International Journal of Informatics and Applied Mathematics
e-ISSN:2667-6990 Vol. 3, No. 2, 35-52

Exploring Specifications and Monitoring Execution Data of Business Processes

Ali Khebizi[1], Hassina Seridi-Bouchelaghem[2]

[1] Department of computer science, LabSTIC Laboratory, 8 May 1945 University, P.O. Box 401, 24000 Guelma, Algeria- khebizi.ali@univ-guelma.dz, ali.khebizi@gmail.com

[2] LabGed, Badji Mokhtar University, Annaba -Algeria- seridi@labged.net

Abstract. The large proliferation of software environments supporting enterprises business processes has lead to a massive data that is receiving a great deal of enthusiasm from the IT managers for its exploitation for various management purposes. Although a lot of work in the business processes field has focused mainly on the modelling and automation aspects, little effort has been done regarding the analysis, optimization and monitoring concerns.

In this paper, a new formal approach for analysing business processes specifications and for exploring their associated execution data is suggested. It exhibits two main complementary features: (i) formal specifications describing business processes, expressed as finite state machines, are extracted in order to satisfy useful users needs, (ii) a parametrizable model supporting **selection rules** that enables querying execution traces of business processes is formalized and illustrated. A real-word scenario is used throughout the paper to illustrate the introduced concepts and formalizations and the proposed approach is implemented and experimented in a software tool.

Keywords: Business processes · Finite state machine· Specification pattern· Process instance· Execution path· Data analysis.

1 Introduction

Modern organizations invest colossal sums and a valuable budget for modelling, managing and monitoring their business processes.
A business process (*in the sequel **BP** for short*) is usually defined as a collection of coordinated activities undertaken by one or more organizations in pursuit of some particular business goals [1]. Its specification must describe how business logics are performed and how they should be conducted under real organizational constraints.

In parallel, **B**usiness **P**rocess **M**anagement (**BPM**) technology is recently recognized as a comprehensive discipline that promote a process-centred approach to align organizations business processes with clients needs and to continuously improve business effectiveness and efficiency.

Nowadays, BPM is intensively used to automate execution of most companies' business logics, to improve processes and to align the needs of customers with company objectives. An immediate consequence of the large adoption of the BPM technology, corroborated by the proliferation of environments supporting different phases of the life cycle of BPs (*e.g.; development, deployment, execution, monitoring*), consists to multiple execution by different users of the activities contained in the deployed BPs. These multiple invocations leave various kinds of temporary or permanent traces in enterprises persistent storage systems. Thereby, over the time a large size of data requiring more attention by IT managers, is generated and stored in the underlying enterprise database(s).
Although a lot of work in the field of BPs has focused mainly on business process modelling and automation, little effort has been done regarding the analysis, optimization and monitoring of BPs. An important omission in current development practice for business process management systems (**BPMS**) consists of analysing specifications for searching particular patterns of interest and comparing relationships of BPs descriptions, including similarities' search and checking inclusions of sub-structures. Another shortcoming is related to querying and exploring generated execution data in the underlying enterprise database(s) that take into account the particular nature of BPs executions traces. Indeed, processes' data are very relevant in the contexts of automated BPs substitution and for maintenance and monitoring tasks, while giving organizations control over their processes and making businesses more efficient and more competitive.
This article addresses such concerns, by presenting a comprehensive approach that allows analysing BPs data. The main goal is to extract relevant knowledge contained in BPs' data storage systems, such as: *relationships between activities and BPs, BPs that can replace failed ones, the most-executed activities or the followed paths, instances status, achieved activities ...*

We propose a formal approach for BPs data analysis that provides a sound theoretical foundation and an operational dashboard for protocol managers. In this perspective, BPs' relationships are formally specified and handled (*inclusion, simulation , ...*). Furthermore, to explore instances of BPs meeting users requirements, a generic query model is specified and evaluated.
The salient feature of the proposed work lies in a rigorous formalism based on

high level specifications to support the analysis and exploration of BP data. BPs are described by means of **F**inite **S**tate **M**achines (**FSM**), descriptive structures are represented with regular expressions and relationships between BP are formally specified. At the execution level, execution paths followed by process instances are expressed as path expressions and a selection rule model is proposed. The conceived approach is implemented via a software prototype.
Paper organization: We start by positioning the problem and motivating our work in section 2. Section 3 presents the different facets of the proposed approach for BPs data analysis and exploration. In section 4, the system architecture is illustrated and the implementation of a software prototype supporting the approach is depicted. Finally, conclusion and potential directions for future works are drawn at section 5.

2 Problem statement and motivations

Given its importance, analysis of BPs data has recently received great attention and interest from both the research and industry communities [2–5], as effectively accomplishing any business process management tasks requires understanding the behaviour of the processes. This behaviour is manifested by the descriptive specifications and reflected by real execution data. Thus, BP data analysis, monitoring and exploration become a critical task for any organization. This need is felt on the two following distinct levels.

At the formal specification level, the main objective of BPs analysis is to allow, on the one hand, the continuous improvement of the already deployed BPs, by taking into account events and factors that arise in their execution environment, such as executions failures induced by network and communication reasons or needs for BPs substitution due to evolution constraints, when new regulations and laws occur [6]. On the other hand, handling BP structures' analysis enables recovery and maintenance actions to be carried out on schemes which manifest errors, anomalies or overloads related to activities' invocation.

Contrary, at the execution data level, the concerns are of a different nature. In fact, once a BP is deployed (*as a Web service, for example*), it is seen as a large public application, for which a huge number of customers may be invoking it at a same time. Each invocation by a given user corresponds to an execution of a separated instance of the BP and over the time, a large volume of data is being generated on a continuous basis. Exploring such generated data is a major asset for evaluating and monitoring BPs and, thus enhancing the organization's resources and its position in the market.
Due to the particular nature of BPs data, the previous requirements can't be easily expressed -or not at all- by using only traditional querying languages.

2.1 Motivating scenarios

First, we present the purchase-order BP, then we illustrate its usage with two concrete scenarios.

