are not interested in the activities between the participants, but the emphasis will be on the messages exchanged and their logical relation. On the other hand, the orchestration diagram formalizes a central process that controls and coordinates all the participants.

3.2 The Distributed D-LOTOS Language

DD-LOTOS [15] is a formal language that extends the D-LOTOS [27] language. D-LOTOS for durational LOTOS is a formal language defined on maximality semantics to explicitly consider the duration of action. It keeps the same syntax as LOTOS, but the semantic model escapes the hypothesis of structural and temporal atomicity. The actions are not atomic and of execution time different from null.

The DD-LOTOS extension takes into account the following aspects:

- Parallel and distributed behaviors in distributed computing;
- Communication and synchronization between processes through the message exchange paradigm;
- True concurrency semantics;
- Temporal constraints.

The DD-LOTOS syntax is defined in Table 2.

$E ::=$	**Behaviors**
	$stop \mid exit\{d\} \mid \Delta^d E \mid X[L] \mid$
	$g@t[SP]; E \mid i@t\{d\}; E \mid hide\ L\ in\ E \mid$
	$E[]E \mid E \mid [L] \mid E \mid E \gg E \mid E[> E \mid$
	$a!v\{d\}; E \mid$
	$a?xE$
$S ::=$	**Systems**
	$\phi \mid S \mid S \mid l(E)$

Table 2. Syntax of DD-LOTOS

The expression $a\{d\}$ represents a temporal restriction. The expression $\Delta^d E$ represents the delay operator and means that the expression E starts after the flow of d. $g@t[SP];E$ is a behavioral expression, where t is a temporal variable, and SP is a logical predicate. $Hide\ L\ in\ E$ is an interiorization. $E[]E$ is a nondeterministic choice, $E|[L]|E$ is a parallel composition, $E \gg E$ is a sequential composition, and $E\,[>\,E$ is a preemption. The expression $a!v\{d\}; E$ expresses emission of the message v on the gate a; this emission is limited by the temporal interval $[0, d]$. The expression $a?xE$ expresses the receipt of a message on the gate a.

A system can be empty, a composition of two systems or a behavioral expression E defined in a locality l.

Definition 1. The actions in the global system are:

- Communication actions, these actions represent the exchange of messages between localities $Act_{com} ::= a!m \mid a?x \mid \tau$ (sending, receiving and the silent action);
- Set of observable actions, the silent action and the terminating action $Act = \mathcal{G} \cup \{i, \delta\}$.

Definition 2. The set $\mathcal{L}$ ranged over by l, denotes set of localities. ϑ an infinite set of channels defined by users ranged over by a, b, ... channels are used for communicating messages between localities.

3.2.1 Structured Operational Semantics

The operational semantics of DD-LOTOS extend that of D-LOTOS. In this context, we limit the proposed work to the rules used for sending and receiving messages, remote communication, and time progression:

Process. $a!v\{d\}; E$ In the configuration ${}_M[a!v\{d\}; E]$, the sending of the message v starts only if all the actions in the set M terminate. In rule 1 below, the condition $Wait(M) = false$, which means some actions have not completed their executions. Rules 2 and 3 express a passage of time. Rule 4 expresses that the sending action must respect the temporal restriction operator; otherwise, it is transformed to *Stop*.

1. $$\frac{\neg Wait(M)}{{}_M[a!v\{d\}; E] \xrightarrow{M^{a!v}x}_{\{x:a!v:t\}} [E]}\ x = get(\mathcal{M}),$$

2. $$\frac{Wait(M^{d'}) \text{ or } (\neg Wait(M^{d'}) \text{ and } \forall \varepsilon > 0. Wait(M^{d'-\varepsilon}))\ d' > 0}{{}_M[a!v\{d\}; E] \xrightarrow{d'}_{M^{d'}} [a!v\{d\}; E]},$$

3. $$\frac{\neg Wait(M)}{{}_M[a!v\{d' + d\}; E] \xrightarrow{d}_M [a!v\{d'\}; E]},$$

4. $$\frac{\neg Wait(M) \text{ and } d' > d}{{}_M[a!v\{d\}; E] \xrightarrow{d'}_M [stop]}.$$

Process. $a?xE$ As in the previous configuration ${}_M[a?xE]$, the reception begins once all actions in the set M complete their execution.

$$\frac{\neg Wait(M)}{{}_M[a?xE] \xrightarrow{M^{a?x}y}_{\{y:a?x:0\}} [E]}.$$

Remote communication. The sending and receiving of messages via the same communication gate consume the silent action, is expressed by the following rule:

$$\frac{-}{{}_M[l(a!v\{d\}; E1)]|_{M'}[k(a?xE2)] \xrightarrow{\tau}_M [l(E1)]|_{M'}[k(E2\{v/x\})]}.$$

Time evolution on system.

$$\frac{E \xrightarrow{d} E'}{l(E) \xrightarrow{d} l(E')},$$

$$\frac{S_1 \xrightarrow{d} S_1' \quad S_2 \xrightarrow{d} S_2'}{S_1 \mid S_2 \xrightarrow{d} S_1' \mid S_2'}.$$

3.2.2 Communicating Durational Action Timed Automaton (C-DATA)

C-DATA [15] is a semantic model that allows taking all of the aspects present in the DATA* (Durational Action Timed Automaton) into account, such as the temporal and structural nonatomicity of actions, urgency of the actions, deadlines, and temporal constraints. In C-DATA, each locality is represented by a DATA*. The global system is represented by the set of DATA* s locals, which communicate by exchanging messages through communication channels.

