

Semantic Modeling and Analysis of Complex Data-aware Processes and their Executions

Piergiorgio Bertoli^c, Francesco Corcoglioniti^b, Chiara Di Francescomarino^a,
Mauro Dragoni^{a,*}, Chiara Ghidini^a, Marco Pistore^{a,c}

^a*Fondazione Bruno Kessler*

^b*Independent Researcher*

^c*SayService*

Abstract

Enriching business processes with the semantic knowledge of an ontology is recognized as a fundamental need in business process management in order to obtain better models and perform more expressive execution analyses. In our work, we model semantically annotated business processes in an OWL knowledge base formalizing the structure of business processes together with the associated business domain. We extend previous work in three substantial manners: first, we show how to exploit the OWL knowledge base to represent the complex data artifacts manipulated by a business process (together with the process itself); second, we show how execution traces can be added into the OWL knowledge base to perform a process execution analysis involving knowledge about both the structure and the domain of a business process; third, we show how to support the semantic modeling and analysis of complex data-aware business processes and their executions. In detail, we illustrate how we extended the MoKi tool for enabling the collaborative modeling of semantically annotated complex data-aware processes, and for supporting their execution analysis via semantic-based reasoning techniques capable of scaling to non-trivial amounts of data while maintaining their advantages. The feasibility and usefulness of the proposed conceptual framework and tool support is shown through a real use case and an experimental evaluation.

Keywords: Semantic Business Process, Business Process Modeling, Semantic Technologies, Business Process Monitoring, Business Process Analysis

*Corresponding author

Email addresses: bertoli@sayservice.it (Piergiorgio Bertoli), francesco@corcoglioniti.name (Francesco Corcoglioniti), dfmchiara@fbk.eu (Chiara Di Francescomarino), dragoni@fbk.eu (Mauro Dragoni), ghidini@fbk.eu (Chiara Ghidini), pistore@fbk.eu (Marco Pistore)

1. Introduction

Semantic Business Process Management (Hepp et al., 2005; Wetzstein et al., 2007) aims at increasing the degree of automation in specifying, executing, and monitoring business processes by enhancing business process management tools with the most important results from the Semantic Web field.

When the focus is on *process modeling*, i.e., the task of specifying a business process at a descriptive level, the annotation of a process description with labels from domain ontologies (hereafter called domain knowledge) is perceived useful to obtain better models. A clear demonstration of this is the usage of domain knowledge to provide additional semantics to elements of process models to support, for instance: the creation of valid diagrams (Di Francescomarino et al., 2009); the alignment with natural language (Thomas and Fellmann, 2007); the mapping and translation of process models (Antunes et al., 2015; Batoulis et al., 2013); the management of exceptional flows (Ghidini et al., 2012a); and the automated composition of process models (Dimitrov et al., 2007). A further evidence is provided by the fact that terms such as Semantic Process Modeling are being introduced in the research community, along with specific challenges that concern them (Mendling et al., 2015). Despite the different approaches, all the work above mainly uses ontologies for annotating the elements of a Business Process Diagram (from now on BPD), with particular emphasis on the control flow, in order to formalize and exploit the meaning of BPD labels. Very little work exists that exploits the domain knowledge contained in an ontology to characterize also the structure of those entities (e.g., data objects) that complement the typical control flow elements and whose description is usually rather abstract in a BPD.

When it comes to *process analysis*, i.e., the task of analyzing actual executions (cases) of a process, the contribution of semantic based technology is not as mature. While in their visionary work Hepp et al. (2005), Pedrinaci et al. (2008b, 2009), and Wetzstein et al. (2007) have argued that semantic knowledge is a necessary complement to the procedural semantics of process execution in order to perform process analysis at a business-level, very little has happened since then and we are still far from devising a semantic process analysis. Though the notion of semantic process analysis would deserve a rigorous definition, intuitively we can follow Wetzstein et al. (2007) and argue that it is a scenario where analysts specify, at modeling time, Process Performance Indicators (PPI) involving knowledge about both the structure and the domain of a business process, and get results in form of dashboards during process execution. For instance, a PPI of an administrative process asking for *the average time consumed and the number of executions of optional activities (i.e., activities that given a path in the process model may be executed or not) carried out by each public office in a birth management process* requires knowledge about the procedural semantics of the process execution together with domain knowledge that specifies the meaning of *public office* and permits to reason on that. For instance, execution traces may contain events performed by a *registry office*. Knowing, and exploiting the fact, that a registry office is a public office, is a simple example of what semantic

knowledge can add to traditional process analysis.

In this paper we aim at advancing the state of the art in semantic modeling and analysis of complex data-aware processes and their executions in three different directions:

- first, we extend the ontology-based architecture previously presented by Di Francescomarino et al. (2009) in order to exploit the OWL¹ (W3C OWL Working Group, 2012) knowledge base not only to annotate entities in a business process model but also to provide a specification of the complex data artifacts manipulated by the process itself (Section 3). This is done to obtain a richer process model description which complements and encompasses the traditional control flow centered process model description of languages such as BPMN² (Object Management Group, 2008, 2011);
- second, we show how execution traces can be added into the ontology-based architecture in order to perform a process execution analysis that involves knowledge about both the structure and the domain of a business process. This produces a first evidence of how to perform semantic based process analysis (Section 4);
- third, we show how to support the ontology-based architecture with a tool for both (i) the modeling of semantically annotated complex data-aware business processes (Section 5), and (ii) the analysis of their execution via semantic-based reasoning techniques (Section 6). The front-end of the tool extends MoKi, a state-of-the-art online tool for the collaborative modeling of semantically annotated business processes, and complements it with the support for the description of the complex data objects manipulated by (BPMN) business processes as well as for the definition of PPIs and the visualization of their results. The back-end of the tool is based on triplestores and other technologies from the Semantic Web field. It aims at supporting the PPI based process analysis, managing the large amounts and the high rates of process execution data that characterizes real-world application scenarios, as well as at providing the ability to make semantic based reasoning and inferences on that data.

In Bertoli et al. (2014), we carried out an initial investigation of the integration of the procedural dimension, the domain-specific dimension and the execution dimension into an ontology-based architecture. In this paper we significantly extend our previous work by:

- explicitly defining a general ontology-based architecture which integrates the procedural, domain-specific and execution dimensions, bearing in mind general scenarios dealing with complex data-aware processes and their executions;

¹<http://www.w3.org/TR/owl2-overview/>

²<http://www.bpmn.org/>

- offering tool support for managing the ontology-based architecture, by providing users with instruments for modeling complex data-aware business processes and analyzing them by means of PPIs;
- significantly extending the evaluation of the proposed approach by: (i) enriching the previous quantitative evaluation to better analyze storage, reasoning and querying performance with more data and additional PPIs with different characteristics; and (ii) adding a qualitative evaluation focused on the tool support.

The tool, the code, as well as all the material used in the evaluation, are available online.³

Besides the core sections (Sections 3 to 6), in the rest of the paper we introduce a real use case in the Public Administration domain (Section 2) and use it throughout the paper to illustrate our approach and perform the qualitative and quantitative evaluation of its effectiveness (Section 7), completing the paper with a discussion of related works (Section 8) and some concluding remarks (Section 9).

Note that the ontology-based architecture and its encoding in a triplestore for supporting process analysis are orthogonal to the specific modeling environment we use. On the one hand, they are not limited to MoKi but realize an original contribution on their own as they can be paired with any front end modeling tool that exports/converts the process model specification in terms of our ontological model. On the other hand, their combination with MoKi permits business analysts to perform process modeling and analysis in a uniform manner within a single modeling environment, which we believe to be a further contribution of our work.

2. A Motivating Example

We introduce a concrete use case – the Italian birth management process – that we will use throughout the paper to clarify the motivations and the ideas behind the presented framework. The use case originates from an industrial case study in the Public Administration (PA) field, investigated as part of the *ProMo* project⁴ whose aim was the semantic modeling, monitoring, and analysis of business processes. The example was chosen as the PA is a paradigmatic example of a complex organization, characterized by different dimensions that need to be carefully taken into account and reconciled when analyzing processes and their executions.

Figure 1 shows the Italian birth management process model in the BPMN notation. The process aims at recording the birth data of newborn Italian

³<https://mokipromo.fbk.eu/>

⁴“ProMo - A Collaborative Agile Approach to Model and Monitor Service-Based Business Processes” was a project funded by the Province of Trento, Italy, within the Operational Programme “Fondo Europeo di Sviluppo Regionale” (FESR) 2007-2013.

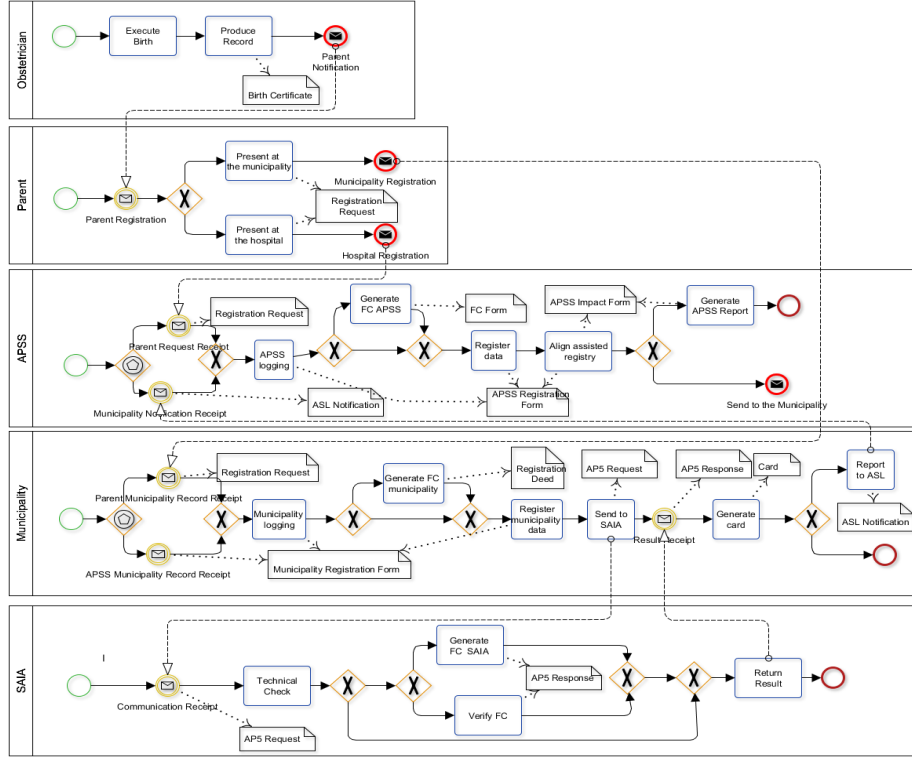


Figure 1: Birth Management Process.

citizens, who have to be registered: (i) in the registries of the local municipality and the central administration (SAIA); and (ii) in the national welfare system repository (APSS). The two registrations can occur in any order depending on the choice of the newborn's parents, require the coordination of many actors, and involve the attribution of some important personal data to the newborn, such as his/her Fiscal Code (FC) identifier (Italian code akin to U.S. SSN).

In this kind of scenarios characterized by large scale application and high public costs, a crucial requirement is the capability to analyze actual process executions for maintenance purposes and for realizing informative what-if analyses. This means being able to understand how a process performs, where, when, and how many bottlenecks and failures occur, and whether and how process executions deviate from the designed model. To that respect, PPIs are an important instrument to specify relevant performance requirements and determine whether strategic goals are met by an organization. Examples of PPIs defined by business analysts for the considered birth management process are:

PPI.1 the average time spent per process execution by a certain municipality;

PPI.2 the number of Registration Request documents processed since January 1st, 2018;

PPI.3 the percentage of process executions where the flow passed first in the APSS pool and then in the municipality pool;

PPI.4 the average time spent and the number of executions of optional activities by each public office involved in the process.

In order to be able to compute these PPIs, we need to take into account different aspects. For instance, to compute PPI.1, we need to understand what a municipality is and, for each municipality, identify the executions that refer to that municipality. In the case of PPI.2, we have to understand what a Registration Request document is, as well as identify the executions processing that type of document and when the document is processed. For PPI.3, we have to look at the process control flow in order to identify the paths passing through the APSS pool first and the municipality pool then and collect the corresponding process executions. Finally, to compute PPI.4, we have to understand what a public office is, i.e., we have to understand that APSS, Municipality and SAIA are all public offices, identify for each of them the control flow and combine these results with the actual executions in order to be able to compute the average time.

The need for taking into account different aspects when computing PPIs is not limited to the case of these examples. As noted by van der Aa et al. (2016), PPIs often refer to, and therefore require the semantics of, model elements such as data and role objects in addition to the more traditional flow objects. This happens also in the case of PPI.1–PPI.4, where we need to take into account three dimensions: (i) the procedural dimension $\mathbf{P}$; (ii) the static knowledge dimension $\mathbf{K}$; and (iii) the execution dimension $\mathbf{T}$ (see Figure 2). The *procedural dimension* $\mathbf{P}$ is defined by the flow objects in the process model. Such a dimension is required, e.g., in PPI.3 for detecting where the execution flow goes first: APSS or municipality. The *static knowledge dimension* $\mathbf{K}$ describes the non-procedural domain knowledge (e.g., data objects and role objects) that refers to the carrying on of the process. This dimension enables identifying, for example, which documents are Registration Requests in PPI.2. The *execution dimension* $\mathbf{T}$ refers to all the data produced by actual process executions, such as the process execution trace and all the data object instances manipulated in its context. This dimension allows for retrieving, for example, the actual execution time of PPI.1.⁵ Thus, managing a scenario such as the one in Figure 1, and finding an answer to PPI.1–PPI.4 poses interesting challenges both in terms of back-end algorithms, able to manage all three dimensions, and front-end tools, able to acquire and exploit the knowledge contained in $\mathbf{K}$, $\mathbf{P}$ and $\mathbf{T}$.