Example 1. A real-world example of a purchase-order BP, called Order, is depicted in **Fig. 1** and it's used for managing sales and strengthening customers relationships. This BP could be deployed on the web by a network of companies (*as a Web service), by publishing its interface and its protocol [7, 8].*

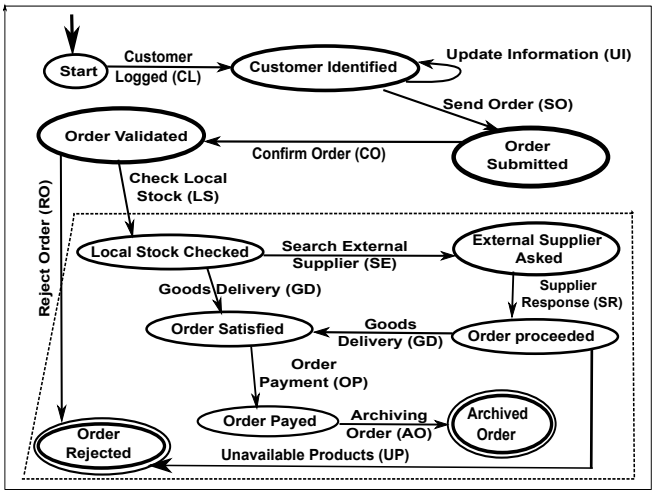

Fig. 1. The purchase-order business protocol (Order)

The business logic behind this protocol is described in what follows. Initially, a new instance of the Purchase-Order protocol is created at the Start state, when a given customer logs to the system. Then, the instance moves to the state Customer Identified when the user fills and submits an application identification form (activity Customer Logged (CL)). Once the customer is authenticated, he can either update his information iteratively, (activity Update Information (UI), *e.g., to change the delivery address or the payment method*), or he can send the purchase order to the supplier (activity Send Order (SO)). This last activity makes the BP instance moving to the state Order submitted and it continues its execution according to the specification described by the protocol.

Scenario 1: Exploring BP structures First we focus on BP models by analysing their descriptive specifications. For example, we can check the existence of particular sub-procedures (*sequence of activities*) or, filter out the most typical BP that can ensure the same activities as those of a failed or overloaded one, or further more, ask for the set of activities allowed from a reached state in the current BP. Thus, some questions, such as: *which active BP contains a particular critical activities' sequence ? The current BP can be replaced by which other ones ? What are the remaining activities for accomplishing a procedure ?*, are examples of frequent concerns to be addressed at the specification level.

Scenario 2: Exploring BP execution data Now, assume that a BP manager wants to filter out instances of the previous BP Order satisfying particular resources constraints and specific management requirements. As an illustration, he may be interested by questions such as: *Are there any bottlenecks in the ordering process? Who proceeded it and what are the most followed procedures? What are the historical traces of instances and their reached states? as well as displaying instances with their completed activities and those in progress.*
Other management concerns can be addressed. For instance, *Is there some unreachable BP states or execution paths that are never taken by instances? What are process instances having not yet executed a particular activity? What activities have been completed by a given customer? Those in progress?*
Answering the previous questions helps considerably protocol managers improving the efficiency and the effectiveness of the deployed processes and facilitates the monitoring tasks. However, formulating such queries by known query languages, such **SQL**, is far from trivial and can't be often easily expressed with traditional query languages.

2.2 Motivations

By extracting relevant information stored in enterprises systems, organizations can improve the quality of their BP and enhance services provided to partners. Thus, analysing BP data aims to achieve to following three goals.

- Managing BPs **substitution** with respect to historical execution traces. Many situations, such as processes failure or best performances and costs reduction needs, require to replace the concerned protocol with another adequate one which offers, at least, the same functionalities. In such contexts, it's imperative to analyse protocol's specification and the accurate progression level of each instance in order to ensure its execution continuation.
- Handling BP **evolution** induced by changes in laws, regulation and policies. In fact, change management requirements imposes to processes' manager comparing old and new protocols and analysing the exact progression level of each process instance in order to handle change impact on execution traces by deploying the suitable migration strategy;
- Extracting **decisions metrics** (*number of pending instances, accomplished steps, activities and process instances duration, rate of achieved instances, instances that have been blocked for an elapsed time, ...*).

From a strategic perspective, exploiting existing BP data constitutes an efficient management tool supporting growth and competitiveness of modern organizations. To meet this goal, techniques, methods and tools are inevitable.
In what follows, we expose our approach for BP data analysis.

3 Analysing and monitoring business processes data

We present below the different facets of our approach for BP data analysis and we expose the underlying models. *(i)* First, we introduce the explicit choices on

formal models used for representing business processes and execution traces, as well as some basic notations and definitions useful for formalizing the approach. *(ii)* Structural analysis of BP is addressed at the abstract specification level. *(iii)* A formal query model for exploring BP instances is described and formalized. In what follows, these steps are deeply discussed and illustrated.

3.1 Business processes and execution traces specification

The formal protocol model manipulated throughout the proposed approach and the associated notations, as well as basic definitions are introduced bellow.

3.1.1 The business protocol model A business protocol, (*shortly* ***the protocol***), describes the external visible behaviours of a given business process, by specifying the constraints (*e.g., ordering of activities, ...*) that partners must comply with in order to correctly interact with the process [9–11].
In our work a basic version of the model basing on automaton is used. It describes the ordering constraints that govern the activities' execution [9, 11, 12] and it's considered as sufficient to illustrate our approach. More formally.

Definition 1 *A business protocol is a tuple* $\mathcal{P} = (S, s_0, \mathcal{F}, \mathcal{M}, \mathcal{R})$*; where:* S *is a finite set of states;* $s_0 \in S$ *is the initial state of the protocol;* $\mathcal{F} \subseteq S$ *is the set of final states;* $\mathcal{M}$ *is a finite set of abstract activities;* $\mathcal{R} \subseteq S \times S \times \mathcal{M}$ *is a transition relation. Each element* $(s, s', m) \in \mathcal{R}$ *represents a transition from a source state* s *to a target one* s' *upon the execution of the abstract activity* m*.*

According to this definition, states of the **FSM** represent the different phases that a BP may go through, while transitions represent activities that a BP can perform to move from one step to another according to the BP logic [10, 12].
The choice of finite state machines to represent protocols of BPs is motivated by the important role of this formalism to represent the behaviour of dynamic systems [13]. Further, this formalism has been extensively used in the field of process theory, **BPM** and web services to support formal analysis [14]. Such a model is very adequate for our purposes since it enables to express the abstract activities and to specify constraints required by users to correctly interact with the BP. Although our approach uses such models, other existing models such PetriNets, BPMN or UML diagrams can be translated to **FSM** by using adequate transformation techniques [15, 16].