Definition 3. A Communicating *DATA* (C-DATA) $A(S, L_S, s_0, \vartheta, H, \Pi, T_D)$ is a subsystem in which

- S is a finite set of states,
- $L_S : S \to 2_{fn}^{\Phi_t(H)}$ is a function which associates with each state s the set F of ending conditions (duration conditions) of actions possibly in execution in s,
- $s_0 \in S$ is the initial state,
- ϑ is the alphabet of the channels on which messages flow between the subsystems,
- H is a finite set of clocks,
- $\Pi = Act_{com} \cup Act$ is the set of internal and communication actions of A, and
- $T_D \subseteq S \times 2_{fn}^{\Phi_t(H)} \times 2_{fn}^{\Phi_t(H)} \times \Pi \times H \times S$ is the set of transitions, where transition $(s, G, D, \alpha/(a(!/?)v)/i, z, s\prime)$ represents switching from state s to state $s\prime$, either by starting execution of action $\alpha \in Act$ or actions (Sending or Receiving) or synchronization for the accomplishment of communication (silent action) and updating clock z, G is the corresponding guard which must be satisfied to fire this transition, and D is the corresponding deadline which requires, at the moment of its satisfaction, that action α must occur. Note that $(s, G, D, \alpha/(a(!/?)v)/\tau, z, s\prime)$ can be written as $s \xrightarrow{G\ ,\ D\ ,\ alpha\ /\ (\ a\ (\ !/\ ?\ )v)\ /i\ ),\ z} s'$.

Definition 4.

System: A system of n C-*DATA*s is a tuple $S = (A_1, \ldots, A_n)$ in which $A_i = (S_i, L_{S_i}, s_{0_i}, \vartheta, H_i, \Pi_i, T_{iD})$ is a C-*DATA*.

States: $GS(S) = (s_1, v_1) \times \cdots \times (s_n, v_n) \times (\vartheta^*)^p$ is the set of states.

Initial state: The initial state of S is $q_0 = ((s_{01}, 0), \ldots, (s_{0n}, 0) : \epsilon_1, \ldots, \epsilon_p)$ such that ϵ is the empty word of the alphabet ϑ.

System states: Let $S = (A_1, \ldots, A_n)$ be a system of n C-*DATA*s, $A_i = (S_i, L_{S_i}, s_{0_i}, \vartheta, H_i, \Pi_i, T_{iD})$. A global state of S is defined by the state of each subsystem and the states of each channel, and a state of S is an element of $(s_1, v_1) \times \cdots \times (s_n, v_n) \times (\vartheta^*)^p$ such that $v_i(h)$ are valuations on H.

4 GENERATION OF C-DATA FROM BPMN DIAGRAMS

In [13], we have defined a formal semantics to interpret BPMN diagrams in terms of the formal language DD-LOTOS. For this purpose, we implemented a tool called BPMN2DDLOTOS for transforming BPMN elements into DD-LOTOS specifications.

The idea is to assign to each element of the collaboration diagram a behaviorally equivalent DD-LOTOS pseudo-code. As an example, the task of sending a message represented by the graphic element.

Figure 2. Message reception task

Is transformed by the behavioral expression DD-LOTOS:

```
a?x:message
```

Expresses the reception of a message through the communication channel a.

4.1 Operational Semantics of the C-DATA

With the aim of analyzing and verifying a DD-LOTOS specification by model checking tools, it must be transformed into a semantic model, hence we have defined the C-DATA model. We generate a C-DATA model from a DD-LOTOS specification. This generation is defined on the operational semantics of C-DATA, where the latter is defined in the form of a formal system of inference rules. The semantic generation rules are presented in Definition 5.

Definition 5. Let $S = (A_1, \ldots A_n)$ be a system of n C-*DATA*s such that each $A_i = (S_i, L_{S_i}, s_{0_i}, \vartheta, H_i, \Pi_i, T_{iD})$ $(1 \leq i \leq n)$. The transition T between the state $s = ((q_1, v_1), \ldots, (q_n, v_n) : x_1, \ldots, x_p)$ and the state $s\prime = ((q\prime_1, v\prime_1), \ldots, (q\prime_n, v\prime_n) : x\prime_1, \ldots, x\prime_p)$ can be either an emission (ER), a reception (RR), an execution of a local

action (LR), a passage of time (PR), or a synchronization by appointment (SR). The semantics of the system S is defined by the smallest transition relation meeting the following rules:

1. ER (emission rule)

$$\frac{((q_i, v_i), a!v, (q_i', v_i')) \in T_D \quad x_j : \text{word of the channel } a}{(\dots, (q_i, v_i), \cdots : \dots, x_j, \dots) \xrightarrow{a!v} (\dots, (q_i', v_i'), \cdots : \dots, x_j.v, \dots,)}.$$

2. RR (reception rule)

$$\frac{((q_i, v_i), a?x, (q_i', v_i')) \in T_D x_j : \text{word of the channel } a}{(\dots, (q_i, v_i), \dots : \dots, x_j.v, \dots) \xrightarrow{a?x} (\dots, (q_i', v_i'), \dots : \dots, x_j, \dots)}.$$

3. LR (execution of a local action)

$$\frac{((q_i, v_i), \alpha, G, D, z, (q_i', v_i')) \in T_D \quad v \models G}{(\dots, (q_i, v_i), \dots : \dots, x_i, \dots) \xrightarrow{\alpha} (\dots, (q_i', v_i'), \dots : \dots, x_i, \dots)}$$

   where G is a temporal constraint or *guard*, D is the corresponding deadline which requires at the time of its satisfaction that the action α must be activated, and z is a clock which must be reset.

4. PR (passage of time)

$$\frac{d \in R^+ \forall d' \leq d(v_i + d) \not\models D}{(\dots, (q_i, v_i), \dots : x_1, \dots, x_p) \xrightarrow{d} (\dots, (q_i, v_i + d), \dots : x_1, \dots, x_p)}.$$

5. SR (synchronization by appointment; for this rule, the channels x_i and x_j do not contain messages)

$$\frac{((q_i, v_i), a?x, (q_i', v_i')) \in T_D \quad \big((q_j, v_j), a!v, \big(q_j', v_j'\big)\big) \in T_D \quad i \neq j}{(\dots, (q_i, v_i), \dots, (q_j, v_j), \dots : x_1, \dots, x_p) \xrightarrow{\tau} \big(\dots, (q_i', v_i'), \dots \big(q_j', v_j'\big), \dots : x_1, \dots, x_p\big)}.$$

The two rules ER and RR express the exchange of messages between two localities; this communication is asynchronous.