Note that, a further source of complexity in the integrated representation of $\mathbf{K}$, $\mathbf{P}$ and $\mathbf{T}$ is provided by the fact that these dimensions belong to two different

⁵We assume that execution data has already been extracted from heterogeneous IT systems and reconciled in coherent execution traces. These tasks, out-of-scope here, can be tackled with existing approaches in the literature that preprocess IT data into execution traces – e.g., as proposed by Bertoli et al. (2012) – and align those traces with the business layer – e.g., via semantic-based (Pedrinaci et al., 2008b) or structural (De Leoni et al., 2012) approaches.

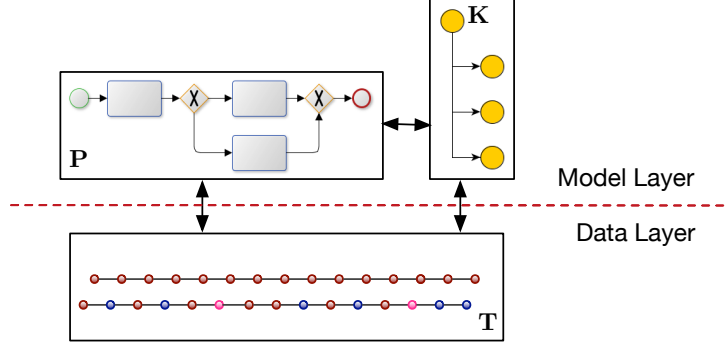


Figure 2: Combining $\mathbf{P}$, $\mathbf{K}$ and $\mathbf{T}$.

layers: the *model layer* ($\mathbf{K}$ and $\mathbf{P}$) that concerns the design time definition of a process, and the *data layer* ($\mathbf{T}$) that concerns the run time process execution, as emphasized in Figure 2. In the next section we start by illustrating how to construct a combined representation of $\mathbf{K}$ and $\mathbf{P}$ to model the knowledge contained in the model layer in an integrated manner. We then continue in Section 4 by integrating the model and the data layers.

3. Representing the $\mathbf{K}+\mathbf{P}$ Model Layer

The first step towards the integration of $\mathbf{K}$, $\mathbf{P}$ and $\mathbf{T}$, whose ontologies are further detailed in the online material,³ is the combination of the two types of knowledge in the model layer, namely the static knowledge $\mathbf{K}$ and the procedural knowledge $\mathbf{P}$.

Several works in the literature (e.g., Weber et al. 2008; Wong and Gibbons 2009; Koschmider and Oberweis 2005; Dimitrov et al. 2007; Thomas and Fellmann 2007; Di Francescomarino et al. 2008; Tomasi et al. 2012) inject semantic knowledge into process models, thus combining $\mathbf{P}$ with $\mathbf{K}$ either for specifying the execution semantics of the process models or for formalizing the meaning of the model elements. An example of typical combination of $\mathbf{K}$ and $\mathbf{P}$, which enriches the components of the BPD with the semantic knowledge, can be seen with the help of the birth management process example. In this example, the APSS pool of Figure 1 is annotated with the role *APSS*, characterized semantically as a *Public_Office* in the role taxonomy of the domain ontology shown in the excerpt of Figure 4b.

The combination we propose is based upon two ontology modules (one for $\mathbf{K}$, and one for $\mathbf{P}$) which are expressed in a single formal language: OWL-DL, that is the Description Logics (DL, Baader et al. 2009) fragment of OWL. This combination, graphically depicted in Figure 3, enables the semantic annotation of BPDs expressed in the BPMN modeling language and the further specification of the structure of data objects (with particular emphasis on documents) manipulated by the process itself.

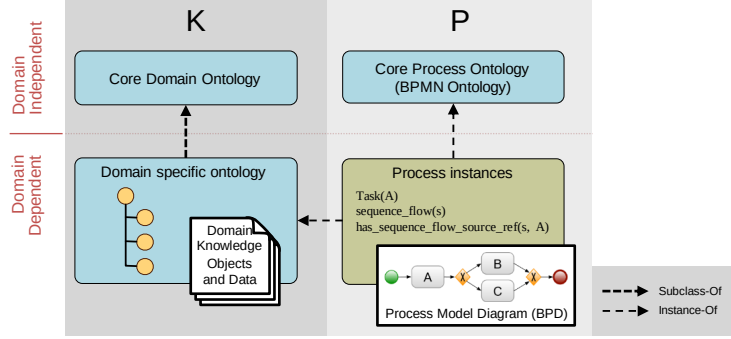


Figure 3: Combining **K** and **P** into a modular ontology.

The two ontologies (possibly further divided into modules) provide a *Domain Ontology* and a *Process Ontology*, which aim at formalizing respectively the static aspects of the domain (i.e., **K**) and the procedural aspects (i.e., **P**). A detailed description of the two ontologies and of their integration in a single conceptual model is provided by Di Francescomarino et al. (2009). Here, we provide a high-level description of the main aspects of the ontologies and of the architectural model required to understand the overall solution, including the extension towards the specification of the data objects.

3.1. The Domain Ontology

The Domain Ontology, whose responsibility is to formalize the static aspects of the domain, consists of an upper-level *Core Domain Ontology* that defines the high-level concepts in a domain-independent way, and a *Domain-Specific Ontology* that captures the concepts specific to a particular domain. For both ontologies, concepts are defined in terms of classes and properties using OWL-DL terminological (T-Box) statements as briefly described below.

The Core Domain Ontology, shown in Figure 4a, defines the key entities and relations that characterize the domain of a generic business process, organized in a framework that can be extended to model domain-specific concepts. In particular, it models the concept of *Activity* as the building block of a *Process*, carried out by an executor with a certain *Role* taken from a hierarchy of roles. Activities produce *Data_Objects* (for instance, a document) modeled as sets of *Data_Fields*, which, in turn, can be *Complex_Fields* composed of (nested) fields (e.g., a structured record) or *Simple_Fields*, i.e., fields at the end of a nesting chain that do not contain any nested field.

As the Core Domain Ontology is independent of the domain, it mainly contains classes and axioms related to high-level concepts, as shown in Table 1 and

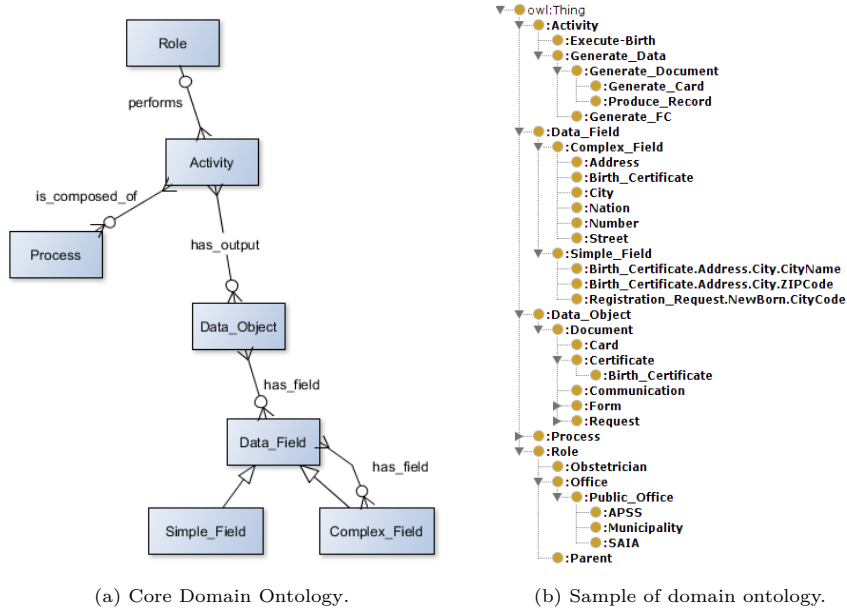


Figure 4: The Domain Ontology.

summarized in column 2 of Table 2⁶ An example of class axiom is given by:

$$\text{Process} \sqsubseteq \exists \text{ is_composed_of. Activity} \quad (1)$$

specifying that a **Process** is composed of at least one **Activity**. Another example of class axiom details the relationship between **Activity** and **Role**:

$$\text{Activity} \sqsubseteq (\leq 1) \text{ performs}^{-1}. \text{Role} \quad (2)$$

The axiom specifies that an **Activity** can be performed at most by one **Role**.

The Domain-Specific Ontology, shown in Figure 4b for the considered birth management process, provides a domain-dependent specialization of the concepts (classes, properties) of the Core Domain Ontology. For instance, the ontology contains activities, roles, and data objects such as **Produce_Record**, **Obstetrician**, and **Birth_Certificate** that refer to activities, roles, and data objects of the BPD and organize them semantically to enable reasoning and facilitate process analysis (e.g., by formalizing that a **Birth_Certificate** is-a **Certificate**, which is-a **Document**, which is-a **Data_Object**, see Figure 4b).⁷

⁶The Core Domain Ontology, the BPMN Ontology and the Core Trace Ontology are available online at <https://mokipromo.fbk.eu/>.

⁷For the sake of simplicity we use here concept names that syntactically match the process labels, so we have both a **Birth_Certificate** class in the Domain Ontology and a data object labeled with **Birth_Certificate** in the BPD (the latter being individual in the corresponding

Class:	Activity
Class:	Complex_Field
Class:	Data_Field
Class:	Data_Object
Class:	Process
Class:	Role
Class:	Simple_Field
Object Property:	has_field
<i>Domain:</i>	has_field has domain Complex_Field $\sqcup$ Data_Object
<i>Range:</i>	has_field has range Data_Field
Object Property:	has_output
<i>Domain:</i>	has_output has domain Activity
<i>Range:</i>	has_output has range Data_Object
Object Property:	is_composed_of
<i>Domain:</i>	is_composed_of has domain Process
<i>Range:</i>	is_composed_of has range Activity
Object Property:	performs
<i>Domain:</i>	performs has domain Role
<i>Range:</i>	performs has range Activity
Axiom:	Activity $\sqsubseteq (\leq 1)$ performs ⁻¹ .Role
Axiom:	Complex_Field $\sqsubseteq$ Data_Field
Axiom:	Complex_Field $\sqsubseteq \exists$ has_field.Data_Field
Axiom:	Data_Field $\sqsubseteq \exists$ has_field ⁻¹ .Data_Object
Axiom:	Data_Object $\sqsubseteq \exists$ has_output ⁻¹ .Activity
Axiom:	Data_Object $\sqsubseteq \exists$ has_field.Data_Field
Axiom:	Process $\sqsubseteq \exists$ is_composed_of.Activity
Axiom:	Role $\sqsubseteq \exists$ performs.Activity
Axiom:	Simple_Field $\sqsubseteq$ Data_Field

Table 1: Core Domain Ontology axioms.

Besides the concepts relevant to the specific scenario at hand, the Domain Ontology is used to specify the fine grained structure of data objects. We illustrate this with the help of an example. For instance, the **Address** (of a newborn) may be a data field in the **Birth_Certificate** data object. In turn, the **Address** field may be structured in the **Nation**, **City**, **Street**, and **Number** nested fields. The **City** field could be further organized in terms of **ZIPCode** and **CityName**, and this process can be reiterated until reaching the simple data fields of the data object, as in the case of **Birth_Certificate.Address.City.ZIPCode** and **Birth_Certificate.Address.City.CityName** (see Figure 4b).

ontology, see Section 3.2). As it will become clearer in Section 5.1, this naming and “mirroring” strategy is not strictly required by the approach as, for instance, the modeler may here omit the **Birth_Certificate** class and include only its parent **Certificate**, if it is enough to encode the terminological knowledge about these data objects. We also do not address here the problem of how to build the domain specific ontology. Again, this will be addressed in Section 5.1.

Feature	Value		
	Core Domain Ontology	BPMN Ontology	Core Trace Ontology
DL Expressivity	$\mathcal{ALCTIQ}$	$\mathcal{ALCHOIN}(\mathcal{D})$	$\mathcal{ALETIQ}(\mathcal{D})$
Classes	7	117	7
Object Properties	4	123	6
Datatype Properties	0	48	3
Individuals	0	104	0
Class Axioms	9	463	12
Object Property Axioms	8	236	12
Datatype Property Axioms	0	96	6
Individual Axioms	0	250	0
Annotation	0	504	0

Table 2: Ontology metrics.

3.2. The Process Ontology

The Process Ontology is responsible of formalizing the procedural aspects of the domain. It consists of a domain-independent core *BPMN Ontology* that defines the structure of a generic business process model based on BPMN, and of a domain-dependent *BPD Ontology* that represents a specific BPD described using BPMN. As such, it focuses on the description of the (mainly graphical) elements of a BPD and their connections.⁸ The BPMN Ontology is modeled at the level of classes and properties using a set of OWL-DL T-Box statements, while the BPD Ontology is modeled as a set of OWL-DL assertional (A-Box) statements on individuals in the language defined by the BPMN Ontology, as briefly described below.

The BPMN Ontology was originally proposed by Di Francescomarino et al. (2009). It formalizes the structural part of a BPD in the ontology language OWL, and its textual description in the formalism of Description Logics is provided in Ghidini et al. (2008). Therefore, its full description is out of the scope of this paper. We only recap here: (i) the main characteristics of the ontology (see Table 2, third column); (ii) the main steps we did follow to encode the Specification of BPMN v1.1 in the ontology; and (iii) how the BPMN Ontology is used to represent a BPD by means of its A-Box.

To encode the Specification of BPMN v1.1, we followed the three-step modelling process described in Rospocher et al. (2014). First, we manually processed the Specification of BPMN v1.1 to identify all the elements of the language and

⁸As remarked by Rospocher et al. (2014), the BPMN Ontology does not aim at modeling the dynamic behavior (i.e., how the flow proceeds) of BPDs.