3.1.2 Basic definitions and notations regarding the BP model Each invocation of a deployed business process corresponds to an execution of a separated instance of the considered process. This execution generates an execution trace which is recorded in adequate databases (*log files*). In the following we introduce the concept of *execution path* which is necessary for expressing execution traces of process instances.
Let $\mathcal{P} = (S, s_0, \mathcal{F}, \mathcal{M}, \mathcal{R})$ be a business process.

An **execution path** $e(\mathcal{P})$ of $\mathcal{P}$ is an alternating sequence of states and activities of $\mathcal{P}$, i.e., $e(\mathcal{P}) = s_k.m_k.s_{k+1}.m_{k+1} \ldots s_n.m_n.s_{n+1}$, that (i) starts at a state s_k of $\mathcal{P}$, (ii) ends at a state s_{n+1} of $\mathcal{P}$, and (iii) is consistent with the transition relationship of $\mathcal{P}$, i.e., $(s_i, s_{i+1}, m_i) \in \mathcal{R}, \forall i \in [0, n]$.
Each instance I interacting with the business process $\mathcal{P}$ generates an **execution trace** $\mathcal{T}_i(\mathcal{P})$, which is made of a finite sequence of activities $m_0.m_1 \ldots m_n$ obtained by removing state names from the associated execution path $e(\mathcal{P}) = s_0.m_0.s_1 \ldots s_n.m_n.s_{n+1} \in \mathcal{P}$ followed by the instance I.
According to the previous definition, an execution trace $\mathcal{T}_i(\mathcal{P})$ of an instance I represents the sequence of historical activities performed by I in $\mathcal{P}$, from the beginning of the process invocation to the current state reached by the instance. We denote by $\mathcal{T}(\mathcal{P})$, the set of all execution traces $\mathcal{T}$ of a business protocol $\mathcal{P}$ and $|\mathcal{T}(\mathcal{P})|$ designates the cardinality of this set. Each element in $\mathcal{T}(\mathcal{P})$ is expressed with $\mathcal{T}_i(\mathcal{P})$, for $i = 1 \ldots |\mathcal{T}(\mathcal{P})|$ and the length of a trace $\mathcal{T}_i(\mathcal{P})$ is noted $|\mathcal{T}_i(\mathcal{P})|$, *i.e., the number of activities contained in the trace* $\mathcal{T}_i(\mathcal{P})$.

Example 2. **Table 1** bellow shows nine execution traces; $\mathcal{T}_1(Order), \ldots, \mathcal{T}_9(Order)$ associated to nine separate instances of the purchase-order protocol of **Fig.1**. At a given time, the considered execution traces have different status and are at different execution levels.

Table 1. Examples of execution traces of the *purchase-order* business protocol.

ID	Trace Name	Execution Trace	Start-date	Start-time	T-stamp	Length	Status
100	$\mathcal{T}_1(Order)$	CL.SO.CO	28/06/20	19:11:17	320 s	3	A
110	$\mathcal{T}_2(Order)$	CL.UI.UI.SO.CO.RO	30/06/19	20:15:14	56 s	6	C
153	$\mathcal{T}_3(Order)$	CL.SO.CO.LS.SE	15/07/20	10:05:11	200 s	5	A
288	$\mathcal{T}_4(Order)$	CL.UI.SO.CO.LS.GD.OP.AO	16/02/18	22:53:08	3600 s	8	C
28	$\mathcal{T}_5(Order)$	CL.SO.CO.LS	15/03/20	09:33:11	5000 s	4	A
88	$\mathcal{T}_6(Order)$	CL.UI.UI.UI.UI	10/10/20	01:15:22	78 s	5	B
115	$\mathcal{T}_7(Order)$	CL.SO.CO.LS.SE.SR.GD	23/02/20	16:17:18	3456 s	7	A
214	$\mathcal{T}_8(Order)$	CL.UI.SO.CO.LS.GD.OP	11/04/20	23:51:45	300 s	7	B
171	$\mathcal{T}_9(Order)$	CL.UI.UI.SO.CO.LS.GD.OP.AO	08/03/20	10:08:00	6000 s	9	C

It's important to notice that execution traces are characterized by other attributes related to execution data, such as the state reached by the process instance, the needed resources and the incurred costs. For example, the instance with the Trace Name= $\mathcal{T}_7(Order)$ and having executed the activities sequence CL.SO.CO.LS.SE.SR.GD is at the current state Order Satisfied, it needs 1000 monetary units and a network connection as resources, while the already engaged cost is about 20 units. Due to lack of space we focus only on the properties presented in **Table 1** which are sufficient for illustrating our approach.
Regarding the progression level of execution, an execution trace $\mathcal{T}_i(\mathcal{P})$ is **complete** if it has reached a final state, else it's called **incomplete** and it's still **active**. Contrary, some instances may be at a blocked state due to network

problems or lack of resources reasons. In **Table 1**, the abbreviations ***A**: Active, **B**: Blocked and **C**: Complete* are used to designate instance status.
Finally, we introduce the notion of **sub-protocol** which is very important in our future developments.