The ER rule represents an emission of a message v on the communication channel a. The premise of the rule expresses the change of configuration in the locality concerned by the emission. The conclusion expresses the emission and storage of the message v in channel a, x_j represents the message already in channel a.

The RR rule expresses the reception of a message via the channel a.

The RR rule expresses the execution of an action in a specific locality, so the rule does not affect the rest of the system.

The PR rule expresses the time progression on the system, so all clocks are incremented by d units of time.

The SR rule expresses the synchronization between two processes in the same locality, so it is a local and synchronous communication. The first process waits for the reception of a message and the second sends the message v.

4.2 From DD-LOTOS to C-DATA

The C-DATA semantic model is defined from the interpretation of the DD-LOTOS specifications. To generate a C-DATA model from a DD-LOTOS specification, the rules of the operational semantics of C-DATA are applied [16].

The tool checks the lexicon and the syntax of the specification; for this, we have defined the grammar for DD-LOTOS. Once the specification is correct, the step of generating C-DATA begins. Algorithm 1 summarizes the idea of generation.

The algorithm takes as input a specification and returns three sets; the set of C-DATA configurations denoted $\mathcal{S}$, the set of transitions denoted $\mathcal{T}$, and the set of clocks denoted by $\mathcal{X}$. Each configuration is made up of all the actions being executed with their duration.

Algorithm 1 Generation of the C-DATA model

Procedure Generation_C-DATA { S: Spec }
Creat_initial_configuration $C0$;
$\mathcal{S} = \{C0\}$; ▷ Countable set of configurations
$\mathcal{T} = \phi$; ▷ Countable set of transitions
$\mathcal{X} = \phi$; ▷ Countable set of clocks
ForEach (P: Process) **Do {**
 ForEach (B: Behavior) **Do {**
  Extract_behavior(B);
  Choice (B) **In {**
   action: Treat_Action($\mathcal{S}, \mathcal{T}, \mathcal{X}$); ▷ Call a function
   predicate: Treat_Predicate($\mathcal{S}, \mathcal{T}, \mathcal{X}$);
   operators: Treat_Operator($\mathcal{S}, \mathcal{T}, \mathcal{X}$); ▷ operators in $\{[], [>, >>, |[L]|\}$
    send-receipt: Treat_Send-Receipt($\mathcal{S}, \mathcal{T}, \mathcal{X}$);
    delay: Treat_Delay($\mathcal{S}, \mathcal{T}, \mathcal{X}$);
    termination: Treat_Exit-stop($\mathcal{S}, \mathcal{T}, \mathcal{X}$);
   }
▷ The loop continues execution until there are no more behaviors in the process.
 }
 EndFor
 Extract_Next-process(P)
}
EndFor
EndProcedure

5 CASE STUDIES

We have evaluated the proposed approach with two case studies. The first case, ordering pizza, has already been treated in the context of our work [13]. The second case is a biometric passport request [28].

5.1 Ordering Pizza

Figure 3 shows a BPMN diagram describing the process of ordering pizza. Knowing that, we have only taken into consideration the customer process.

The customer begins his process by choosing a pizza (*Spizza*), then he launches his order (*Opizza*), which triggers sending of the preparation message ($p!x$). If the elapsed time exceeds 60 minutes, the customer requests his pizza again. If the elapsed time does not exceed 60 minutes, and the customer receives his pizza ($p?x$) then he has to pay (*Paypizza*), and he eats (*Eatpizza*), before the end of the process.

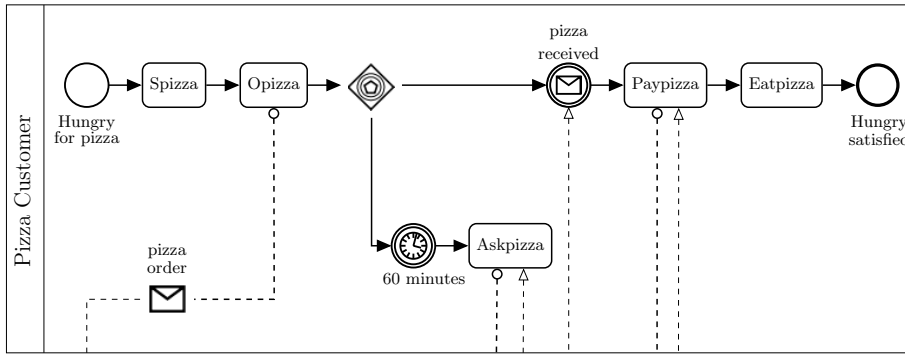

Figure 3. Ordering pizza

The DD-LOTOS specification generated by the BPMN2DDLOTOS tool is given as follows:

```
Specification Pizza[p]:=
Behavior L(pizzaCustomer);

where
process pizzaCustomer[p](timer):=

[true]->(Spizza;Opizza;p!x;
  (
  [timer>=60]-> Askpizza;
    []
  [timer<60]-> (p?x;Paypizza;Eatpizza;exit)
```

```
  ))
EndProc
EndSpec
```

The specification is composed of a single locality L, which contains the running *pizzaCustomer* process. The C-DATA model is generated as follows:

In the initial configuration, no action is being performed. This configuration is represented by the behavioral expression:

$$\underbrace{_{\phi}[pizzaCustomer]}_{c0}.$$

The first action to be performed in the configuration $c0$ is *Spizza*:

$$\underbrace{_{\phi}[pizzaCustomer]}_{c0} \xrightarrow{\phi, Spizza, x}_{c1} \underbrace{_{\{x \geq t_1\}}[P1]}.$$

```
P1:=(Opizza;p!x;
     (
      [timer>=60]-> Askpizza;
       []
      [timer<60]-> (p?x;Paypizza;Eatpizza;exit)
      )
     )
```

where t_1 is the time required for the execution of the *Spizza* action. *Timer* is a clock that calculates the time elapsed from the request for pizza until it is received. It is initialized to zero once the *Opizza* action is executed.