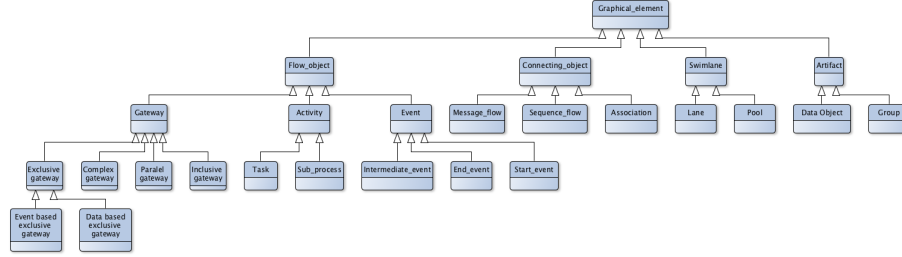


Figure 5: The BPMN taxonomy (excerpt).

we associated each of these elements to a class in the ontology. Furthermore, guided by the classification of these elements provided by the BPMN Specification, we formalized the initial taxonomy of the ontology classes encoding the BPMN elements. An excerpt of the upper part of the taxonomy of graphical elements of the BPMN Ontology is reported in Figure 5. It represents the elements of a generic BPD such as activity, gateway, sequence_flow, message_flow, swimlane and so on. It also states that flow_object has three subclasses, namely event, activity and gateway.

Second, for each of the BPMN elements we considered its attributes in the BPMN Specification and we formalized it either as a datatype property or an object property (see Rospocher et al. (2014) for details). We also did encode the multiplicity details as an OWL cardinality restriction on the class having the attribute. For example, the (0..1) multiplicity of the **State** attribute of BPMN element **DataObject**, reported in **Table 9.42 - Data Object Attributes** at (Object Management Group, 2008, p. 94) is formalized in the BPMN Ontology as the OWL cardinality restriction:

$$\text{DataObject} \sqsubseteq (\leq 1)\text{hasState} \quad (3)$$

When necessary, further conditions ruling the usage of the attribute are expressed by means of suitable axioms. For example the condition

$$\text{“If the result of an End Event is an Event Detail of type Error, then the Error Code for the Event Detail MUST be supplied.”} \quad (4)$$

is formalized as

$$\text{EndEvent} \sqsubseteq \neg\text{hasResult.Error} \sqcup \text{hasResult.}(\text{Error} \sqcap \exists\text{hasErrorCode}) \quad (5)$$

In the third step we formalized the conditions concerning the usage of the BPMN elements to compose a BPD. Two examples that concern the usage of Start Event and Connecting Object are:

$$\text{“A Start Event MUST be a source for Sequence Flow.”} \quad (6)$$

whose formalization is the following:

$$\text{start_event} \sqsubseteq \exists \text{has_connecting_object_source_ref}^{-1}.\text{sequence_flow} \quad (7)$$

and

$$\text{“A Connecting Object originates ONLY from Intermediate Event OR Start Event OR Task OR Sub Process OR Gateway”} \quad (8)$$

whose formalization is the following:

$$\begin{aligned} \text{sequence_flow} \sqsubseteq & \forall \text{has_connecting_object_source_ref} . (\text{intermediate_event} \\ & \sqcup \text{start_event} \sqcup \text{task} \sqcup \text{sub_process} \sqcup \text{gateway}) \end{aligned} \quad (9)$$

As a result of this process, the properties of each BPMN element, and its usage in composing a BPM are described in the BPMN Ontology⁹. An example is provided by the formalization of the BPMN element Gateway, and its attribute GatewayType having values among Exclusive, Inclusive, Complex (see **Table 9.32 - Common Gateway Attributes** at (Object Management Group, 2008, p. 71)), which is reported in Table 3. Similar specifications are provided for all the other BPMN elements and properties.

The encoding of a BPD is done using an A-Box consisting of individuals, and assertions on those individuals, using the classes and properties defined by the BPMN Ontology. A detailed description of how to represent a BPD as an A-Box is provided by Rospocher et al. (2014). We recall here the basic idea with the help of a simple example in Figure 6a, whose encoding (limited to the graphical elements and connections) is shown in Figure 6b¹⁰.

The *BPD individuals* in the A-Box of Figure 6b represent the graphical objects of the BPD (indicated also in the BPD in Figure 6a for the sake of the example). The assertions involving these individuals in Figure 6b belong to two groups: *type assertions* and *structural assertions*. The former indicate the type of graphical element an individual belongs to. For example, the type assertion $\text{sequence_flow}(s_1)$ states that the BPD individual s_1 is a *sequence_flow*. The latter indicate how the graphical objects are connected to create a BPD. For example, the structural assertion $\text{has_sequence_flow_source_ref}(s_1, \text{start})$ states that the source of the sequence flow s_1 is the (start) event start , while the structural assertion $\text{has_associated_pool}(\text{start}, o)$ ¹¹ states that the (start)

⁹Due to expressiveness limitation imposed by Description Logics and by the fact that we wanted to remain in a decidable fragment of OWL, there is a limited number of conditions described in the BPMN Specification which are not represented in the BPMN Ontology. This limited number of conditions is documented in Rospocher et al. (2014).

¹⁰Please note that the Description Logic (DL) notation used for the encoding in the reported examples is equivalent to the ontology representation in OWL.

¹¹For the sake of simplicity we introduce here a property *has_associated_pool* which is obtained by chaining the native roles *has_pool_process_ref* and *has_process_graphical_element* of the BPMN ontology. These two roles connect a pool and the process it contains, and a process and its graphical elements, respectively.

Class:	gateway
Class:	gateway_types
Object Property:	has_gateway_gateway_type
<i>Domain:</i>	has_gateway_gateway_type has domain gateway
<i>Range:</i>	has_gateway_gateway_type has range gateway_types
Instance:	<i>exclusive</i>
Instance:	<i>inclusive</i>
Instance:	<i>parallel</i>
Instance:	<i>complex</i>
Axiom:	gateway $\sqsubseteq$ ($= 1$)has_gateway_gateway_type
Axiom:	gateway_types $\equiv \{exclusive, inclusive, complex, parallel\}$
Axiom:	$(\neg\{exclusive\})(inclusive)$
Axiom:	$(\neg\{exclusive\})(complex)$
Axiom:	$(\neg\{exclusive\})(parallel)$
Axiom:	$(\neg\{inclusive\})(complex)$
Axiom:	$(\neg\{inclusive\})(parallel)$
Axiom:	$(\neg\{complex\})(parallel)$
Axiom:	exclusive_gateway $\equiv$ gateway $\sqcap \exists$ has_gateway_gateway_type. $\{exclusive\}$
Axiom:	inclusive_gateway $\equiv$ gateway $\sqcap \exists$ has_gateway_gateway_type. $\{inclusive\}$
Axiom:	parallel_gateway $\equiv$ gateway $\sqcap \exists$ has_gateway_gateway_type. $\{parallel\}$
Axiom:	complex_gateway $\equiv$ gateway $\sqcap \exists$ has_gateway_gateway_type. $\{complex\}$

Table 3: Gateway and GatewayType.

event e belongs to the pool o .¹² The encoding of BPDs as individuals and assertions instantiating the BPMN Ontology allows implementing different reasoning services, such as query answering on BPDs and compliance checking of BPDs against the BPMN Specification, as illustrated by Di Francescomarino et al. (2009).

3.3. Integrating $\mathbf{K}$ and $\mathbf{P}$

We now recall Figure 3 and show how its components described previously are linked in an integrated ontology that combines the dimensions $\mathbf{K}$ and $\mathbf{P}$.

The upper part of Figure 3 contains the domain independent representation of a scenario, namely the Core Domain Ontology (dimension $\mathbf{K}$), and the BPMN Ontology (dimension $\mathbf{P}$). They are the building blocks of the integrated ontology at the domain-independent level.

The lower part of Figure 3 provides a domain-specific modeling in terms of the Domain-Specific Ontology (dimension $\mathbf{K}$) and the process instances of the BPD Ontology (dimension $\mathbf{P}$). *Subclass-Of* (is-A) relationships link the concepts of the Domain-Specific Ontology to the ones of the Core Domain Ontology that they specialize, as discussed in Section 3.1 and exemplified in Figure 4b. *Instance-Of* relationships link instead the individuals in the BPD Ontology to

¹²See <http://selab.fbk.eu/difrancescomarino/SemanticBPM/> for examples of instantiated BPDs.

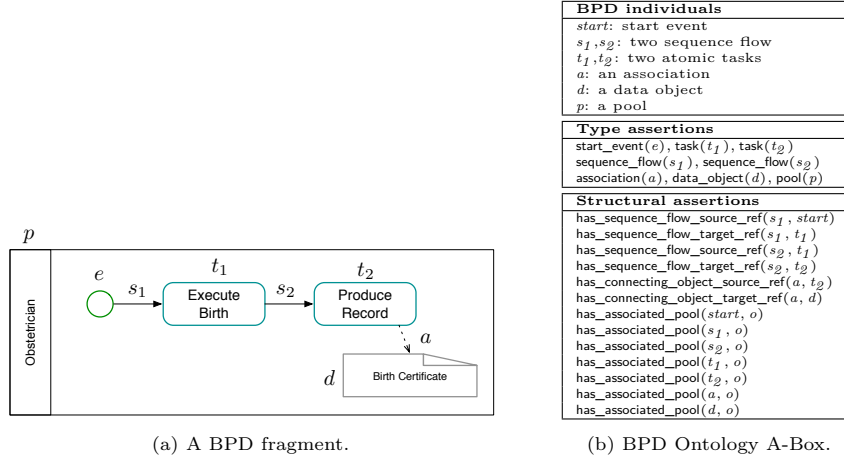


Figure 6: Encoding a BPD fragment in the BPMN Ontology.

the concepts of the BPMN Ontology they instantiate, as discussed in Section 3.2 and exemplified in Figure 6b.

The connection between **K** and **P**, defined by the semantic annotations of the BPD, is encoded in our ontological model via *Instance-Of* relationships stating that the individuals of the BPD Ontology are instances of specific concepts of the Domain-Specific Ontology (more generally, of the Domain Ontology). For instance, expressing the fact that the task t_2 (labeled “Produce Record”) in the BPD Ontology (see Figure 6a) is an activity of type `Produce_Record` in the Domain-Specific Ontology (see Figure 4b) is done by adding an assertion

$$\text{Produce_Record}(t_2) \quad (10)$$

to the A-Box in Figure 6b.¹³ Similarly, the assertion

$$\text{Birth_Certificate}(d) \quad (11)$$

can be used to specify that the “Birth Certificate” data object d in the BPD Ontology is a document of type `Birth_Certificate` in the Domain-Specific Ontology. Note that, in this case, the annotation of the BPD element with a concept of the Domain Ontology enriches d with the structure of a birth certificate object as specified in the Domain Ontology. For instance we now know that d must contain a complex field `Address` further structured in `Street`, `Number`, `City`, `Nation` and so on, as reported in Section 3.1.

¹³Further semantic annotation examples are provided by Di Francescomarino et al. (2008, 2009).

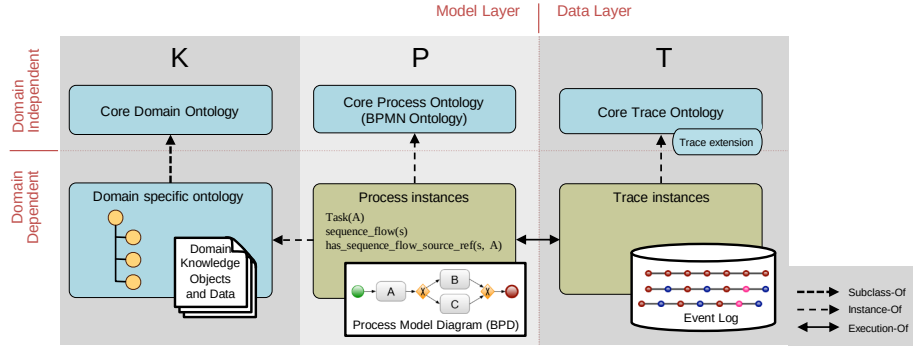


Figure 7: Combining **K**, **P**, and **T** into a modular ontology.

4. Adding the **T** Data Layer

The second step towards the integration of **K**, **P** and **T** is the integration of the data coming from process executions (that is, the **T** dimension) into the OWL-DL representation of **K** and **P** illustrated in the previous section. This integration enables the incorporation in the ontology of the actual data that constitute the actual enactment of the business process, such as event occurrences and data object instances.

The integration we propose is based upon adding a new OWL-DL *Trace Ontology* module for **T** to the structure for **K** and **P**, as graphically illustrated in Figure 7. The Trace Ontology aims at providing a formal description of execution traces. We summarize here the main aspects of the Trace Ontology and then present its integration with the **K** and **P** components.

4.1. The Trace Ontology

The Trace Ontology is modeled in terms of two T-Box parts – *Core Trace Ontology* and *Trace Extension Ontology* – that contain the fundamental domain-independent and dependent concepts, plus an A-Box part – the *Trace Instance Ontology* – that encodes the specific execution traces of an event log.