Definition 2 *Let $\mathcal{P} = (S, s_0, \mathcal{F}, \mathcal{M}, \mathcal{R})$ be a **FSM** representing a business protocol and let $s \in S$ a state of $\mathcal{P}$. The sub-protocol $\mathcal{C}_{\mathcal{SP}}(\mathcal{P}, s)$ of $\mathcal{P}$ is the protocol $\mathcal{C}_{\mathcal{SP}}(\mathcal{P}, s) = (S'_S, s'_{0S}, \mathcal{F}'_S, \mathcal{M}'_S, \mathcal{R}'_S)$ obtained from $\mathcal{P}$ as follows :*

- *s is the initial state of $\mathcal{C}_{\mathcal{SP}}(\mathcal{P}, s)$, $s'_{0S} = s$.*
- *$S'_S \subseteq S$ contains all states of S reachable from s by using the transition relation $\mathcal{R}$ of $\mathcal{P}$,*
- *$\mathcal{F}'_S = S'_S \cap \mathcal{F}$,*
- *$\mathcal{R}'_S = \{(s, s', m) \in \mathcal{R} \mid \{s, s'\} \subseteq S'_S\}$,*
- *$\mathcal{M}'_S \subset \mathcal{M}$ is constituted of messages $\mathcal{M}$ appearing in the transitions of $\mathcal{R}'_S$.*

Thus, the sub-protocol $\mathcal{C}_{\mathcal{SP}}(\mathcal{P}, s)$ allows to capture all the behaviours of $\mathcal{P}$ starting from the state s. For instance, the protocol $\mathcal{C}_{\mathcal{SP}}$ (**Order**, Local Stock Checked) of the Purchase Order protocol of **Fig. 1** corresponds to the sub-protocol starting at the initial state s'_{0S}=Local Stock Checked and including all the states reachable from this initial state.

*(the sub-protocol is highlighted with dashed lines in **Fig. 1**).*

Consequently, a sub-protocol specification represents a business logic with its constraints and it corresponds to a particular working sub-procedure.
We now recall the notion of **simulation relationship** which is very useful for comparing two protocols based on their behaviours [10].

Definition 3 *Let $\mathcal{P} = (S, s_0, \mathcal{F}, \mathcal{M}, \mathcal{R})$ and $\mathcal{P}' = (S', s'_0, \mathcal{F}', \mathcal{M}', \mathcal{R}')$ be two **FSM** representing two business protocols, such that:*
A state $s_1 \in s$ is simulated by a state $s'_1 \in S'$, noted: $s_1 \ll s'_1$, Iff the following two conditions holds:

- *$\forall\ m \in \mathcal{M}$ and $\forall\ s_2 \in S$ with :$(s_1, s_2, m) \in \mathcal{R}$, there exists $(s'_1, s'_2, m) \in \mathcal{R}'$ such as: $s_2 \ll s'_2$.*
- *If $s_1 \in \mathcal{F}$, then $s'_1 \in \mathcal{F}'$.*
  $\mathcal{P}$ is simulated by $\mathcal{P}'$, noted: $\mathcal{P} \ll \mathcal{P}'$ Iff: $s_0 \ll s'_0$.

3.2 Analysing business processes specifications

BP data analysis at the descriptive level, (*i.e., the model level*), deals with specifications' properties expressed in BP models, by exploring the contained structures in order to seek patterns of interest. In fact, over the time a large volume of BP descriptive data is being generated and stored in adequate databases. Since these data are, generally, based on W3C standards, especially the **XML** format, it is so possible to process them in order to extract the knowledge hidden in BP models and the potential relationships among them. In this sense, protocol

managers are often interested by examining particular execution paths with their allowed and forbidden activities' sequences. Also, in many situations they want to check the existence of specific sub-procedures. Such concerns express useful management needs and are deeply studied in what follows.

3.2.1 Specification patterns deployment Generally, protocol managers explore and analyse protocols' models for searching descriptions that express particular management constraints. The aim is to check if the current protocol offers particular functionalities reflecting behaviours relating to a specific business rules. This is the scenario, for example, when they want *ensure that the business process allows operating a repetitive selection of articles ? if it's possible to cancel the purchase order after having operated the payment ?* or *at which steps the protocol logic tolerate to withdraw the purchase order ?*
More generally, patterns' search aims to explore protocols specifications in order to analyse relationships between activities of the BP models. The most frequent relations between BPs operations that can offer a major and frequent interest are the following.

- **Succession rule**: *What is (are) the activity (ies) that follows (follow) a given activity* X *?*
- **Precedence rule**: *A given activity* X *is preceded by which activity (ies) ?*
- **Iteration rule:** *Does the business process tolerate the repetitive execution of a given activity* X *?*

To face to such concerns, we suggest using the concept of **Specification Pattern** (**SP**), which is an abstract tool used to express generic situations that can occur repeatedly during BP executions. According to the used formal description for representing BP as automaton, a **SP** is formally expressed by using regular expressions representing the executions paths of the corresponding protocol.
In what follows, we denote by $L(\mathcal{P})$ the language of $\mathcal{P}$, i.e., the set of all complete execution paths of $\mathcal{P}$, while w^0 and w^1 designate two valid activities sequences. We give now the formal definition of a specification pattern.

Definition 4. *Let* $\mathcal{P} = (S, s_0, \mathcal{F}, \mathcal{M}, \mathcal{R})$ *be a* ***FSM*** *representing a BP. A specification pattern* $\boldsymbol{SP}(\mathcal{P})$ *is an activities' sequence of* $\mathcal{P}$*, noted* $\lambda \in \mathcal{P}$*, for which* $\exists w \in L(\mathcal{P})$*, and such that* $w = w^0.\lambda.w^1$*, (i.e,* $\lambda \subset w \in L(\mathcal{P})$*).*

According to this definition, a specification pattern ensures that the activities sequence (*concatenated symbols*) λ is contained in a word satisfying the regular expression associated to the automaton $\mathcal{P}$. The activities sequence λ is made of a finite sequence of activities labels $m_0, m_1, \ldots, m_n \in \mathcal{P}$ which are aggregated with the concatenation operator "." This operator consists of joining names of activities end-to-end (e.g: CL.UI.SO).
We recall bellow the classical quantifiers used to formalize regular expressions.

- ? which defines an expression that exists zero or one time, e.g., $(\mathsf{a.b})^?$ corresponds to the empty execution path ϵ or a.b,

- $*$ which defines an expression that exists zero, one or more times, e.g., $(\mathsf{a.b})^*$ corresponds to the empty execution path ϵ or a.b, a.b.a.b, a.b.a.b.a.b, ...,
- $+$ which defines an expression that exists one or more times, e.g., $(\mathsf{a.b})^+$ corresponds to a.b or a.b.a.b or a.b.a.b.a.b, ...,

Example 3. **Using specification patterns**
Basing on the protocol model of **Fig.1**, the following three **SP** express different management needs.