Given the semantics of the prefix operator ";", the *Opizza* action can only begin execution when the *Spizza* action ends:

$$c1 \xrightarrow{t} \underbrace{_{\phi}[P1]}_{c2}$$

where t represents a duration of time greater than t_1. Once the *Spizza* action ends, the *Opizza* action can begin execution in the interval $]t_1, +\infty[$. Based on configuration $c2$, the *Opizza* action can be executed:

$$c2 \xrightarrow{\phi, Opizza, y} \underbrace{_{\{y \geq t_2\}}[P2]}_{c3}.$$

where t_2 is the time required for the execution of the *Opizza* action.

```
P2:=(p!x;
     (
      [timer>=60]-> Askpizza;
```

```
 []
[timer<60]-> (p?x;Paypizza;Eatpizza;exit)
)
)
```

Similarly, the action of sending $p!x$ on the p channel is executed once the *Opizza* action is completed:

$$c3 \xrightarrow{t} \underbrace{\phi[P2]}_{c4},$$

$$c4 \xrightarrow{\phi, p!x, \{z, timer\}} \underbrace{_{\{z \geq t_3\}}[P3]}_{c5}.$$

where t_3 is the time required to complete the send pizza order $p!x$ action.

```
P3:=(
      [timer>=60]-> Askpizza;
       []
      [timer<60]-> (p?x;Paypizza;Eatpizza;exit)
      )
```

$$c5 \xrightarrow{t} \underbrace{\phi[P3]}_{c6}.$$

If the system has exceeded 60 units of time since the pizza was ordered, the customer still requests the pizza. In this case, the *Askpizza* action is offered to the environment:

$$c6 \xrightarrow{\phi, Askpizza, timer \geq 60, x} \underbrace{_{\{x \geq t_6\}}[P4]}_{c7}.$$

```
P4:= Stop
```

and the process stops without success.

In the second case, where the system has not exceeded 60 units of time since the pizza was ordered, the customer receives the pizza and then pays, so the receive action $p?x$ on the channel p is offered to the environment:

$$c6 \xrightarrow{\phi, p?x, timer < 60, x} \underbrace{_{\{x \geq t_4\}}[P5]}_{c8}$$

where t_4 is the time required for executing the action of receiving pizza $p?x$.

```
P5:=(Paypizza;Eatpizza;exit);
```

The customer pays for the pizza once the receiving action is complete, so the *Paypizza* action is offered to the environment:

$$c8 \xrightarrow{t} \underbrace{\phi[P5]}_{c9},$$

$$c9 \xrightarrow{\phi, Paypizza, x \geq t_5, y} \underbrace{_{\{x \geq t_5\}}[P6]}_{c10}$$

where t_5 is the time required for executing the action *Paypizza*.

```
P6:=(Eatpizza;exit);
```

$$c10 \xrightarrow{t} \underbrace{_{\phi}[P6]}_{c11},$$

$$c11 \xrightarrow{\phi, Eatpizza, y \geq t_6} \underbrace{_{\{y \geq t_6\}}[P7]}_{c12}$$

where t_6 is the time required for executing the action *Eatpizza*.

```
P7:=exit;
```

After executing the different steps detailed in this section, we can consider that the process is successfully completed. Figure 4 shows the C-DATA model generated from the initial specification.

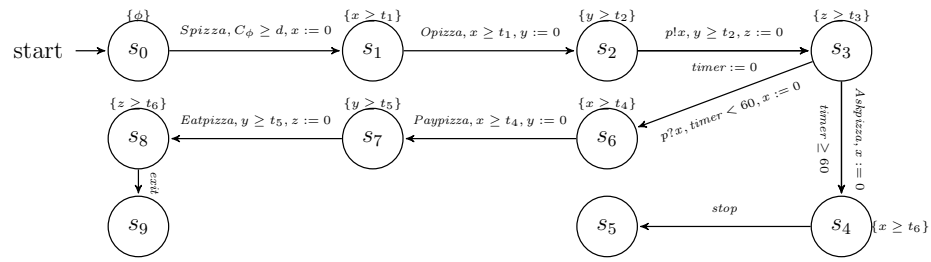

Figure 4. C-DATA: Ordering pizza

5.1.1 Model Checking

In this section, we propose checking of the following property classes:

1. Safety properties, which make it possible to verify that the system will never be in an undesirable state.
2. Liveness properties, which make it possible to verify that the system will reach a certain desirable state.
3. Reachability and non-blocking properties.

The following properties are expressed in TCTL logic and are checked with the UPPAAL model-checker:

1. The absence of deadlock.
   In UPPAAL a state is defined as a deadlock state if there are no outgoing action or delay successor transitions.

   To check the absence of deadlock we have the property:

   ```
   A[]not deadlock
   ```

   This is a safety property. In our case, it is satisfied.
2. Always the customer will end up having his pizza.

   ```
   A<> customerPizza.Eat
   ```

   This property is not satisfied in our system. In this case, the UPPAAL model checker provides a counterexample. The latter is generated when a property is violated.

   The counterexample is a trace in the UPPAAL model, loaded into the simulator after verification. It can be browsed and displayed in the simulation pane or saved as a file. We can generate the shortest or the fastest.
3. If the time since the pizza was ordered exceeds 60 units of time, the customer never gets to eat pizza.

   ```
   A[]( timer >= 60 imply A<>
                  not customerPizza.Eat)
   UPPAAL:
   timer >= 60 --> not customerPizza.Eat
   ```

   This property is satisfied in our system.