The Core Trace Ontology, whose Entity-Relationship representation is reported in Figure 8a (blue entities), and its main characteristics are reported in Table 2, fourth column, aims at providing a general representation of the knowledge that may be traced by IT systems and organized in execution traces. Its key concepts *Trace*, *Event*, and *Log* are modeled taking inspiration from the eXtensible Event Stream (XES)¹⁴ standard for representing event logs based on XML (Verbeek et al., 2010). Since we are interested in events as much as in the domain entities that refer to them – and in particular in the organizational roles performing events, in the data objects created or modified by events, as

¹⁴<http://www.win.tue.nl/ieetfpm/lib/exe/fetch.php?media=shared:downloads:xes-r12d2.pdf>

Class:	Data_object	Data Property:	event_name
Class:	Event	<i>Domain:</i>	event_name has domain Event
Class:	Field	<i>Range:</i>	has_performer has range STRING
Class:	Log	Data Property:	timestamp_end
Class:	Performer	<i>Domain:</i>	timestamp_end has domain Event
Class:	Process_Element	<i>Range:</i>	timestamp_end has range DATETIMESTAMP
Class:	Trace	Data Property:	timestamp_start
Object Property:	contains_data_object	<i>Domain:</i>	timestamp_start has domain Event
<i>Domain:</i>	contains_data_object has domain Trace	<i>Range:</i>	timestamp_start has range DATETIMESTAMP
<i>Range:</i>	contains_data_object has range Data_object	Axiom:	Data_object $\sqsubseteq (= 1)$ has_data_object ⁻ Event
Object Property:	contains_event	Axiom:	Data_object $\sqsubseteq (= 1)$ contains_data_object ⁻¹ .Trace
<i>Domain:</i>	contains_event has domain Trace	Axiom:	Data_object $\sqsubseteq$ Process_element
<i>Range:</i>	contains_event has range Event	Axiom:	Data_object $\sqsubseteq \exists$ has_data_object_field.Field
Object Property:	contains_trace	Axiom:	Event $\sqsubseteq$ Process_element
<i>Domain:</i>	contains_trace has domain Log	Axiom:	Event $\sqsubseteq (= 1)$ contains_event ⁻¹ .Trace
<i>Range:</i>	contains_trace has range Trace	Axiom:	Event $\sqsubseteq (\leq 1)$ has_performer.Performer
Object Property:	has_data_object	Axiom:	Field $\sqsubseteq (= 1)$ has_data_object_field ⁻¹ .Data_object
<i>Domain:</i>	has_data_object has domain Event	Axiom:	Log $\sqsubseteq \text{exists}$ contains_trace.Trace
<i>Range:</i>	has_data_object has range Data_object	Axiom:	Performer $\sqsubseteq \exists$ has_performer ⁻¹ .Event
Object Property:	has_data_object_field	Axiom:	Trace $\sqsubseteq \exists$ contains_event.Event
<i>Domain:</i>	has_data_object_field has domain Data_object	Axiom:	Trace $\sqsubseteq (= 1)$ contains_trace ⁻¹ .Log
<i>Range:</i>	has_data_object_field has range Field		
Object Property:	has_performer		
<i>Domain:</i>	has_performer has domain Event		
<i>Range:</i>	has_performer has range Performer		

Table 4: Core Trace Ontology axioms.

well as in the data object fields written or updated by the events – we represent them directly as concepts **Performer**, **Data_Object** and **Field** in the core trace ontology. Since we consider events with a duration, we model the timestamp by means of a start and an end event timestamp. Instantaneous events, and BPMN events, can be modeled as a special case with start and end event timestamps coinciding.¹⁵ Moreover, as in XES, we use the *event_name* attribute to specify the name of the activity whose execution the event refers to and the attribute *value* for indicating the value of a data object field.

Table 4 reports the Core Trace Ontology classes, object and data properties, as well as axioms. An example of Core Trace Ontology’s class axiom is given by:

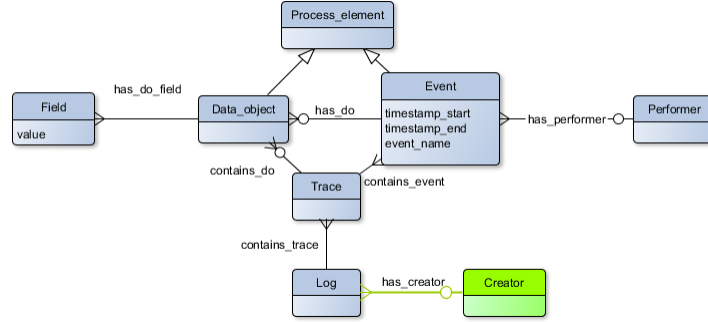
$$\text{Trace} \sqsubseteq (= 1) \text{contains_trace}^- \text{Log} \quad (12)$$

clarifying that a **Trace** is contained in exactly one **Log**. The relationship between **Data_object** and **Field** is instead specified by the axiom:

$$\text{Data_object} \sqsubseteq \exists \text{has_data_object_field} \text{Field} \quad (13)$$

reporting that a **Data_object** has at least one **Field**.

¹⁵In XES, the start and the end timestamps are usually modeled through the XES lifecycle extension, which allows for specifying the transaction type associated to an event. The timestamp of the event with transaction type “start” corresponds to the starting timestamp, while the timestamp associated to the corresponding event with transaction type “complete” to the completion timestamp.



(a) Trace Ontology (Trace Extension Ontology in green).

```

<log xes.version="1.0" xes.creator="Flux">
<string key="concept:name" value="l12">
...
<trace>
<string key="concept:name" value="t173772">
  />
<event> <!-- event e -->
<string key="concept:name"
value="Produce_Record"/>
<string key="lifecycle:transition"
value="complete"/>
<date key="time:timestamp"
value="2016-10-01T16:35"/>
<string key="org:resource"
value="JohnSmith35"/>
<container key="birth_certificate"> <!-- data
object bc34786 -->
<string key="citizenship"
value="Italian"/> <!-- field f3563 -->
<container key="address">
<container key="city">
<string key="city_name"
value="Rome"/> <!-- field f4562 -->
</container>
</container>
</container>
</event>
...
</trace>
...
</log>
  
```

(b) A simple event log in XES.

Trace individuals
<i>l12</i> : log <i>t173772</i> : trace <i>e</i> : event <i>JohnSmith35</i> : performer <i>bc34786</i> : data object <i>f3563</i> : field <i>f4562</i> : field <i>Flux</i> : log creator
Type assertions
log(<i>l12</i>) trace(<i>t173772</i>) event(<i>e</i>), performer(<i>JohnSmith35</i>) data_object(<i>bc34786</i>) field(<i>f3563</i>) field(<i>f4562</i>) log_creator(<i>Flux</i>)
Structural assertions
contains_trace(<i>l12</i> , <i>t173772</i>) contains_event(<i>t173772</i> , <i>e</i>) event_name(<i>e</i> , "Produce_Record") timestamp_start(<i>e</i> , "2016-10-01T16:35") timestamp_end(<i>e</i> , "2016-10-01T16:35") has_performer(<i>e</i> , <i>JohnSmith35</i>) has_do(<i>e</i> , <i>bc34786</i>) do_has_field(<i>bc34786</i> , <i>f3563</i>) do_has_field(<i>bc34786</i> , <i>f4562</i>) value(<i>f3563</i> , "Italian") value(<i>f4562</i> , "Rome")

(c) Trace Instance Ontology A-Box.

Figure 8: The Trace Ontology. Comments in XES (Figure 8b) specify the *trace individual* in the Trace Ontology (Figure 8c) corresponding to a XES object without a unique identifier.

The Trace Extension Ontology specializes the Core Trace Ontology (via *Subclass-Of* relationships) to enrich the modeling of event logs based on the specific kinds of information gathered by IT systems and analyzed in the current scenario. For instance, information about the provenance of data or the log creator may be available and can be represented by extending the Core Trace Ontology with additional properties, as shown by the green **Creator** entity in Figure 8a.

The Trace Instance Ontology, finally, is the crucial A-Box component that encodes the actual traces of the event log in terms of the Trace Ontology T-Box. We provide the basic idea of the encoding with the help of the XES trace segment in Figure 8b, whose encoding is shown in Figure 8c. Similarly to what was illustrated for the BPD Ontology A-Box, the *trace individuals* shown in Figure 8c represent the elements of the log that we aim at encoding in the ontology. Assertions on trace individuals in Figure 8c belong to two groups: *type assertions* and *structural assertions*. Type assertions indicate the type of trace element a trace individual belongs to. For example, the assertion `performer(JohnSmith35)` states that the trace object *JohnSmith35* is a **performer**. Similarly, the assertion `field(f3563)` indicates that the trace object *f3563* is a **Field**. Structural assertions indicate how the objects tracked down in the log are connected to each other, as well as the values that they assume. For instance, the assertion `has_performer(e, JohnSmith35)` in Figure 8c states that the event *e* is performed by individual *JohnSmith35*, while the assertion `value(f3563, "Italian")`, that the value of the field *f3563* is "Italian".

4.2. Integrating **T** with **K** and **P**

Event logs contain the execution of specific BPDs. Thus, together with predicating that *e* is an event that belongs to trace *t173772* in log *l12*, as stated in the Trace Instance Ontology A-Box depicted in Figure 8c, one has to connect it with the corresponding element in the BPD Ontology and, equally important for the task of semantic analysis, to the Domain Ontology activity it instantiates. Assume, for instance, that *e* is an execution of the task *t₂* (labeled "Produce Record") in Figure 6a. To establish the necessary links, we introduce a specific binary role `execution_of(·,·)`¹⁶ so that assertions of type `execution_of(e, t2)` indicate that the log event *e* is an execution of the BPD element *t₂*. This latter element is qualified to be a task in the BPD Ontology A-Box (see Figure 6b) and moreover is a **Produce_Record** task of the Domain Ontology, as stated in assertion (10). Similarly `execution_of(bc34786, d)`, with

¹⁶Detecting how to create the actual `execution_of(·,·)` A-Box statements in a specific application scenario is not the aim of this paper as different strategies can be implemented. For our use case we have used manual mappings between the (implicit) schema of the XES event log and the domain and process ontology, and the lexicographic (edit) distance between the activity names recorded in the domain ontology and the event names recorded in the trace and between the **Simple_Field** classes in the Domain Ontology and the XES container structure of the field, respectively. This step may be further enhanced exploiting approaches and tools for Ontology Based Data Access (OBDA) as the ones described by Calvanese et al. (2017b,c).

d a `Birth_Certificate` according to assertion (11), specifies that the data object *bc34786* in the execution log is an actual birth certificate.

5. Semantic Modeling and Analysis: a Concrete Tool

The management of the integrated model of **T**, **K**, and **P** has been enabled through an extension of MoKi (Di Francescomarino et al., 2014a; Ghidini et al., 2012b), hereafter called MoKiP.

MoKiP aims at personalizing the collaborative environment MoKi (Rospocher et al., 2009) to support the modeling of complex data-aware processes and the analysis of their executions. A complete illustration of the pre-existing functionalities of the MoKi tool, as well as of its collaborative features, is provided by Ghidini et al. (2012b). What is important to mention here is that MoKi is a collaborative tool based on MediaWiki¹⁷ for modeling ontological and process-related knowledge in an integrated manner. How will become clear in the next two sections, the usage of MoKiP enables different types of modeling actors, such as *Domain Experts* and *Knowledge Engineers*, to easily (i) model semantically annotated complex data-aware business processes, and (ii) analyse their execution by means of PPI via semantic-based reasoning techniques.

MoKi uses OWL as the language for modeling ontological knowledge, and BPMN as the reference language for procedural knowledge, associating each semantic term of these languages – i.e., each entity of the ontology and the process model – to a wiki page (see Figure 9, which illustrates a MoKi page describing the ontology concept *Mountain*).

From a high level point of view, the main features of interest of MoKi for the contributions described in this paper are:

- the support for the integrated modeling of different types of conceptual models, covering both ontological and procedural knowledge, with multiple modes of accessing knowledge. In particular, the fully-structured access mode enables users to view/edit the conceptual model using a formal specification interface, i.e., OWL-DL axioms for ontological knowledge and BPMN BPDs (via the integrated Oryx editor¹⁸) for procedural knowledge. The lightly-structured access mode, instead, enables users to work on the conceptual model using semi-formal templates. This multiple access mode allows for fostering the collaboration among modelers with different expertise.
- the support for online collaboration between *Domain Experts* and *Knowledge Engineers* composing the modeling team. *Domain Experts* play the role of driving the core changes in the ontology due to their expertise in the domain being modeled, while *Knowledge Engineers* support the overall modeling process by ensuring the formal correctness of the model. MoKi

¹⁷<http://www.mediawiki.org/>

¹⁸<http://bpt.hpi.uni-potsdam.de/Oryx/>

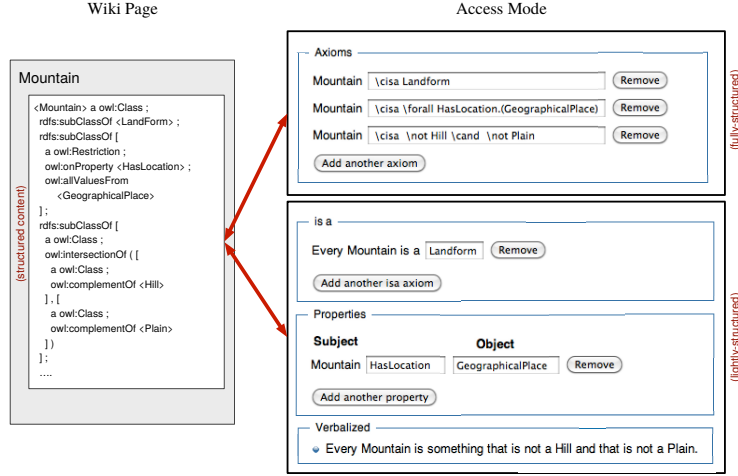


Figure 9: Multi-mode access to the wiki page for a concept in MoKi. The fully-structured access mode uses the *latex2owl* (Matassoni et al., 2014) syntax for OWL 2, a Latex-style syntax for compactly writing ontologies with a text editor, convertible to/from other syntaxes.

supports the simultaneous modeling activity performed by several modelers geographically distributed. Then, by exploiting the basic features of MediaWiki, the tool is able to present the full history of changes occurred on each entity of the ontological and procedural knowledge created.

5.1. MoKiP: Modeling Functionalities

We describe here the modeling component customizations that we developed within MoKiP to facilitate the acquisition of knowledge related to the **K** and **P** dimensions, that is, ‘the business process and ontology modeling required for the business process annotation and specification of the data objects manipulated by the process.’¹⁹

In order to acquire the different types of knowledge characterizing complex scenarios and their interrelations, MoKiP has been equipped with interfaces for: (i) the acquisition of domain-specific knowledge in **P**, i.e., of the process model constituting the BPD Ontology; (ii) the acquisition of domain-specific knowledge in **K**, i.e., of the Domain-Specific Ontology, e.g., characterizing the data objects manipulated by the process; (iii) the explicit definition of the properties associating the entities in **P** and the ones in **K**. Let us briefly explore these functionalities one by one.

Modeling P. Concerning the modeling of the BPD Ontology, MoKiP integrates the Oryx process modeling editor thus allowing modelers to draw BPDs in BPMN. Given the emphasis on complex data (and in particular on structured

¹⁹A detailed tool description can be found in (Bertoli et al., 2013b).

documents) we have extended the Oryx stencil with a green version of the BPMN *data object* graphical symbol. This element denotes a data object whose structure is further specified in the domain ontology. Besides creating a wiki page for the specific BPD modeled in the Oryx editor, MoKiP enables the automatic creation of wiki pages for each of the relevant process entities (i.e., Activity, Data_Object, and Role) occurring in the BPD. These pages will then be exploited for the (semi-automatic) integration with the domain ontology as explained below.