- **Activities succession (SP1):** Does the activity Supplier Response (SR) follows directly the activity Check Local Stock (LS) ? the **SP** answering such a need is formalized with: SP1(Order)=w^0.LS.SR.w^1 (w^0, w^1 *are two valid activities' sequences* and $\lambda = LS.SR$), which is an expression that is not recognized by the automaton depicted in **Fig.1**. Thus, the pattern SP1(Order)=w^0.LS.SR.w^1 does not satisfy the protocol specification and the answer is negative.
  However, if we want to know if the customer have to confirm his purchase order, just after having sent it ? the adequate pattern (SP1'(Order)=$w^0.SO.CO.w^1$), which leads to a positive response is used.
  As another illustration: *The protocol permits delivering the ordered goods (activity GD) before order payment (activity OP)?*
  The **SP** answering such a requirement is expressed with:
  SP1"(Order)=$w^0.GD.OP.w^1$, which is an expression recognized by the automaton describing the BP, i.e., the activity GD precedes OP;
- **Activities precedence (SP2):** *What are the possible executions leading to the state "Order Satisfied" ?* .
  To reach the target state Order Satisfied, the activity Goods Delivery (GD) must be executed in one of the two possible paths. Backward activities preceding the activity GD are expressed with the following patterns:
  SP2(Order)=$CL.UI^*.SO.CO.LS.GD$ and
  SP2'(Order)=$CL.UI^*.SO.CO.LS.SE.SR.GD$. Thus, two separate paths leading to the target state Order Satisfied are obtained. Hence, the activities LS and SR precede GD.

- **Iteration (SP3):** *Can the customer update his information (activity UI) in several sessions ? can he proceed to order payment (activity OP) on several instalments ?* Such requirements are expressed, respectively, with the following specification patterns SP3(Order)=$w^0.UI^*.w^1$ and SP3'(Order)=$w^0.OP^*.w^1$. As it can be observed in **Fig.1**, the first pattern is a sequence that satisfies the protocol specifications. However, the second one isn't recognized by the protocol. Therefore payment must be made in a single transaction.

3.2.2 Processing complete execution paths Instead of dealing with simple specification patterns, as exposed above, the protocol manager can be interested in a complete management procedure that reflects an entire business goal. Such a requirement is described by an execution path that starts at the initial

state of the BP and it ends at one of its possible final states. We call such an execution path a **complete execution path**, which is formalized as follows.

Definition 5. *Let $\mathcal{P} = (S, s_0, \mathcal{F}, \mathcal{M}, \mathcal{R})$ be a business protocol.*
An execution path $e(\mathcal{P}) = s_k.m_k.s_{k+1}.m_{k+1} \ldots s_n.m_n.s_{n+1}$ is complete if: (i): s_k is the initial state of the automaton $\mathcal{P}$, i.e, $s_k = s_0$ and (ii): s_{n+1} is a final state of $\mathcal{P}$, i.e, $s_{n+1} \in \mathcal{F}$.

As an illustration, in the protocol of **Fig. 1** the path $e(Order)$ =Start.CL.Customer Identified.UI.Customer Identified.SO.Order Submitted.CO.Order Validated.RO. Order Rejected is a complete execution path starting from the initial state s_0=Start, followed by an alternating sequence of activities and states and which terminates at the final state Order Rejected $\in \mathcal{F}$.
The notion of complete execution path is useful in the following situations.

- **Substitution requirements:** if an obsolete procedure is to be replaced by a more elaborate one or when a business logic improvement must be operated;
- **Evolution requirements:** changes that arise in the organization's environment require updating existing tasks and procedures. For example, due to changes in laws and regulations;
- **Business processes maintenance:** The detection of anomalies in the deployed specifications of BP requires a continuous control to address execution bugs and deadlocks by operating corrective actions;
- **Compatibility reasons:** BPs integration involves several stakeholders and needs composing different processes [17]. In this context, procedures to be integrated must be compatible and compliant. If this not the case, adequate improvements and adjustments must be made.

Example 4. As an illustration, assume that the protocol manager decides to improve the BP depicted in **Fig. 1**, in a manner that it allows customers updating their purchase order in an iterative fashion. Thus, a new activity Update Purchase Order (UO) is to be inserted after the activities Confirm Order (CO) in the protocol specification. This improvement involves to create a cycle on the state Order Validated. To achieve the previous business goal, the protocol manager must substitute the complete execution path $e_1(Order)$ =Start.CL.Customer Identified.UI.Customer Identified.SO.Order Submitted.CO.Order Validated.RO. Order Rejected with the improved one: $e'_1(Order)$ =Start.CL.Customer Identified.UI.Customer Identified.SO.Order Submitted.CO.Order Validated.$(UO.OrderValidated)^*$.RO.Order Rejected. (*It's the same for the other possible execution paths)*

3.2.3 Ensuring sub-procedures achievement As explained previously in § **3.1.2**, the notion of sub-protocol specifies management sub-procedures which handle, at the same time, a set of working constraints, while allowing different execution paths starting from a particular state s. The concept of sub-protocol is very useful, for example, to ensure that some critical activities (e.g., Payment, Validation, Canceling, ...) are still available in the future executions. It could also be used to ensure that some commitments of a BP provider are still satisfied after

having operated changes on the BP description (*e.g., the ability for a customer to cancel the current execution after reaching a particular state*). Thus, the sub-protocol concept is a powerful formal tool that can be used in the following management scenarios.

- After a failure event. In order to ensure that all the executions from the failure state s are still allowed by the replacing protocol. This last one must perform, at least, the same activities which were allowed from the state s of the old protocol.
- It serves as a tool for facilitating BPs integration. In fact, inserting a sub-protocol $SP_1 \subset \mathcal{P}$ in another protocol $\mathcal{P}'$ is a common action requested during systems cooperation and interchange, for sharing common activities and business rules. For example, during enterprises merging and buyout.
- Ensuring forward compatibility of protocols [18] by checking that the set of future executions of an instance I, having reached a state $s \in \mathcal{P}$, are included in the set of forward execution paths starting from a corresponding state in the target protocol $\mathcal{P}'$ of a partner, i.e., $s' \in \mathcal{P}'$.
- In the BPs evolution context, sub-protocols are used to ensure that changes are transparent to customers interacting with the evolved BP. Hence, users are unawareness of changes between protocol versions $\mathcal{P}$ and $\mathcal{P}'$.