Once the command time exceeds 60 units of time, the system ends with stop (unsuccessful termination). We can modify the specification to avoid the problem of starvation by adding a recursive process after the action *Askpizza* so that the client can execute the action *Eat*:

```
Specification Pizza[p]:=
Behavior L(pizzaCustomer);
where
process pizzaCustomer[p](timer):=
[true]->(Spizza;Opizza;p!x;
  (
  [timer>=60]-> Askpizza;B,
    []
  [timer<60]-> (p?x;Paypizza;Eatpizza;exit;)
  ))
EndProc
process B[p](timer):=
```

```
  (
  [timer>=60]-> Askpizza;B,
    []
  [timer<60]-> (p?x;Paypizza;Eatpizza;exit;)
  )
EndProc
EndSpec
```

With this new specification, we can confirm that the customer always ends up getting pizza. The property can be expressed in TCTL as follows:

```
A<> customerPizza.Eat
```

In this case, this is a liveness property.

5.2 Biometric Passport Request

In this section, we present a case study of BPMN modeling of the business process "biometric passport request".

The process begins with booking an appointment with any municipality, then entering the request, enrolling biometric data, and finally issuing a receipt. Afterwards, the citizen receives an SMS inviting them to come to the filing location to receive their biometric passport.

The proposed BPMN collaboration diagram contains two pools, "Citizen" and "Biometric Passport". The details of the processes are illustrated in Figures 5 and 6.

5.2.1 Data Enrollment Cycle

The data enrollment process is depicted in Figure 6. A passport application can be either a renewal, a new application, or a replacement of a lost, stolen, or damaged passport. These applications are verified and processed.

5.2.2 DD-LOTOS Specification of the Biometric Passport Application Process

The BiometricPassport process begins by executing the appointment request action, followed by receiving and sending data. In the case of a request regarding the passport's renewal, and if the expiry date of the passport is greater than six months, the request is refused. If the date is less than six months, the request is sent, followed by the Enrollment process. In the Enrollment process, if the applicant's age is over 45 years old, the passport will be issued directly. On the other hand, if the age is less than 45 years old, then a verification step of the information is necessary.

```
Specification  BiometricPassportApplication
    [App_Request,Receipt_date,Fix_app,Enter_req]() :=
Behavior
```