Modeling K. Concerning the modeling of the Domain-Specific Ontology, MoKiP provides two basic instruments: (i) graphical browsing and editing facilities for manipulating the entire ontology (and in particular the is-A structure); and (ii) dedicated interfaces for the modeling of knowledge related to roles, data objects, and data fields of the Domain-Specific Ontology via the lightly-structured access mode. Figure 10 shows the form used for defining a new subclasses of Data_Object in the Domain-Specific Ontology.

Lightly-structured Access Mode: BirthCertificate

The form is titled "BirthCertificate" and is divided into two main sections: "is a" and "has field".

The "is a" section shows "Every BirthCertificate is a Document" with a "Remove" button and an "Add another isa axiom" button.

The "has field" section shows "Every BirthCertificate has the following fields:" followed by a list of fields: Name, Surname, Address, Citizenship, Maternity, and Paternity. Each field has a "Remove Field" button. There is also a "Copy Fields From" button and an "Add another has field axiom" button at the bottom.

Figure 10: MoKiP form for defining Data_Object sub-classes in the Domain-Specific Ontology.

In detail, Figure 10 shows the form used for defining new classes of Document, a subclass of Data_Object in the PA domain. *Domain Experts* can use these interfaces to make explicit, for example, the knowledge about data objects and their relationships with other entities, i.e., with other data objects for defining a hierarchical organization or with data fields for defining the data object internal structure. In this example a domain expert can specify that a BirthCertificate is a Document, whose fields are Name, Surname, Address etc.

The complete underlying formal description of this entity can be accessed by *Knowledge Engineers* through the fully-structured access mode. Working on the same shared model, both *Domain Experts* and *Knowledge Engineers* are called to collaboratively evolve the static model of the domain. This task is made simpler by the specialized collaborative features offered by MoKiP (e.g., discussions and history tracking), which improve awareness and communication among experts.

Modeling the relation between $\mathbf{K}$ and $\mathbf{P}$. The high-level relationships among entities in $\mathbf{K}$ and $\mathbf{P}$ are directly extracted from the BPD. In fact, for each BPMN pool, BPMN data object, and BPMN task (sub-process), MoKiP automatically creates wiki pages and the corresponding ontology concepts that are placed as sub-concepts of the appropriate Core Domain Ontology concept.²⁰ Then, for each BPMN construct in the BPD denoting an activity (role, data object), MoKiP automatically creates an A-Box individual t in the BPD Ontology with the appropriate type assertions (*Instance-Of* relations) in terms of the BPMN Ontology presented in Section 3.2 and in terms of the Domain Ontology as described in Section 3.3.

Besides the relationships that can be extracted from the BPD, MoKiP also enables to specify, by means of easy to use tabular views, for each activity, which **Data_Objects** it manages and how, i.e., which among the process data objects and simple fields the activity creates, updates, or only reads. In addition to enriching the integrated ontological model, this information enables advanced forms of reasoning on execution data such as the ones discussed by Bertoli et al. (2013a), which are beyond the scope of this paper.

5.2. MoKiP: Semantic Data Storage Functionalities

We describe now the architecture for the run time collection of process execution data in the form of XES execution traces and its integration into the comprehensive integrated model of Sections 3 and 4, in order to enable the evaluation of PPIs such as the ones illustrated in Section 2.

The architecture is designed to address two main challenges: (i) *scalability*, to support the large data amounts and rates characterizing collected traces; and (ii) *inference*, to support analysts in querying also for the implicit knowledge that can be derived from the process model and the collected traces. As an example of the latter challenge, PPI.4 mentions generic public offices while the execution traces may store data concerning specific offices. Being able to exploit the classification of public offices contained in the Domain Ontology in order to answer PPI.4 is one of the aims of the architecture that we describe here.

We address the aforementioned challenges by leveraging Semantic Web *triple-stores*, i.e., repositories storing RDF triples (and thus ontologies) within named graphs, supporting different forms of reasoning (e.g., RDFS, OWL, rule-based)

²⁰MoKiP also provides an auto-completion facility to suggest roles, activities, or data objects that exist in the ontology to be used in the BPMN definition of the BPD.

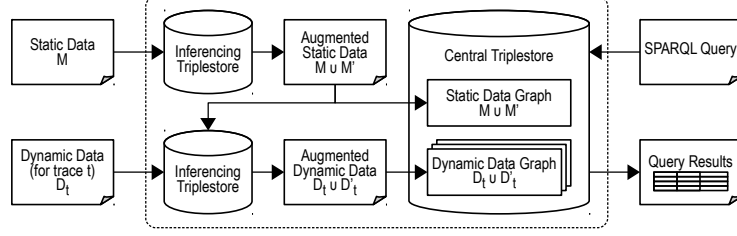


Figure 11: Overview of the architecture proposed for semantic data storage. M corresponds to $\mathbf{K} + \mathbf{P} + \mathbf{T}$ -Box of $\mathbf{T}$ (design time data). D corresponds to the A-Box of $\mathbf{T}$ (run time data) with $D = \bigcup_t D_t$ where D_t is the set of dynamic data triples for trace t . M , D and D_t denote asserted triples, M' , D' and D'_t inferred triples.

and query answering via SPARQL, a language and protocol standardized by W3C (Seaborne and Harris, 2013). Triplestores have become a mature technology, with different products now able to manage huge amounts of RDF data (e.g., GraphDB,²¹ Blazegraph,²² and Virtuoso,²³ to name a few). This, and their support for ontological data, makes triplestores an optimal choice for the implementation of our model.

Figure 11 and the following sections detail the organization and population of data in our triplestore-based architecture, while in Sections 6 and 7.2 we show the effectiveness of this solution in addressing the challenges it was designed for.

Data Organization. We encode all the T-Box and A-Box components of our ontological model as RDF triples, and store them in a single *central triplestore* where all inferable triples are *materialized*. As inference regime we opt for OWL 2 RL, which encompasses most of OWL DL employed in our model and is amenable to efficient rule-based reasoning, where if-then entailment rules are repeatedly applied to RDF data until a fix point is reached. Other regimes may be configured too, such as RDFS that is computationally cheaper although it does not support all the inferences entailed by our model, thus limiting the PPIs that can be answered (e.g., PPI.4 involving OWL property chain axioms).

Compared to a decentralized architecture, the use of a central triplestore favors query evaluation, as data can be easily retrieved from a single system rather than from multiple locations via complex solutions such as federated queries. Inference materialization also improves query performance, as all the data involved in evaluating a query is readily available and indexed in the triplestore (at the price of a slight increase in storage size and time) and no expensive query time backward-chaining reasoning (e.g., via query rewriting) is needed.

When organizing data in the triplestore, we make a distinction between *static data* available at design-time and not related to a particular trace, and *dynamic*

²¹<http://graphdb.ontotext.com/>

²²<http://www.blazegraph.com/>

²³<http://virtuoso.openlinksw.com/>

data generated at run time and describing actual traces. Static data comprises the entirety of **K** and **P** that form the *model layer* (see Section 2 and Figure 2), augmented with the T-Box part of **T**, that is the Core Trace Ontology and the Trace Extension Ontology (see Section 4.1). Dynamic data comprises the XES traces that are automatically encoded (as described in Section 4) in the A-Box of **T**, that is the Trace Instance Ontology. In Figure 11 and hereafter, we denote with M (from “model”) and D the sets of RDF triples forming respectively the static and the dynamic data that are *explicitly asserted* (e.g., modeled in MoKiP or collected from event logs), with $D = \bigcup_t D_t$, where D_t is the set of dynamic data triples for trace t . Similarly, we denote with M' , D' , and D'_t the sets of new RDF triples that can be inferred from $M \cup D$ and pertain respectively to static data, dynamic data, and dynamic data related to a specific trace t .

We store M and its inferences M' in a dedicated named graph in the central triplestore, whereas we store the trace data D and its inferences D' in separate named graphs, each one containing the data D_t and the inferences D'_t of a completed/running execution trace t . This organization permits to easily address and manipulate each individual trace t in isolation, through the use of SPARQL FROM and USING constructs and of SPARQL graph management methods.

Data Population. The static data M refers mainly to the model layer and changes infrequently if compared to the trace data D that evolves rapidly at run time and requires performing a *trace update operation* every time some new data about a running process is collected.²⁴ Such trace update operation is targeted at the named graph of the trace and either creates or modifies its content, possibly in a non-monotonic way (i.e., retracting previously stored data) due to the need for changing acquired trace properties such as the overall time spent or other aggregate parameters that are updated as the trace execution proceeds.

A straightforward approach for data population would be to directly store M and D triples in the central triplestore, leveraging its inference support to compute and materialize inferred triples M' and D' each time data is added or modified. This solution is however inefficient, as it demands a single system to simultaneously deal with storing, inferencing and querying tasks that may overload it, especially when considering the cost of retracting previously inferred triples that no longer hold after a non-monotonic trace update operation.

A more efficient approach exploits the two facts that (i) M does not depend on D , as it encodes general knowledge not specific to a particular trace, and that (ii) traces in D can be assumed to be largely independent one to another, as they describe different and unrelated cases and little or no meaningful knowledge can

²⁴Changes in M result from the modeling activities performed in MoKiP, which typically occur before the trace data of the modeled processes is collected. In the current version of our tool, a change in the model M is handled by invalidating and recomputing inferences for all the traces affected by the change. For more complex scenarios, multiple versions of the model can be maintained in the system, each trace linked to (and its deductive closure materialized using) the version of the model holding when the trace was collected.

be derived from the combination of different traces.²⁵ As a result, inferences M' can be computed starting from M only, while inferences D' can be computed one trace at a time as $D' = \bigcup_t D'_t$, with D'_t being the inferences for the individual trace t computed starting from $D_t \cup M \cup M'$.

Figure 11 shows the processing workflow under the assumption of trace independence. When the system is initialized and after each model changes, the static data M is processed by an inference engine such as a temporary triplestore used in that role (top-left *inferencing triplestore* in Figure 11). This produces the inferences M' that are combined with M to form the *augmented static data*, which is then saved in the central triplestore once for all in a dedicated *static data graph*. When a trace update operation occurs for a trace t , its dynamic data D_t is processed by another inference engine (bottom-right *inferencing triplestore* in Figure 11) together with the triples of the augmented static data $M \cup M'$, obtaining inferences D'_t that are merged with D to form the *augmented dynamic data* for trace t . From this, we discard redundant triples already included in $M \cup M'$ or not needed for PPI evaluation (e.g., $\times$ owl:sameAs $\times$ and $\times$ rdf:type $_$:bnode triples) and we store the result in a trace-specific *dynamic data graph* in the central triplestore (possibly replacing previous contents).

A first advantage of the presented approach is the implementation of reasoning and querying via separate components (inference engine/triplestore and central triplestore) that can be selected and tuned differently based on their respective tasks. We currently use GraphDB for both components, configuring the chosen inference regime and disabling persistence when used as inference engine, and turning off inference and increasing index sizes when deployed as central triplestore. Another advantage, perhaps more important, is the possibility to reduce the amount of data inference performed and to process multiple trace update operations in parallel. Indeed, by exploiting several inference triplestores on one or multiple machines, it is possible to achieve massive scalability. The only possible limiting factor is the use of a single central triplestore for the storage of processed data. This operation is however efficient, since it manipulates whole graphs and does not involve inference materialization or retraction. Moreover, the risk of introducing a bottleneck, as well as a single point of failure, can be mitigated by employing commercial-grade triplestores offering specific features for fault-tolerance and scalability, both vertical (i.e., achieved using better machines) and horizontal (i.e., achieved using clusters of machines).

We are now ready to discover how MoKiP enables the creation of PPIs, their translation into SPARQL queries targeted at the central repository just presented here, and the display of the results.

²⁵We refer to the materialization of new A-Box assertions about an instance (e.g., a document) starting from T-Box statements and other A-Box assertions on the same instance, based on the chosen inference regime. Since traces are typically unrelated and thus share little or no instances, little or no additional inferences can be derived by combining multiple traces. Nevertheless, note that it is possible to derive useful knowledge via comparison or aggregation of trace data in a non-OWL setting, a task achievable using the querying facilities of MoKiP.

6. Supporting the analysis of complex data-aware processes

In the previous section, we described the facilities that we developed for modeling the different layers of knowledge and for storing data. Here, we focus on the analysis task by presenting the features enabling the domain experts to query the repositories for performing an analysis of the process executions.

In order to analyze data concerning the process execution, MoKiP has been equipped with two additional components: (i) a PPI builder, and (ii) a set of functionalities to explore the PPI results. In the following, we explore the two components one by one also illustrating how PPIs are automatically transformed in SPARQL queries that interact with the triplestore described in Section 5.2.

6.1. PPI definition using the PPI builder

The PPIs used for process monitoring and analysis are formulated as SPARQL queries that are evaluated in the central triplestore against the integrated knowledge base and the inferences materialized as previously described. As an example, the SPARQL query and its results for PPI.4 are reported respectively in Listing 1 and Table 5.

MoKiP provides two mechanisms for the definition of PPIs: (i) through the explicit definition of the SPARQL query; (ii) through the instantiation of parametric templates.

In the first case, the PPI builder of MoKiP allows users to create and edit the SPARQL definitions of PPIs. Figure 12 shows the interface used to edit PPIs allowing the user to specify their information:

- PPI ID: the string used to uniquely identify the PPI.
- Reference Process: the process the PPI is associated with. The dropdown list contains both complex processes and single activities, as a PPI can be related to the whole process or to a single activity.
- PPI Type: this parameter can assume two values: *Single* and *Aggregated*, according to whether the PPI is a metric referred to a single process execution or is an aggregated metric on several process executions. When the *Single* option is selected, the query result contains detailed information for each process execution that matches the query. Instead, when the *Aggregated* option is selected, the query returns the aggregated metrics computed over the process executions that match the query.
- Returned value type: this parameter describes the datatype returned by the PPI among the five allowable values – *Numeric*, *Numeric Count*, *Percentage*, *String*, *Enumeration*. If this parameter is set to *Percentage*, the SPARQL query field will be composed of two queries, the numerator and the denominator used for computing the percentage-based result.
- PPI Description: a textual description used to catalog the PPI.
- SPARQL Query: the actual query associated with the modeled PPI.