In all the previous situations, two versions $\mathcal{P}$ and $\mathcal{P}'$ of the used protocol, with their sub-protocols $\mathcal{C}_{\mathcal{SP}}(P, s)$ and $\mathcal{C}_{\mathcal{SP}'}(P', s'))$ are manipulated. To answer the previous requirements, the sub-protocol $\mathcal{C}_{\mathcal{SP}'}(P', s')$ can reproduce the same behaviour of the sub-protocol $\mathcal{C}_{\mathcal{SP}}(P, s)$, if and only if the old sub-protocol $\mathcal{C}_{\mathcal{SP}}(P, s)$ is simulated by the new one $\mathcal{C}_{\mathcal{SP}'}(P', s')$. More formally.

$$I_s(\mathcal{P}) \rightarrow I_{s'}(\mathcal{P}');\ iff : \mathcal{C}_{\mathcal{SP}}(P, s) \ll \mathcal{C}_{\mathcal{SP}'}(P', s')) \tag{1}$$

The semantics of the previous equation is interpreted as follows. An instance I having an execution path $e(\mathcal{P})$ and a current state $s \in \mathcal{P}$, can continue its execution from a corresponding state $s' \in \mathcal{P}'$, if when starting from s' the instance I can execute each execution that was possible from s. Thus, the simulation relation $\mathcal{C}_{\mathcal{SP}}(P, s) \ll \mathcal{C}_{\mathcal{SP}'}(P', s')$ makes it possible to ensure that s' can reproduce all the behaviours that were possible from the state s.

3.3 Querying business processes instances

Having in the same hand: (i) a process specification (*business protocol*) and (ii) its running instances (*followed execution paths*), in what follow we propose a mapping formalism that enables writing high-level specifications, called **selection rules**, that allow querying process instances in order to filter those instances that satisfy a set of user constraints.
To achieve this goal, generalized path expressions (*or simply path expressions in the sequel*), originally introduced to query semi-structured or schema-less data [19], are used to define selection rules. In fact, path expressions allow expressing navigational paths with (i) regular expression operators and (ii) path variables

(*i.e., variables that take as values paths*). More precisely.
Let $\mathcal{P}$ be a protocol specification. A selection rule of the form: $\sigma_{(\mathcal{P})}$ uses a path expression δ_e as a query to select instances of $\mathcal{P}$ that satisfy a given constraint. Thus, a query on BP instances is simply a path expression, i.e., $\sigma_{(\mathcal{P})} = \delta_e$.
We introduce these notions formally in the sequel.
Let Σ be the alphabet expressing business activities of $\mathcal{P}$. We denote by Σ^* the infinite set of paths obtained by the concatenations of the activities in Σ and $Exec(\mathcal{P})$ designates the set of all possible execution paths of $\mathcal{P}$.
First, we introduce the notions of path variables and path expressions. We denote by $\mathcal{V}$ a set of variables, called path variables (*i.e., variables that take as values execution paths*). A valuation over $\mathcal{V}$ and Σ is a mapping $\mu : \mathcal{V} \rightarrow \Sigma^*$ that associates to each variable $v \in \mathcal{V}$, a path $\mu(v) \in \Sigma^*$.
A **path expression** δ_e is a regular expression formed by using constants (*i.e., names of activities*), path variables and the classical quantifiers "?", " $*$ " and " $+$ ".
We extend the notion of valuations to path expressions by requiring that any valuation μ has to satisfy the following two constraints: (i) μ is the identity function on the elements of Σ, i.e., $\mu(\mathsf{a}) = \mathsf{a}$, $\forall\ \mathsf{a} \in \Sigma$, and (ii) for any path expressions a and b, we have $\mu(a.b) = \mu(a).\mu(b)$.
As an illustration, $\delta_e = CL.UI^*SO.CO.LS.v.$ is a path expression representing three distinguished execution paths of the BP of **Fig. 1**, i.e.,
$\delta_1 = CL.UI.SO.CO.LS.GD.OP.AO$, $\delta_2 = CL.UI.SO.CO.LS.SE.SR.UP$ and $\delta_3 = CL.UI.SO.CO.LS.SE.SR.GD.OP.AO$, corresponding respectively, to three separate valuations of the variable v, (i) $v = GD.OP.AO$, (ii) $v = SE.SR.UP$ and (iii) $v = SE.SR.GD.OP.AO$.
We say that an execution path δ satisfies a path expression δ_e, noted $\delta \models \delta_e$, if there is a valuation μ such that $\mu(\delta_e) = \delta$. In our example, the execution path $\delta_1 = CL.UI.SO.CO.LS.GD.OP.AO$ **satisfies** the path expression $\delta_e = CL.UI.SO.CO.LS.v$.
Now, let $Exec(\mathcal{P})$ be the set of all possible execution paths of $\mathcal{P}$, then we define a **selection rule** over $\mathcal{P}$ as a path expression $\delta_e(\mathcal{P})$ that can be used to query instances of the protocol $\mathcal{P}$. In this case, the set of answers of $\delta_e(\mathcal{P})$ includes the execution paths (*traces*) of $\mathcal{P}$ that satisfy δ_e. Hence, the selection rule is expressed as follows.

$$\delta_e(\mathcal{P}) = \{\delta \in Exec(\mathcal{P}) \mid \delta \models \delta_e\}. \tag{2}$$

As an example, let $\mathcal{P}$ be a business protocol, v a path variable and a an activity. The selection rule $\sigma_{(\mathcal{P})}$ formalized with the path expression $v.\mathsf{a}$ corresponds to any execution path ending with a. Hence, v could for example take as value the execution path $\mathsf{b.c.d}$ and the selection rule $\sigma_{(\mathcal{P})} = v.\mathsf{a}$ returns the execution path $\mathsf{b.c.d.a}$. This is obtained by a valuation μ that assigns to v the value $\mu(v) = \mathsf{b.c.d}$ and hence we obtain $\mu(v.\mathsf{a}) = \mu(v).\mu(\mathsf{a}) = \mathsf{b.c.d.a}$.
As another illustration, the selection rule $\sigma_{(\mathcal{P})} = \mathsf{a}.v.b^*.c$, with $v \in \mathcal{V}$ a path variable, enables to select any path starting with a and ending with an arbitrary sequence of b followed by c (e.g., the path $\mathsf{a.e.f.g.b.b.c}$).