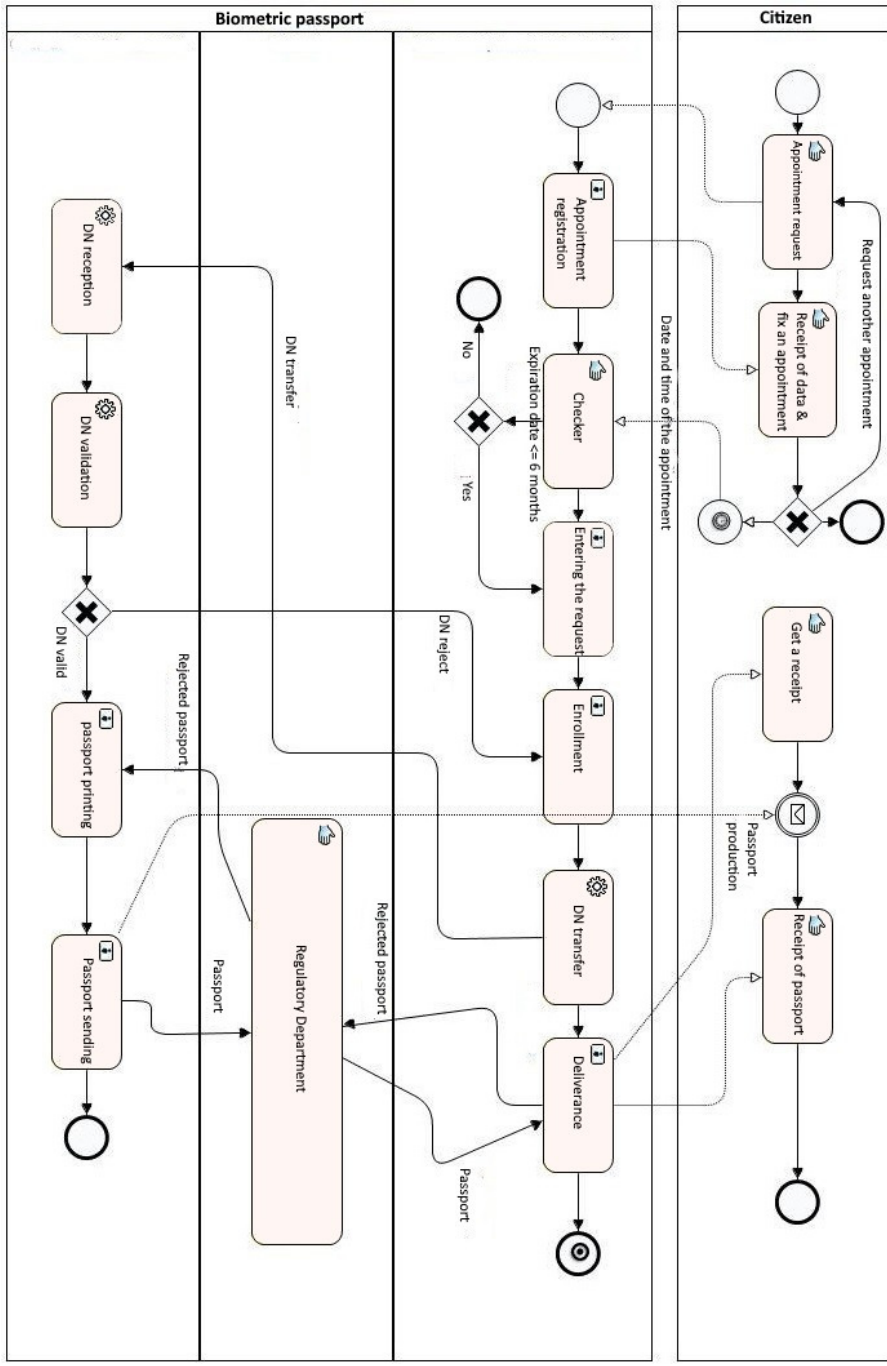

Figure 5. Biometric passport application process

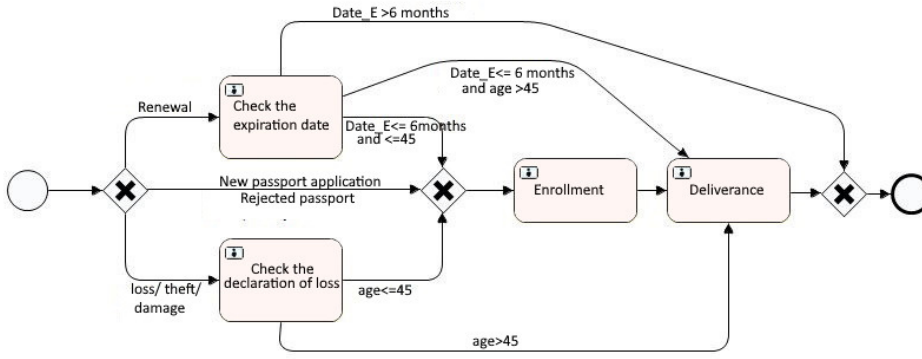

Figure 6. Enrollment cycle sub-process

```
        L(BiometricPassport);
Where
Process BiometricPassport[App_Request,Receipt_date,
                                    Fix_app,Enter_req]:=
App_Request;Receip_data;Fix_app!date;
([date> 6] → exit;
    []
 [date<= 6] → (Enter_req!request;
               Enrollment(request)>>
               DN_transfert[DN_receipt]>>Deliverance;
               exit)
)
EndProc

Process Enrollment(request):=
[request ='Renewal'] →
 (
 [date>6] → exit
    []
 [date<=6 and age>45] → Deliverance;exit
    []
 [date<=6 and age<=45] → Enrollment(request)
                                    >>Deliverance
 )
  []
[request ='Loss'or theft or damage] →
 (
```

```
  [age>45] → Deliverance
   []
  [age<=45] → Enrollment(request)>>Deliverance
 )
  []
[request ='New'or Reject] →
                          (Enrellment;Deliverance;exit)
Endproc

Process DN_transfert[DN_reception]:=
DN_reception!DN;
 (
  [DN ='valid'] →
     passport_printing;passport_sending;
               Regulatory_depart;exit
   []
  [DN ='invalid'] → Enrollment(request);exit
 )
Endproc

Process Deliverance:=
Get_receipt;passport_prod;Receipt_passport;exit
Endproc
Endspec
```

From this specification, we generate the equivalent C-DATA model. The system is described in Figure 7 and is composed of three automata.

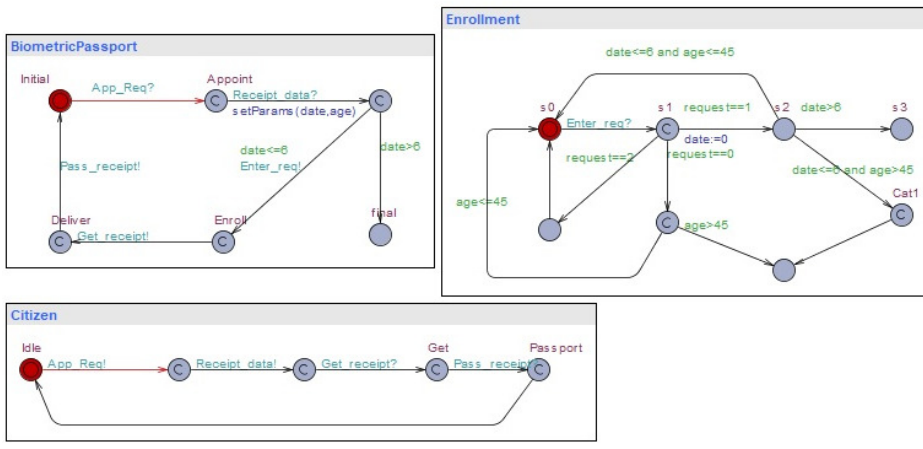

Figure 7. System in UPPAAL

5.2.3 Model Checking

Figure 8 shows the model checking of some properties using the UPPAAL tool for the biometric passport case.

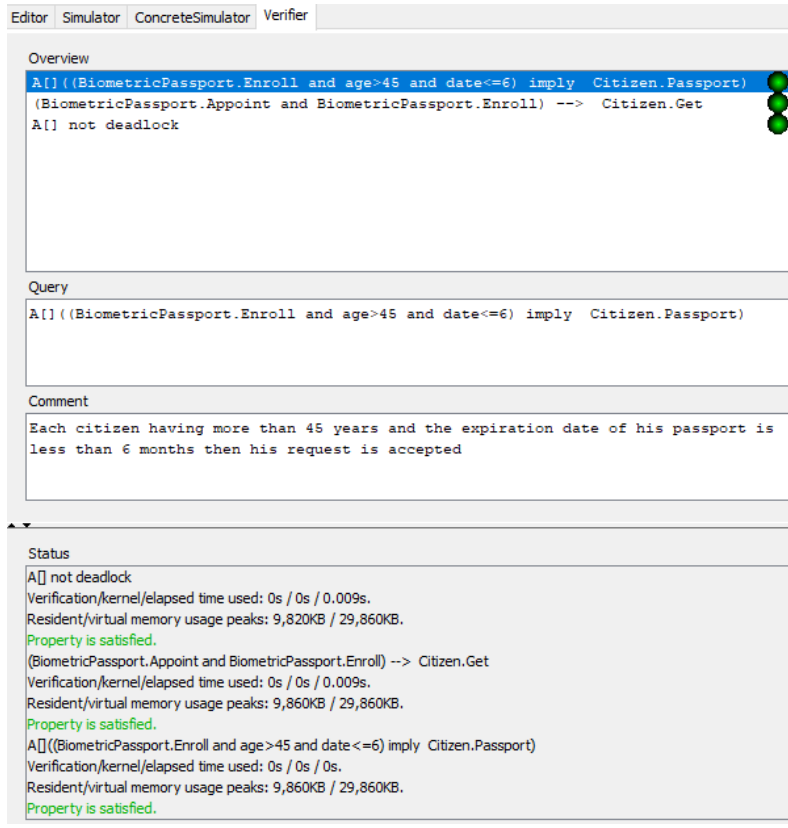

Figure 8. Model checking of TCTL properties

1. The absence of deadlock.

   ```
   A[]not deadlock
   ```

   It is a safety property. In our case, it is satisfied.