Table 5: Query results for the parametric PPI.4: “Number of cases and average time spent for optional activities by public offices involved in the birth management process.”

Office_name	Activity_name	Executions	Time
“SAIA”	“VerifyCF”	86	501.097 s
“SAIA”	“GenerateCFSAIA”	15009	270.122 s
“Municipality”	“GenerateCFMunicipality”	21000	485.541 s
“APSS”	“GenerateCFAPSS”	8315	418.327 s

```

PREFIX ofn: <http://www.ontotext.com/sparql/functions/>
PREFIX bpmn: <http://dkm.fbk.eu/index.php/BPMN_Ontology#>
PREFIX domain: <https://dkm.fbk.eu/#>
PREFIX trace: <http://dkm.fbk.eu/Trace_Ontology#>

SELECT ?office_name ?activity_name (COUNT(?t) AS ?executions) (AVG(?t) AS ?time)
WHERE {
  [] a bpmn:business_process_diagram, domain:BirthManagement; # retrieve process
  bpmn:has_business_process_diagram_pools [ # and public office
    bpmn:has_pool_participant_ref [ # participants for
      a domain:PublicOffice; # each pool of the
      bpmn:has_participant_name ?office_name ]; # birth management
    bpmn:has_pool_process_ref ?process ]. # process diagram

  ?process bpmn:has_process_graphical_elements # require the
  ?gateway1, ?gateway2, ?activity . # participant process
  # to contain two
  ?gateway1 a bpmn:gateway; # gateways and an
  bpmn:is_directly_connected_via_sequence_flow ?gateway2; # activity connected
  bpmn:is_connected_via_sequence_flow ?activity. # by two paths:
  # - a direct path
  ?gateway2 a bpmn:gateway. # between gateways
  # - an indirect
  ?activity a bpmn:activity; # path passing
  bpmn:has_flow_object_name ?activity_name; # through the
  bpmn:is_connected_via_sequence_flow ?gateway2. # activity

  OPTIONAL { # if the activity
    ?activity_execution trace:execution_of ?activity ; # was performed,
    trace:initial_start_dateTime ?start ; # get its execution
    trace:final_end_dateTime ?end. # time leveraging the
    BIND (ofn:asMillis (?end - ?start ) AS ?t) # custom function
  } # ofn:asMillis
}
GROUP BY ?office_name ?activity_name

```

Listing 1: SPARQL query defining the parametric PPI.4: “Average time spent and number of executions of optional activities by each public office involved in the process.”

The second mechanism, relying on parametric PPIs, allows MoKiP to foster the involvement of users in the PPI definition. Parametric PPIs are templates that have to be instantiated before being run to retrieve data from the data store. Figure 13 shows an example of form for the instantiation of a parametric PPI template. Note that this solution offers the possibility for expert users (say business analysts) to design generic parametric PPIs that end users can instantiate easily to define specific PPIs carrying out their own desired analysis. The parameters provided in the parametric PPIs, indeed, are automatically

PPIs Builder

PPI ID:

Reference Process: BirthManagement ▼

PPI Type: Single ▼

Returned value type: Numeric ▼

PPI Description:

SPARQL Query:

Figure 12: Form for the definition of a PPI provided by the PPI builder of MoKiP.

reported into the SPARQL query underlying the specific PPI, by replacing the corresponding keywords. MoKiP currently allows the parametrization of data objects, activities and data structures. Depending on the kind of element that has been parameterized, the keywords used within the SPARQL queries are different:

- `_PROCESSID_n_` for identifying the process that is used in the PPI.
- `_ACTIVITYID_n_` for identifying the activity used in the PPI. If the keyword `_PROCESS_n_` is present, the provided list is limited to the subset of activities included in the selected process. Otherwise, the full list of activities included in the model is shown.
- `_DOCUMENTID_n_` for identifying the document used in the PPI. If the keyword `_ACTIVITYID_n_` is present, the provided list is limited to the subset of documents that are directly connected with the selected activity. Otherwise, the full list of documents included in the model is shown.
- `_FIELDID_n_` for identifying the document field used in the PPI. If the keyword `_DOCUMENTID_n_` is present, the provided list is limited to the subset of fields contained in the selected document.
- `_VALUEID_n_` for specifying the field value that has to be monitored by the PPI, in case field keywords are defined in the template.

Listing 2 reports the SPARQL queries associated to an example parametric PPI, showing the use of the keywords defined above.

Besides the usage of templates and parameters, what is important to notice here is the ability of MoKiP to easily define PPIs involving knowledge on both the business process and the data (or role) components.

Configure PPI:

PPI ID	PPI2_
PPI Description	Percentage of process instances which passes through _ACTIVITYID_1_ and _ACTIVITYID_2_ and have the field _FIELDID_1_ of the document DOCUMENTID_1_ filled in _ACTIVITYID_2_ (with respect to all the process instances passing through _ACTIVITYID_1_ and _ACTIVITYID_2_).
PROCESSID 1	-----
ACTIVITYID 1	-----
ACTIVITYID 2	-----
DOCUMENTID 1	-----
FIELDID 1	-----
VALUEID 1	
Save Cancel	

Figure 13: Parametric PPI instantiation form in MoKiP.

Concerning the use of SPARQL, we found that analytical PPIs take advantage from SPARQL 1.1 aggregates, but SPARQL built-in support for managing temporal information (e.g., to compute time differences) results rather inadequate. We addressed this issue by leveraging custom SPARQL functions, either provided by the employed GraphDB triplestore or implemented by us using the extension mechanism offered by GraphDB/RDF4J.²⁶ However, this solution is not portable to other triplestore products (e.g., Virtuoso), which provide their own extension mechanisms.

6.2. PPI evaluation and results visualization

The PPIs defined in the PPI builder can then be instantiated and evaluated, with the corresponding SPARQL query(ies) being executed on the back end semantic repository (described in Section 5.2) that contains all the knowledge of **K**, **P**, **T** in RDF form opportunely organized in named graphs.

The instantiation of a PPI can be performed by selecting from the PPIs list (Figure 14) one of the available template. Parametric templates are identified by the blue background and by the presence of the *Create Instance* link.

Once the PPI to instantiate has been selected, MoKiP shows the mask for setting the parameters to use for instantiating the PPI. An example of this mask is shown in Figure 13 for the parametric PPI of Listing 2. For each parameter defined in the PPI query, an input element is shown.

Once the PPI has been selected or instantiated, it can be executed using the *Execute* link in the PPI Manager list (see the links nearby the PPIs in Figure 15) and the results are shown below the PPI. For instance, Figure 15

²⁶<http://www.rdf4j.org/>

```

PREFIX bpmn: <http://dkm.fbk.eu/index.php/BPMN_Ontology#>
PREFIX domain: <https://dkm.fbk.eu/#>
PREFIX trace: <http://dkm.fbk.eu/Trace_Ontology#>

SELECT (COUNT(DISTINCT ?trace) AS ?numerator)
WHERE
{
  ?trace trace:execution_of / a bpmn:business_process_diagram, domain:_PROCESSID_1_;
  trace:has_process_trace_flow_object ?executed_activity1, ?executed_activity2.

  ?executed_activity1 trace:execution_of / a bpmn:flow_object, domain:_ACTIVITYID_1_.
  ?executed_activity2 trace:execution_of / a bpmn:flow_object, domain:_ACTIVITYID_2_;
  domain:has_output_field ?filled_field.

  ?filled_field a domain:_FIELDID_1_; ^domain:has_field / a domain:_DOCUMENTID_1_.
}

SELECT (COUNT(DISTINCT ?trace) AS ?denominator)
WHERE {
  ?trace trace:execution_of / a bpmn:business_process_diagram, domain:_PROCESSID_1_;
  trace:has_process_trace_flow_object ?executed_activity1, ?executed_activity2.

  ?executed_activity1 trace:execution_of / a bpmn:flow_object, domain:_ACTIVITYID_1_.
  ?executed_activity2 trace:execution_of / a bpmn:flow_object, domain:_ACTIVITYID_2_.
}

```

Listing 2: SPARQL formulation of an example parametric PPI showcasing 4 keywords (parametric PPIs defined by knowledge engineers are typically simpler): “Percentage of process instances which passes through `_ACTIVITYID_1_` and `_ACTIVITYID_2_` and have `_FIELDID_1_` of document `DOCUMENTID_1_` filled in `_ACTIVITYID_2_`, with respect to all the process instances passing through `_ACTIVITYID_1_` and `_ACTIVITYID_2_`.”

shows the result of the execution of the aggregated PPI, PPI1, which returns the number of process instances.

7. On the usage of MoKiP

The evaluation of the viability of a complex tool such as MoKiP is a similarly complex activity. Several aspects and functionalities can be evaluated both in terms of technical robustness and front-end usability. Different functionalities of MoKi, as a modeling tool, have already been evaluated. Di Francescomarino et al. (2014a) present a comprehensive account of the advantages of MoKi in fostering the collaboration between different actors when building conceptual models. The work by Di Francescomarino et al. (2014b) is, possibly, even more relevant to what described here as it shows the (positive) effects of MoKi as a tool that enables building semantically annotated process models. The results show that modeling an ontology alongside the business process favors designing models more accurately and carefully, producing models that are better from a qualitative point of view than the models realized without leveraging semantic annotations.

Given these previous works, here we focus on investigating the effectiveness of the completely new components of the tool, namely the PPI component and

PPI ID	Reference Process	PPI Type	Returned value type	PPI Description	Modify	Delete	Create Instance
PPI1	BirthManagement	Aggregated	Numeric	Process number	Modify	Delete	
PPI2	BirthManagement	Aggregated	Numeric Count	List all processes	Modify	Delete	
PPI3	BirthManagement	Aggregated	Numeric	Number of processes compliant with the process model	Modify	Delete	
PPI4	BirthManagement	Aggregated	Numeric Count	List of processes compliant with the process model	Modify	Delete	
PPI51	BirthManagement	Single	String	Is the process compliant with the model?	Modify	Delete	
PPI52	BirthManagement	Single	String	Reason of no-compliance of a process with respect to the process model	Modify	Delete	
PPI53	BirthManagement	Single	String	Possible repair actions on process execution	Modify	Delete	
PPI54	BirthManagement	Single	Enumeration	Has the process finished its execution?	Modify	Delete	
PPI5	BirthManagement	Aggregated	Numeric	Number of processes not compliant with the process model	Modify	Delete	
PPI6	BirthManagement	Aggregated	Numeric Count	List of processes not compliant with the process model	Modify	Delete	
PPI13_INST1	BirthManagement	Aggregated	Percentage	Percentage of processes finishing their execution (with respect to all processes)	Modify	Delete	
PPI7_SCH	BirthManagement	Aggregated	Numeric Count	List of processes with provenance _VALUEID_1_	Modify	Delete	Create Instance
PPI8_SCH	BirthManagement	Aggregated	Numeric Count	List of compliant processes with provenance _VALUEID_1_	Modify	Delete	Create Instance
PPI9_SCH	BirthManagement	Aggregated	Numeric Count	List of not compliant processes with provenance _VALUEID_1_	Modify	Delete	Create Instance
PPI10_SCH	BirthManagement	Aggregated	Numeric Count	List of processes with provenance _VALUEID_1_ started later than _VALUEID_2_ and finished before _VALUEID_3_	Modify	Delete	Create Instance
PPI11_SCH	BirthManagement	Aggregated	Numeric Count	List of compliant processes with provenance _VALUEID_1_ started later than _VALUEID_2_ and finished before _VALUEID_3_	Modify	Delete	Create Instance
PPI12_SCH	BirthManagement	Aggregated	Numeric Count	List of not compliant processes with provenance _VALUEID_1_ started later than _VALUEID_2_ and finished before _VALUEID_3_	Modify	Delete	Create Instance
PPI13_SCH	BirthManagement	Aggregated	Percentage	Percentage of processes passing through activity _ACTIVITYID_1_ (with respect to all processes)	Modify	Delete	Create Instance
PPI14_SCH	BirthManagement	Aggregated	Numeric Count	List of processes with provenance _VALUEID_1_ started later than _VALUEID_2_ and before _VALUEID_3_	Modify	Delete	Create Instance
PPI_ANOMALIE	BirthManagement	Aggregated	Numeric		Modify	Delete	

Figure 14: PPI list supporting the selection and instantiation of the PPI to evaluate.

the triplestore repository. For the first we provide a proof-of-concept using opinions about the collaborative process modeling and monitoring capabilities of the tool collected from users during the use of MoKiP in a project in the Public Administration domain (Section 7.1). For the second we perform a quantitative evaluation of our triplestore-based solution in terms of storage, reasoning and PPI evaluation performance (Section 7.2).

7.1. Modeling and building PPIs in MoKiP

The evaluation of MoKiP aims at measuring its effectiveness in supporting different experts in the collaborative modeling and analysis of complex organizations' scenarios. Such evaluation has been performed on the real Birth Management use case, in the context of the *ProMo* local project in the Public Administration domain (Section 2).