Example 5. Basing on execution traces of **Table 1**, we can filter out instances having accomplished correctly the BP (*the two activities Goods delivered (GD) and Order Payment (OP) were performed* and the last activity in the execution trace is Archiving Order (AO). Such a query is formalized with the following selection rule: $\sigma_{(\mathsf{Order})} = v.\mathsf{AO}$ and its valuation returns the two instances $\mathcal{T}_4(Order)$ and $\mathcal{T}_9(Order)$ of **Table 1**.
Assume now that we are interested to select instances of the BP of **Fig. 1** having executed the activity Update Information (UI) at most once and for which the last activity is *(Goods Delivery (GD))*. Such a query can be expressed with the selection rule: $\sigma_{(\mathsf{Order})} = CL.(UI)^{?}.v.\mathsf{GD}$.
Two valuations μ that assign to v the values $\mu(v) = \mathsf{SO.CO.LS}$ and $\mu(v) = SO.CO.LS.SE.SR$ are possible. These valuations lead to two distinguished execution paths, respectively:
$CL.(UI)^{?}.SO.CO.LS.GD$ and $CL.(UI)^{?}.SO.CO.LS.SE.SR.GD$.
From **Table 1**, only $\mathcal{T}_7(Order)$=CL.SO.CO.LS.SE.SR.GD, with $(UI)^{?} = \epsilon$, meets the constraints imposed by the previous selection rule.

4 Implementation and Experiments

This section briefly describes a prototype, named **B**usiness **P**rocesses **D**ata **A**nalyzer (**BPDA**) which implements the proposed approach. First, we present the architecture and the functionalities performed by the system, then we give some examples of Cypher queries allowing to query execution data.

4.1 System Architecture and functionalities

Java language with the Eclipse development environment are used to implement the proposed approach. Further, the graph database management system Neo4j [20, 21], using Cypher [22, 23] as a query language, is deployed for analysing BP specifications and for querying execution data. Cypher is based on the Property Graph Model, which organizes data into nodes and edges, called "relationships". Such a graph model is perfectly suited to our process model (*an automaton can be perceived as a graph*) and, thus it's very suitable for the suggested formalization and developments.

As illustrated in **Fig. 2**, the prototype **BPDA** is organized around four main components (**A,B,C** and **D**) interacting with the conceived graph database (Protocols and Traces).
The first component(**A**) of **BPDA** is the Business Protocol Manager module which allows describing BP specifications as automaton (FSM). A graphical tool-box performs creating and updating the considered specifications and storing the generated descriptive data in the protocols database as a graphs (*graph database*). Furthermore, this module offers a conformance-checking tool which enables ensuring the verification of a set of correctness criteria (*existence of initial state, final states, absence of unreachable states ...*).

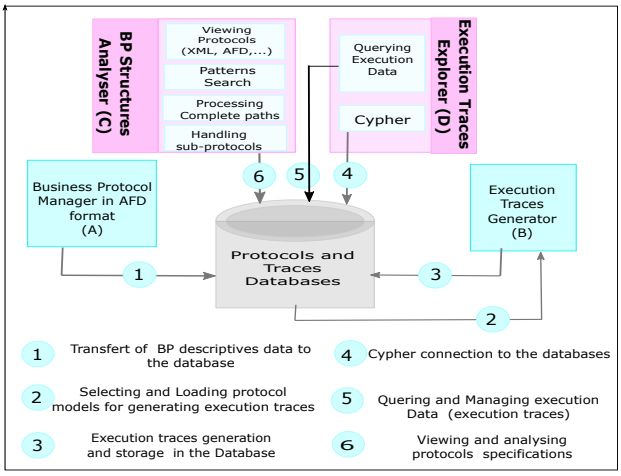

Fig. 2. Arhitecture and functionalities of the prototype BPDA

The second module Execution Traces Generator (**B**) is dedicated to execution traces generation. Thus, after having selected and loaded a BP protocol from the database, the system user can either generate individual traces by choosing activities to be performed or he can configure the number of instances to be generated randomly. These actions ensure populating the database with synthetic data needed for experiments reasons.
The third module BP Structures Analyser (**C**) constitutes the system kernel. It realizes the most contributions of the proposed approach, regarding the BP description level. Thus, the features of pattern search, processing complete execution paths and handling sub-protocols are handled by this component.
After generating the database, a last module Execution Traces Explorer (**D**) can be activated to query execution data by using path expressions (*see §3.3*). However, for users who are familiarized with the Cypher language, they can directly formulate simple queries on execution data stored in the graph database.

4.2 Using Cypher for exploring BP Data

Cypher is a declarative language allowing to query and update a graph. The most common defined functions are MATCH and WHERE. MATCH, mainly based on relationships, is used to describe the search model. While, WHERE is used for adding constraints to queries [24].
In what follows, we show different Cypher queries which allow querying data of the BP model of **Fig. 1** and its related execution traces of **Table 1**.

Example 6. **Extracting complete execution paths of the protocol Order**

```
MATCH p = (n:Etat type:'initial')-[rs:TRANSITION*]→(m:Etat type:'final')
WHERE (:Protocole nom: "Order") -[:CONTIENT]→(n)
RETURN extract(n IN rs — n.nom)
```

The previous query returns the following four complete execution paths:
["CL", "UI", "SO", "CO", "RO"]
["CL", "UI", "SO", "CO","LS", "SE", "SR", "UP"]
["CL", "UI", "SO", "CO","LS", "GD", "OP", "AO"]
["CL", "UI", "SO", "CO","LS", "SE", "SR", "GD", "OP", "AO"]

Now, we give some cypher queries for exploring data contained in **Table 1**.