2. Each citizen must obtain a receipt once they have filed a passport application. In TCTL this property is expressed by the following expression:

   ```
   A[]((BiometricPassport.Appoint and
       BiometricPassport.Enroll)imply
   ```

```
    A<> Citizen.Get)

UPPAAL:
(BiometricPassport.Appoint and
BiometricPassport.Enroll)--> Citizen.Get
```

Appoint is a logical assertion attached to a state in the system; it indicates that the citizen has applied for a passport.

Enroll indicates that the citizen provided this information and that the expiry date of their passport is less than six months away.

Get indicates that the citizen received a receipt.

This is a liveness property. In our case, it is satisfied.

3. For each citizen older than 45 years whose passport expiration date is less than six months away, the request is accepted. In TCTL this property is expressed by the following expression:

```
A[]((BiometricPassport.Enroll and age>45
   and date<=6) imply  Citizen.Passport)
```

Passport indicates that the citizen has to recover their passport.

This is a liveness property. In our case, it is satisfied.

6 CONCLUSION AND FUTURE WORK

One of disadvantages of the BPMN notation is the lack of formal semantics concerning its diagrams. This insufficiency can lead to inconsistencies in the development process. Several works have been proposed to provide different solutions to this problem. However, most works proposed different formalizations in terms of the model of specification, such as CCS, CSP, LOTOS, Maude, Petri nets. The major drawback of these approaches is that they are defined on an interleaving semantics. In other words, that the actions are instantaneous and atomic with null durations. In the proposed approach, we escape this hypothesis by the fact that DD-LOTOS language supports structural and temporal non-atomicity of actions.

Consequently, in this work, we have proposed a model-based formal verification approach for BPMN collaboration diagrams. The first step is to transform the BPMN collaboration diagram into a formal DD-LOTOS code. The second step consists of generating the semantic model for a possible verification step. The semantic model defined for DD-LOTOS is the C-DATA.

In the verification step, we have proposed checking properties to ensure the safety and liveness of the system.

As future work, we are interested in validating the choreography diagrams. This work will involve carrying out an automatic approach from the modeling by the choreography diagrams to the generation of the C-DATA semantic model, to which we will then apply the model-based checking.

REFERENCES

[1] OMG: Business Process Model and Notation (BPMN). Object Management Group, 2011. Availaible at: `https://www.omg.org/spec/BPMN/2.0/`.

[2] van der Aalst, W.—La Rosa, M.—Santoro, F. M.: Business Process Management – Don't Forget to Improve the Process! Business and Information Systems Engineering, Vol. 58, 2016, No. 1, pp. 1–6, doi: 10.1007/s12599-015-0409-x.

[3] Fortiş, A.: Business Process Modeling Notation – An Overview. Annals Computer Science Series, Vol. 4, 2006, No. 1, pp. 41–49.

[4] Takemura, T.: Formal Semantics and Verification of BPMN Transaction and Compensation. Proceedings of the 3rd IEEE Asia-Pacific Services Computing Conference (APSCC 2008), Yilan, Taiwan, 2008, pp. 284–290, doi: 10.1109/apscc.2008.208.

[5] Dijkman, R. M.—Dumas, M.—Ouyang, C.: Semantics and Analysis of Business Process Models in BPMN. Information and Software Technology, Vol. 50, 2008, No. 12, pp. 1281–1294, doi: 10.1016/j.infsof.2008.02.006.

[6] Martens, A.: Analyzing Web Service Based Business Processes. In: Cerioli, M. (Ed.): Fundamental Approaches to Software Engineering (FASE 2005). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 3442, 2005, pp. 19–33, doi: 10.1007/978-3-540-31984-9_3.

[7] Raedts, I.—Petković, M.—Usenko, Y. S.—Van der Werf, J. M.—Groote, J. F.—Somers, L.: Transformation of BPMN Models for Behaviour Analysis. Proceedings of the 5th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems (MSVVEIS 2007), Funchal, Madeira, Portugal, pp. 126–137, doi: 10.5220/0002428801260137.

[8] Decker, G.—Weske, M.: Interaction-Centric Modeling of Process Choreographies. Information Systems, Vol. 36, 2011, No. 2, pp. 292–312, doi: 10.1016/j.is.2010.06.005.

[9] Wong, P. Y. H.—Gibbons, J.: Verifying Business Process Compatibility. The Eighth International Conference on Quality Software, Oxford, UK, 2008, pp. 126–131, doi: 10.1109/qsic.2008.6.

[10] Wong, P. Y. H.—Gibbons, J.: A Relative Timed Semantics for BPMN. Electronic Notes in Theoretical Computer Science, Vol. 229, 2009, No. 2, pp. 59–75, doi: 10.1016/j.entcs.2009.06.029.

[11] El-Saber, N.—Boronat, A.: BPMN Formalization and Verification Using Maude. Proceedings of the 2014 Workshop on Behaviour Modelling – Foundations and Applications, ACM, 2014, Art. No. 1, pp. 1–12, doi: 10.1145/2630768.2630769.