Purpose of the evaluation was answering the following research question related to the collaborative process modeling and monitoring:

RQ_{MM1} *Is MoKiP front-end effective in supporting the collaborative modeling and monitoring of processes in complex scenarios?*

To this aim, a round of interviews with real experts involved in the project was performed to identify positive and negative aspects of the implemented tool. Different typologies of experts were engaged, each accomplishing a different task: 3 *Domain Experts*, responsible of describing the domain (both static and dynamic aspects); 2 *Knowledge Engineers*, in charge of formalizing the (static) knowledge described by the *Domain Experts* using formal languages; 2 *Business Designers* for the formalization of the procedural knowledge; and 2 *Business*

PPIs Manager

Select process to analyze: BirthManagement

PPI ID	Reference Process	PPI Description	
PPI1	BirthManagement	Process number	Execute
Result: 226			
PPI2	BirthManagement	List all processes	Execute
PPI3	BirthManagement	Number of processes compliant with the process model	Execute
PPI4	BirthManagement	List of processes compliant with the process model	Execute
PPI5	BirthManagement	Number of processes not compliant with the process model	Execute
PPI6	BirthManagement	List of processes not compliant with the process model	Execute
PPI13_INST1	BirthManagement	Percentage of processes finishing their execution (with respect to all processes)	Execute
PPI7_SCH	BirthManagement	List of processes with provenance _VALUEID_1_	Configure and Execute
PPI8_SCH	BirthManagement	List of compliant processes with provenance _VALUEID_1_	Configure and Execute
PPI9_SCH	BirthManagement	List of not compliant processes with provenance _VALUEID_1_	Configure and Execute
PPI10_SCH	BirthManagement	List of processes with provenance _VALUEID_1_ started later than _VALUEID_2_ and finished before _VALUEID_3_	Configure and Execute
PPI11_SCH	BirthManagement	List of compliant processes with provenance _VALUEID_1_ started later than _VALUEID_2_ and finished before _VALUEID_3_	Configure and Execute
PPI12_SCH	BirthManagement	List of not compliant processes with provenance _VALUEID_1_ started later than _VALUEID_2_ and finished before _VALUEID_3_	Configure and Execute
PPI13_SCH	BirthManagement	Percentage of processes passing through activity _ACTIVITYID_1_ (with respect to all processes)	Configure and Execute
PPI14_SCH	BirthManagement	List of processes with provenance _VALUEID_1_ started later than _VALUEID_2_ and before _VALUEID_3_	Configure and Execute
PPI_ANOMALIE	BirthManagement		Execute

Figure 15: PPI outcome

Sex	Female	22%
	Male	78%
Age	36 - 45	56%
	46 - 55	44%
Educational Qualification	Master	33%
	PhD	67%

Table 6: Experts' demographic information

Analysts, responsible of the definition of the PPIs used for querying the knowledge base to retrieve data for the analysis. Table 6 reports the demographic information related to the experts engaged in the evaluation.

In detail, experts were provided with a half-day training session on the tool. Each of them was then involved on a specific task in the project according to his/her own expertise (e.g., defining the Birth Management process, designing the documents produced in such a process, defining and analyzing the PPIs). At the end of the project, interviews have been conducted with each of them individually in order to collect opinions and feedback about MoKiP.

Specifically, we investigated the front-end tool usability and usefulness. Concerning usability, according to the ISO/IEC 9126-1 specification (ISO/IEC, 2001), we focused on *understandability*, *learnability* and *ease of use* (operability) of the tool. Concerning usefulness, we looked at the impact of the functionalities offered by the tool on the efficiency and the quality of the experts' work. The main questions asked to the experts during the individual interviews are reported in Appendix A. Once completed the individual interviews we manually inspected the answers provided by the experts.

Usability. Interviewed subjects judged the MoKiP interface easy to understand and offering an immediate way for retrieving and accessing knowledge (*understandability*). They mainly appreciated the dedicated interfaces for the different experts which make it easy for each of them to interact with the instrument as well as to collaborate with other experts. Only a minor concern related to the navigability from the dynamic visualization of the documents (in the process model) to their structural description was raised.

Almost all experts declared that it only took half a day (i.e., the duration of the tutorial given at the beginning of the project) to get familiar with the tool. The MoKiP front-end has been overall evaluated as very simple to learn, thus confirming the easy *learnability* of the tool.

Last but not least, experts provided very positive evaluations about the *ease of use* of MoKiP, though they found a bit annoying the time required by the tool to load the process editor and its graphic components.

Summing up, the tool has been judged as overall usable, despite few improvements have been identified and will be implemented in the future versions.

Usefulness. The involved experts unanimously approved the MoKiP support for the collaboration. All users perceived the collaboration as very effective: specifically they mainly appreciated the possibility to discuss and comment on the modeling issues, as well as the shared workspace, which allowed them to edit different aspects of the model in parallel, thus saving time. Overall, MoKiP has been judged as a good comprehensive tool, that through all the integrated facilities dedicated to the definition of different aspects (i.e., processes, data objects, fields, roles, and the interactions among these entities), enables the creation of a complete and integrated model of a domain.

The main limit identified by the users (especially by *Domain Experts* and *Business Designers*) was the lack, in the process model editor, of a facility for comparing different versions of the processes in a graphical way. Such a functionality has been perceived as very important not only for maintaining the big picture of the evolution of the modeling work, but also for having an effective starting point for creating discussions about process changes and for analyzing the possible impact of such changes on other processes of the domain. A second improvement suggested by the users, in particular by *Business Analysts*, is related to the PPI editing functionality. On the one side they would like to have more powerful instruments for defining PPIs, e.g., PPI templates accepting other types of parameters, thus allowing them more flexibility in the PPIs definition; on the other side, they would appreciate having a prompt information about data objects and data structures when defining the PPIs. The received feedback would allow us to improve MoKiP, e.g., by adding the new feature for the graphical comparison of process model versions requested by the experts.

Overall, taking into account the observations related to the usability and the usefulness of the MoKiP front-end, we are able to assess that it is effective in supporting the collaborative modeling and monitoring of processes in complex scenarios (**RQ_{MM1}**). The analysis carried out also gave us the opportunity to collect a number of suggestions to improve the usability and usefulness of the

instrument.

7.2. Populating and querying the triplestore

To assess the effectiveness of the triplestore-based storage solution of MoKiP described in Section 5.2, we evaluated its performance in terms of *data population* and *data querying* (i.e., PPI evaluation) on the same Birth Management use case employed in the user evaluation of Section 7.1 and throughout the paper as motivating example, scaled to a hypothetical nationwide deployment.

Purpose of the evaluation was answering the following research questions related to the triplestore-based solution adopted for storing data:

RQ_{TS1} *Is MoKiP storage solution able to cope with non-toy data volumes and rates typical of real-world applications?*

RQ_{TS2} *Does MoKiP storage solution outperform the baseline solution where a single triplestore is used for both data storage and inference?*

Aim of **RQ_{TS1}** is investigating the real-world suitability of the MoKiP storage solution both in case of *data population* and *data querying* tasks, as they represent the two core tasks demanded to the storage solution. To answer the question, we simulated these two tasks with typical real-world data volumes and investigated the performance (in terms of required time, throughput and capability to complete the task) of the MoKiP solution to deal with these scenarios.

Aim of **RQ_{TS2}** is evaluating whether the MoKiP storing solution (**MoKiP system**), in which a central triplestore is supported by triplestores devoted to materialize triples, has actually better performance for the data population task than the standard solution (**baseline system**), in which a single triplestore is employed. To answer the question, we carried out the *data population* task and we compared the performance (in terms of required time, throughput and capability to complete the *data population* task) obtained by using the **MoKiP system** and the **baseline system**. We did not compare the two systems for the *data querying* task, since for both of them the data to be queried are stored in the same central triplestore, leading to the same query performance.

A first investigation partially addressing **RQ_{TS1}** was reported by Bertoli et al. (2014). Here, we extended it to cover both research questions, including more data, PPIs, and analyses, as well as harvesting all the system improvements occurred since that preliminary investigation – both in the employed triplestore and in the implementation of MoKiP driving the triplestore.²⁷

²⁷We employed GraphDB SE 8.10.1 leveraging its optimized OWL 2 RL and RDFS rulesets and its support for the separation between immutable data (called “schema” in GraphDB and used for the model triples $M \cup M'$ of Section 5.2) and mutable data (the traces $D \cup D'$). Both GraphDB and MoKiP were run on a Linux 3.10 server with 2 Xeon E5-2430 CPUs (12 cores overall, hyper-threading not used in order to obtain consistent test results), 192 GB RAM and 400 GB SSD. Due to the different software and hardware setup, the results reported here are not directly comparable to the ones by Bertoli et al. (2014).

In the following, we report, besides settings and data used in the evaluation, the results related to the *data population* and *querying* tasks used to answer to the research questions. All the material for this evaluation, including detailed results, as well as the code and data required to reproduce the experiments, is available online.³

Evaluation procedure. We evaluated the performance of the MoKiP storing solution both in the *data population* and in the *data querying* tasks.

Concerning the *data population* task, we evaluated the performance (in terms of required time, throughput and capability to complete the task) of the MoKiP system and the **baseline system (RQ_{TS2})** to populate the storing solution with (i) different volumes of data up to typical real-life scenarios (**RQ_{TS1}**); (ii) different numbers of employed CPU cores to assess scalability; and (iii) different inference regimes. In particular, we opted to investigate the impact on the performance of two different inference regimes:

- OWL 2 RL, which allows for fully leveraging the expressiveness of the model and performing even complex PPIs (as for instance, PPI.4) over it.
- RDFS, which with lower inference capabilities allows for dealing only with part of the expressiveness of the model, but is computationally less expensive and thus may still be of interest in certain application scenarios.

Concerning the *data querying* task, we focused on OWL 2 RL inference as, in order to evaluate the capability of the storage solution to deal with real-life scenarios, we want to be able to compute even complex PPIs. We evaluated the performance (in terms of time required for returning results and capability to return results) of MoKiP to query the triplestore with: (i) different volumes of data up to typical real-life scenarios; and (ii) different and even complex PPIs typical of real-life scenarios (**RQ_{TS1}**).²⁸

Evaluation data. We considered the integrated **K+P+T** ontology for the Italian birth management use case introduced in Section 2 and developed within the industrial case study involving the Public Administration. In this context:

- **K** is a domain ontology with 379 classes and 5 properties detailing 28 activities and 12 data objects, with 25 data fields per data object on average organized in a 4 levels field structure (see Figure 4b).
- **P** is a BPD with 4 pools, 19 activities and events, 14 gateways, 54 sequence flows, 6 message flows, and 11 domain objects (see Figure 1). Combined with **K**, it forms a model layer of over 7000 axioms corresponding to 31102 or 16600 triples after, respectively, OWL 2 RL or RDFS inference materialization (~ 12185 triples before inference).

²⁸Here we consider queries in isolation and therefore we focus on query evaluation times rather than throughput. Also, we do not consider the number of CPU cores as an independent variable as it does not affect query response times.

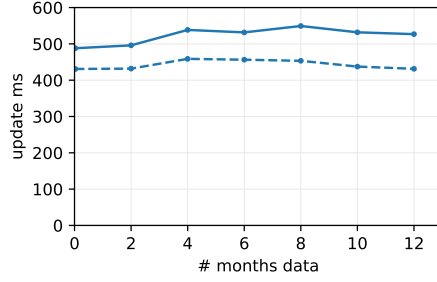
- **T** is a set of execution traces synthesized according to **P**, **K** and few examples of real traces that cannot be used directly for privacy reasons. On average, and due to the simulation of different paths that may omit certain events, each complete synthesized trace spans 10 activities with associated data objects and fields, encoded in 3072 or 2389 A-Box triples after, respectively, OWL 2 RL or RDFS inference materialization (1708 triples before inference).

We considered a deployment of the system at the Italian national level, characterized by 500 000 newborns per year that map to an equal amount of traces in **T**, i.e., roughly 1 new trace starting every minute. Since a trace update operation occurs (at least) every time an activity of the trace is recorded, i.e., around 10 times per trace, we obtain a rate of 10 trace update operations per minute that need to be readily processed by the system. We also assume that 1 year worth of data, i.e., 500 000 traces corresponding to 1.54B OWL 2 RL triples or 1.19B RDFS triples in our synthesized data, has to be maintained by the system at any time, in order to accommodate the necessary analysis and reporting needs that are typically regulated by yearly deadlines. Therefore, the system must be able to store and support PPI evaluation on 1 year worth of data, at the same time providing a data population throughput of (at least) 10 update operations per minute to be usable in the considered use case (**RQ_{TS1}**).

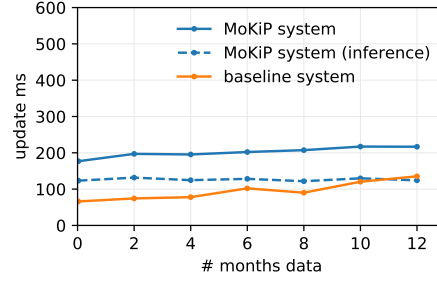
Data population. We pre-populated both the **MoKiP system** and the **baseline system** with synthetic traces considering both OWL 2 RL and RDFS inference regimes and different data sizes up to 1 year worth of traces. Then, for each system, inference regime, data size, and number of employed CPU cores,²⁹ we simulated the population of 300 new test traces, each one starting from an empty trace and being repeatedly updated after each one of its activities is completed, for a total of 3174 trace update operations. The simulation was done both sequentially, considering one trace update operation at a time to measure execution times, and concurrently to measure the maximum achievable throughput in terms of trace update operations per minute.

Under the OWL 2 RL inference regime we could not manage to complete the simulation on the **baseline system**, independently of other test parameters (we obtained throughputs lower than 1 trace per minute and the simulations timed out). At the root of the problem, there is the need for the **baseline system** to retract inferences each time a trace is non-monotonically updated. Indeed, we could complete the test on the **baseline system** if we avoid inference retraction by performing only one update operation per trace, i.e., if we insert a trace in the system only when completed. However, this alternative setting is unrealistic in practice as it would severely limit the analysis capabilities (a trace spans days to weeks and some PPIs concern ongoing, incomplete traces), so in this section we disregard it and include the **baseline system** only for RDFS where the problem is not observed. For the sake of completeness, an analysis of

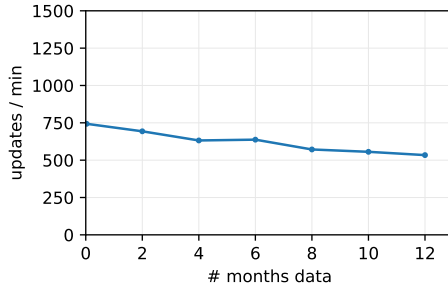
²⁹We “pin” the processes of the system under test to a subset of CPU cores using **taskset**.