Example 7. Various queries for exploring execution data

- R_1 : The number of instances that are still running (Status="A").
  MATCH (i:Instance { Status: 'A' })
  RETURN count(i)
  - The answer: 4.
- R_2 :: Which instances have started before 23/02/2020 ?
  MATCH (i:Instance)
  WHERE i.Start-date ⩽'23-02-2020'
  RETURN i

  -The query answer:
  - {"ID":110, "Trace Name":"$\mathcal{T}_2(Order)$", "Execution Trace":"CL.UI.UI.SO.CO.RO", "Start-date":"30/06/19","Start-time": "20:15:14", "T-stamp":"56", "Length":"6", "Status":C}
  - {"ID":288, "Trace Name":"$\mathcal{T}_4(Order)$", "Execution Trace":"CL.UI.SO.CO.LS.GD.OP.AO", "Start-date":"16/02/18 ","Start-time": "22:53:08", "T-stamp":"3600", "Length":"8", "Status":C}
  - {"ID":115, "Trace Name":"$\mathcal{T}_7(Order)$", "Execution Trace":"CL.SO.CO.LS.SE.SR.GD", "Start-date":"23/02/20", "Start-time":"16:17:18", "T-stamp":"3456", "Length":"7", "Status":A}
- R_3 :: Are there instances that have performed exactly 5 activities and which are blocked (Status="B")?
  MATCH (i:Instance)
  WHERE i.Length : '5' and Status: 'B'
  RETURN i

  -Only one instance satisfies constraints of the query: - {"ID":88, "Trace Name":"$\mathcal{T}_6(Order)$", "Execution Trace":"CL.UI.UI.UI.UI", "Start-date":"10/10/20", "Start-time":"01:15:22", "T-Stamp":"78", "Length":"5", "Status":B}

5 Conclusion and Future Work

In this work an approach for analysing BPs structures and for exploring execution data is formalized. Business processes are expressed as **FSM**, a set of specifications patterns is proposed to analyse abstract models and a selection

rule model is formalized to query the generated data.
Our contribution relates to two complementary aspects; (i) formal specification of protocols' structures allowing to convert the managers needs to the problem of handling automata and their relationships and (ii) Exploring business execution data by taking advantages of semi-structured and graph databases.
In future works, we plan to tackle the storage problem of data involved during BPs life-cycle, by comparing relational and graph databases and by deploying queries optimization techniques. As a potential direction, we project to experiment the proposed approach on big data originating from social networks.

References

1. C. Beeri, A. Eyal, T. Milo, and A. Pilberg. Monitoring business processes with queries. In *VLDB*, pages 603–614, 2007.
2. W. v. Aalst. Using process mining to bridge the gap between bi and bpm. *Computer*, 44:77–80, 12 2011.
3. Wil M. P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Publishing Company, Inc., 1st edition, 2011.
4. R. Ahmed, M. Faizan, and A. I. Burney. Process mining in data science: A literature review. In *2019 13th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS)*, pages 1–9, 2019.
5. Dusanka Dakic, Srdjan Sladojevic, Teodora Lolic, and D. Stefanovic. Process mining possibilities and challenges: A case study. *2019 IEEE 17th International Symposium on Intelligent Systems and Informatics (SISY)*, pages 000161–000166, 2019.
6. Jorge Saldivar, Carla Vairetti, Carlos Rodríguez, Florian Daniel, Fabio Casati, and Rosa Alarcón. Analysis and improvement of business process models using spreadsheets. *Information Systems*, 57:1–19, 2016.
7. Nathan D. Ryan and Alexander L. Wolf. Using event-based translation to support dynamic protocol evolution. ICSE '04, pages 408–417, 2004.
8. A. Azough, E Coquery, and M-S Hacid. Supporting web service protocol changes by propagation. WI-IAT '09, pages 438–441, 2009.
9. B. Benatallah, F. Casati, and F. Toumani. Web service conversation modeling: A cornerstone for e-business automation. *IEEE Internet Comp.*, 8(1):46–54, 2004.
10. B. Benatallah, F. Casati, and F. Toumani. Representing, analysing and managing web service protocols. *D.K.Eng.*, 58(3):327–357, 2006.
11. D. Berardi, F. Cheikh, G. De Giacomo, and F. Patrizi. Automatic service composition via simulation. *IJFCS*, 19(2):429–451, 2008.
12. Antonio Brogi and Sara Corfini. Behaviour-aware discovery of web service compositions. *Int. J. Web Service Res.*, 4(3):1–25, 2007.
13. C.A. Middelburg and M.A. Reniers. Introduction to process theory. Technical report, Technische Universiteit Eindhoven, 2004.
14. K. Klai, S. Tata, and Jörg Desel. Symbolic abstraction and deadlock-freeness verification of inter-enterprise processes. In *BPM*, 2009.
15. F. Cassez and Olivier-H. Roux. Structural translation from time petri nets to timed automata. *Electronic Notes in Theoretical Computer Science*. AVoCS 2004.
16. Xi Wang, Huaikou Miao, and Liang Guo. Towards automatic transformation from uml model to fsm model for web applications. *JSEA*, 1(1):68–75, 2008.
17. D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of e-services that export their behavior. In *ICSOC*, Dec. 2003.

18. S. H. Ryu, F. Casati, H. Skogsrud, B. Benatallah, and R. Saint-Paul. Supporting the dynamic evolution of web service protocols in service-oriented architectures. *ACM Trans. Web*, 2(2):1–46, May 2008.
19. Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the web: from relations to semistructured data and XML*. Morgan Kaufmann, San Francisco, 2000.
20. Justin Jay Miller. Graph database applications and concepts with neo4j. 2013.
21. Mahesh Lal. *Neo4j Graph Data Modeling*. Packt Publishing, 2015.
22. A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, Stefan P., M. Shuster, P. Selmer, and H. Voigt. Updating graph databases with cypher. *Proc. VLDB Endow.*, 12(12):2242–2254, August 2019.
23. Nadime Francis, Andrés Taylor, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, and Petra Selmer. Cypher: An evolving query language for property graphs. pages 1433–1445, 05 2018.
24. N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, M. Schuster, P. Selmer, and A. Taylor. Formal semantics of the language cypher. *CoRR*, abs/1802.09984, 2018.