[12] Corradini, F.—Fornari, F.—Polini, A.—Re, B.—Tiezzi, F.—Vandin, A.: A Formal Approach for the Analysis of BPMN Collaboration Models. Journal of Systems and Software, Vol. 180, 2021, Art. No. 111007, doi: 10.1016/j.jss.2021.111007.

[13] Maarouk, T. M.—Merah, E.—Ghaoui, S.—Rahabi, N.: Formal Semantics and Transformation of BPMN Models. International Journal of Business Process Integration and Management, Vol. 9, 2019, No. 3, pp. 158–169, doi: 10.1504/IJBPIM.2019.100922.

[14] Saidouni, D. E.—Courtiat, J. P.: Relating Maximality-Based Semantics to Action Refinement in Process Algebras. In: Hogrefe, D., Leue, S. (Eds.): Formal Description Techniques VII (FORTE 1994). Springer, Boston, IFIP Advances in Information and Communication Technology, 1995, pp. 293–308, doi: 10.1007/978-0-387-34878-0_24.

[15] Maarouk, T. M.—Saidouni, D. E.—Khergag, M.: DD-LOTOS: A Distributed Real Time Language. Proceedings 2nd Annual International Conference on Advances in Distributed and Parallel Computing (ADPC 2011) Special Track: Real Time and Embedded Systems (RTES 2011), Published and organized by Global Science and Technology Forum (GSTF), Singapore, 2011, pp. 45–50.

[16] Maarouk, T. M.—Saidouni, D. E.—Mahdaoui, R.—Houassi, H.: Interpretation of DD-LOTOS Specification by C-DATA*. In: Morzy, T., Valduriez, P., Bellatreche, L. (Eds.): New Trends in Databases and Information Systems (ADBIS 2015). Springer, Cham, Communications in Computer and Information Science, Vol. 539, 2015, pp. 414–423, doi: 10.1007/978-3-319-23201-0_42.

[17] 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, Vol. 119, 2021, Art. No. 100630, doi: 10.1016/j.jlamp.2020.100630.

[18] Corradini, F.—Morichetta, A.—Polini, A.—Re, B.—Rossi, L.—Tiezzi, F.: Correctness Checking for BPMN Collaborations with Sub-Processes. Journal of Systems and Software, Vol. 166, 2020, Art. No. 110594, doi: 10.1016/j.jss.2020.110594.

[19] Krishna, A.—Poizat, P.—Salaün, G.: Checking Business Process Evolution. Science of Computer Programming, Vol. 170, 2019, pp. 1–26, doi: 10.1016/j.scico.2018.09.007.

[20] Kheldoun, A.—Barkaoui, K.—Ioualalen, M.: Formal Verification of Complex Business Processes Based on High-Level Petri Nets. Information Sciences, Vol. 385–386, 2017, pp. 39–54, doi: 10.1016/j.ins.2016.12.044.

[21] Cheikhrouhou, S.—Kallel, S.—Guermouche, N.—Jmaiel, M.: Toward a Time-Centric Modeling of Business Processes in BPMN 2.0. The 15th International Conference on Information Integration and Web-Based Applications and Services (IIWAS 2013), Austria, 2013, pp. 154–163, doi: 10.1145/2539150.2539182.

[22] Mallek, S.—Daclin, N.—Chapurlat, V.—Vallespir, B.: Enabling Model Checking for Collaborative Process Analysis: From BPMN to 'Network of Timed Automata'. Enterprise Information Systems, Vol. 9, 2015, No. 3, pp. 279–299, doi: 10.1080/17517575.2013.879211.

[23] Güdemann, M.—Poizat, P.—Salaün, G.—Dumont, A.: VerChor: A Framework for Verifying Choreographies. In: Cortellessa, V., Varró, D. (Eds.): Fundamental Approaches to Software Engineering (FASE 2013). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 7793, 2013, pp. 226–230, doi: 10.1007/978-3-642-37057-1_16.

[24] van der Aalst, W. M. P.—ter Hofstede, A. H. M.: YAWL: Yet Another Workflow Language. Information Systems, Vol. 30, 2005, No. 4, pp. 245–275, doi: 10.1016/j.is.2004.02.002.

[25] Wynn, M. T.—Verbeek, H. M. W.—van der Aalst, W. M. P.—ter Hofstede, A. H. M.—Edmond, D.: Business Process Verification – Finally a Reality! Business Process Management Journal, Vol. 15, 2009, No. 1, pp. 74–92, doi: 10.1108/14637150910931479.

[26] Wynn, M. T.—Verbeek, H. M. W.—van der Aalst, W. M. P.—ter Hofstede, A. H. M.—Edmond, D.: Reduction Rules for YAWL Workflows with Cancellation Regions and OR-Joins. Information and Software Technology, Vol. 51, 2009, No. 6, pp. 1010–1020, doi: 10.1016/j.infsof.2008.12.002.

[27] Saidouni, D. E.—Courtiat, J.-P.: Taking into Account the Duration of Action in Process Algebras Through the Use of Maximality Semantics. Protocol Engineering (CFIP 2003), 2003 (in French).

[28] Hoggas, N.: Checking Business Process BPMN: Logical Approach. Master Thesis, 2020 (in French).

Toufik Messaoud Maarouk is Lecturer in the Department of Mathematics and Computer Science, Faculty of Sciences and Technology, University of Khenchela, Algeria. He received his Ph.D. in computer science from the Constantine University, Algeria, in 2012. His main areas of research include formal methods, concurrency theory, formal semantics and distributed computing.

Mohammed El Habib Souidi is Lecturer in the Department of Mathematics and Computer Science, Faculty of Sciences and Technology, University of Khenchela, Algeria. He received his Ph.D. in computer science from the Harbin Institute of Technology (China), in 2017. His main areas of research include multi-agent task coordination, reinforcement learning, game theory and path planning.

Nadia Hoggas received her M.Sc. in computer science from the University of Khenchela, Algeria in 2020.