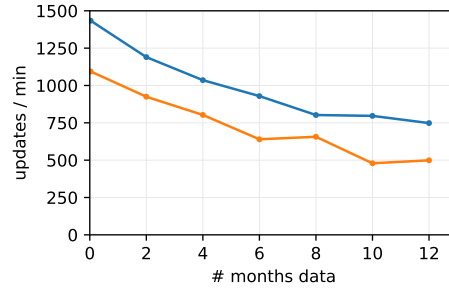
(a) Time by data size – OWL 2 RL, serial execution.



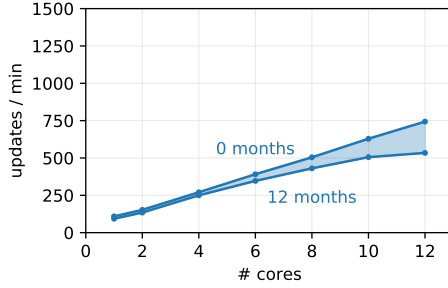
(b) Time by data size – RDFS, serial execution.



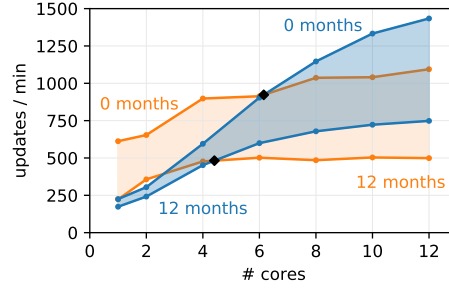
(c) Throughput by data size – OWL 2 RL, 12 cores.



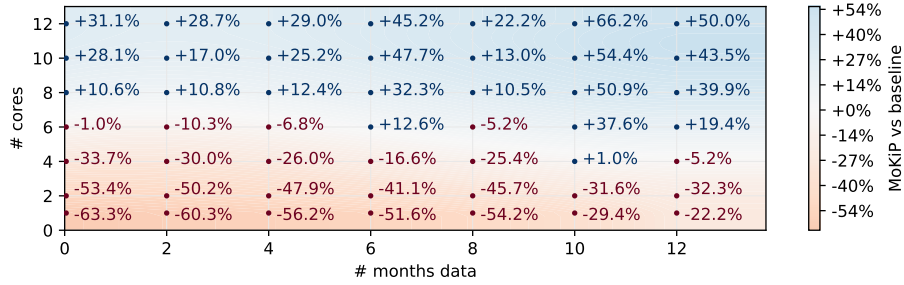
(d) Throughput by data size – RDFS, 12 cores.



(e) Throughput by CPU cores – OWL 2 RL.



(f) Throughput by CPU cores – RDFS.



(g) Improvement on throughput of MoKiP system against baseline system – RDFS.

Figure 16: Population time and throughput of MoKiP with OWL 2 RL and RDFS, different data sizes and CPU cores, compared with a baseline system using a single triplestore on RDFS.

this alternative setting is reported in Appendix B and gives answers to **RQ_{TS1}** and **RQ_{TS2}** similar to those reported here with retractable inferences.

Figures 16a and 16b show the average execution time of a trace update operation at different data sizes. Execution times are substantially lower on RDFS, where the architecturally simpler **baseline system** has a clear advantage on the **MoKiP system**. There is a slow increase of execution time with data size, which is more marked for the **baseline system** than for the **MoKiP system**, while on the **MoKiP system** it stems from increasing storage costs in the central triplestore since inference times (dashed lines) remain approximately constant as expected (443 ms for OWL 2 RL, 126 ms for RDFS, on average).

Figures 16c and 16d analyze the sustainable throughput at different data sizes when using all the CPU resources of the test machine (12 cores). Again, performance is better on RDFS and decreases with data size, significantly more markedly for RDFS where inference is cheaper and size-sensitive storage costs have a larger relative impact. The **MoKiP system** consistently outperforms the **baseline system** on RDFS, and its throughput on OWL 2 RL for 1 year of data (534 updates/min) is even better than the one of the **baseline system** on the simpler RDFS (499 updates/min). Overall, the two figures show that both the **MoKiP system** and the **baseline system** (limited to RDFS) easily meet the storage and throughput requirements of the considered nationwide Birth Management scenario, thus allowing us to positively answer **RQ_{TS1}** for what concerns data population.

Figures 16e and 16f investigate the (vertical) scalability of the systems by varying the number of available CPU cores. For each system and inference regime, throughput is a function of both the number of cores and the data size and results comprised within two boundary lines corresponding to the extreme settings where no data is stored in the system (“0 months” line) and 1 year worth of data is stored (“12 months” line). This region (colored area in the figures) turns out to be wider for RDFS than OWL 2 RL, the throughput for the latter being more insensitive to data size. The conclusion is that for the considered data sizes and CPU cores, the central triplestore featured by both systems represents a bottleneck mainly for RDFS, and more markedly for the **baseline system** where after certain thresholds adding more cores has little effect (0 months line) or no effect at all (12 months line). Interestingly, Figure 16f also shows that the **baseline system** is more efficient than the **MoKiP system** when few CPU cores are used, and specifically up to a threshold number of cores (black diamonds in the figure) that varies with the data size (from ~ 4 to ~ 6 cores moving from 12 to 0 months of data).

Figure 16g further investigates the difference in performance between the two systems on RDFS where they are comparable, by reporting the percentage throughput improvement (or worsening) of the **MoKiP system** against the **baseline system** as a function of both data size and number of cores. The figure further confirms that the **baseline system** is competitive only when few CPU cores and/or small data sizes are considered, which are conditions rarely encountered in practice. This result, together with the infeasibility of using the **baseline system** on OWL 2 RL, allow us to positively answer **RQ_{TS2}**.

Table 7: PPI analysis along dimensions **P**, **K** or **T** and requirement for inference. (**Inf**)

PPI	Description	P	K	T	Inf
PPI.1	Avg. time per process execution spent by a specific municipality M			✓	
PPI.2	Total number of Registration Request documents filled from January, 1st, 2018		✓	✓	
PPI.3	Percentage of times in which the flow followed is the one which passes first through the APSS pool and then through the municipality one	✓		✓	
PPI.4	Number of cases and avg. time spent by each public office involved in the birth management process for executing optional activities	✓	✓	✓	✓
PPI.5	Number of times in which the municipality sends to SAIA a request without FC	✓	✓	✓	✓
PPI.6	Last event of trace T			✓	
PPI.7	Cycle time of trace T			✓	
PPI.8	Does the trace T pass through the activity labeled with “Present at the hospital”?	✓		✓	

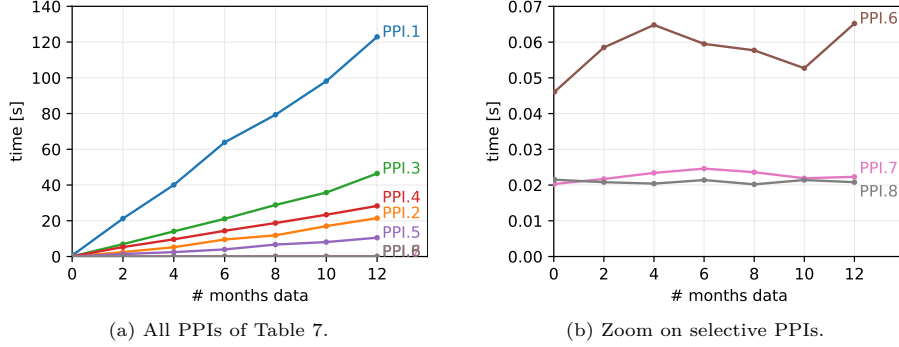


Figure 17: PPI evaluation times of MoKiP with different data sizes.

Data querying. To assess the feasibility of querying stored data in real application scenarios, we selected and analyzed 8 PPIs among the ones defined by the business analysts involved in the investigation of the case study. The 8 PPIs are reported in Table 7 (text only) and Appendix C (SPARQL definitions) and were selected based on their importance for business analysts and their representativeness of the variety of PPIs supported by the system (e.g., in terms of different PPI results and single vs. multiple trace analysis).

A qualitative analysis of the selected PPIs is reported in Table 7. Each PPI is characterized in terms of the dimensions involved by its query (columns **P**, **K**, **T**), and in terms of whether inference is required to properly evaluate the PPI query (column **Inf**). It can be observed in the table that the three dimensions and the inference considered in our approach are all needed to support the considered PPIs, this way motivating the importance of integrating them in a single solution as well as the usefulness of our approach in real business scenarios.

A quantitative evaluation of PPI evaluation times as a function of data size,

using MoKiP with the OWL 2 RL inference regime, is reported in Figures 7a (all PPIs) and 7b (zoom on a subset of PPIs). Evaluation time increases almost linearly with data size for *non-selective* PPIs that involve aggregation over traces (PPI.1 – PPI.5), whereas it remains small and largely independent of data size for *selective* PPIs defined in terms of single traces (PPI.6 – PPI.8). Overall, the figures show that the evaluation of all the considered PPIs remains feasible in MoKiP even when storing 1 year worth of data. In particular, this is true for complex queries like PPI.4, which exploits all the dimensions and the materialized inferences stored in the central triplestore. Execution times are in the order of seconds to few minutes, which although slow are still acceptable for online data analysis. These results permit us to positively answer **RQ_{TS1}** also for what concerns *data querying*.

8. Related Work

Approaches adding annotations to extend the semantics of business process models (also specified in BPMN) have been long investigated in literature. We can organize existing proposals in two broad groups: the first group concerns approaches adding semantics to formalize the dynamic behavior of business processes (Weber et al., 2008; Wong and Gibbons, 2009; Koschmider and Oberweis, 2005; Guzzo et al., 2021); the second group instead collects approaches adding semantics to formalize the meaning of BPD elements and this way further automate business process management (Dimitrov et al., 2007; Thomas and Fellmann, 2007; Di Francescomarino et al., 2008). The latter approaches are more relevant in the context of this paper, since their focus is on models rather than process execution semantics.

Thomas and Fellmann (2007) investigate the semantic annotation of EPC process models. Their framework integrates ontologies and process models via annotation properties (e.g., the “semantic type” of an element) that enrich process instances. Within the SUPER project (Dimitrov et al., 2007), an ontology (the SUPER ontology) enables the enrichment of both EPC and BPMN process models with semantic annotations supporting automated process mediation, composition, and execution. Weber et al. (2008) introduce semantic annotations of process models to support the validation of constraints about their execution semantics.

Di Francescomarino et al. (2008, 2009) exploit the BPMN ontology described by Rospocher et al. (2014) to inject semantic knowledge into BPMN process models, establishing subsumption relations among the classes of two ontologies: one providing a formalization of the BPMN notation and another formalizing the specific domain of the business process. Starting from these relations, they define an ontology integration scheme (by means of hierarchical ontology merge) that enables the automated verification of constraints defining the correctness of semantic annotations (Di Francescomarino et al., 2008), structural constraints (Di Francescomarino et al., 2009) and exceptional flows (Ghidini et al., 2012a) in annotated process models.

Further efforts towards the semantic annotation of business process models are devoted to the creation of tools that support this activity. Examples are: SeMFIS, MoKi, and Cognitive Process Designer. SeMFIS (Fill, 2017) is a tool that supports the semantic annotation of a wide range of conceptual modeling languages, among which BPMN, with concepts taken from ontologies expressed in Semantic Web based languages. Annotations can then be used, e.g., to complement the BPMN models with additional knowledge (a sample mentioned in the paper above concerns risk knowledge) so that models can be analyzed taking into account also this extra knowledge. Both MoKi (Ghidini et al., 2012b) and Cognitive Process Designer (Weller and Maleshkova, 2016) are collaborative MediaWiki-based tools that support the annotation of BPMN BPDs using an intuitive graphical user interface.

Regardless of the different focus and specific goals, all these efforts aim at integrating the procedural and the ontological knowledge, i.e., they mainly pertain to the modeling side only, while forgetting about process executions. On the contrary, several efforts are devoted to the development of query languages for analyzing business process executions focusing only on execution data (see, e.g., Beheshti et al. 2011, 2016).

Few works within the SUPER project (e.g., Hepp and Roman 2007; Pedrinaci et al. 2008b,a; Missikoff et al. 2011) have attempted to integrate procedural and static business level aspects (similar to the ones we consider in this paper) with execution data. However, either they focus on proposing comprehensive frameworks for Semantic Business Process Management (e.g., Hepp and Roman 2007; Marin-Castro and Tello-Leal 2021) or, when dealing explicitly with process analysis and monitoring (Pedrinaci et al., 2008b), they focus on proposing domain-independent process monitoring solutions based on semantic technologies, which can be instantiated in specific organizations' domains to support reconciling the data and business levels. Differently, semantic technologies are exploited in this work to combine the three investigated dimensions and support process analysis through querying and reasoning.

Similarly to us, Missikoff et al. (2011) provide a logic based approach for querying (annotated) business process repositories that also contain execution traces. Their approach is based on logic programming and business processes are described using BPAL, a declarative language for the representation of the procedural workflow knowledge. Domain knowledge is contained in a Business Reference Ontology (BRO) and semantic annotations define correspondences between the elements of BRO and the ones of a BPD. Differently from us, their annotations only concern pure workflow elements (e.g., activities) and are not used to represent, e.g., complex data. Also, their prototype framework is mainly devoted to the implementation of the BPAL back end reasoning engine, and no support is provided at the front end stage.

The problem of supporting large amounts of data with semantic-based approaches without losing their advantages has also been a subject of investigation. To that respect, Ontology Based Data Access (OBDA) is a well known and employed solution (Xiao et al., 2018; Lee et al., 2021; Oliva et al., 2019). OBDA mediates the access to heterogeneous (relational) data sources through a seman-

tic layer consisting in an ontology, this way providing a high-level conceptual view of data (Calvanese et al., 2009) that greatly facilitates its querying and analysis. Efficient OBDA systems, both commercial and open-source, are available, such as Ontop (Calvanese et al., 2017a), Ultrawrap (Sequeda et al., 2014) and Mastro (Calvanese et al., 2011). These systems allow executing SPARQL queries over the ontological abstraction by translating them to SQL queries on the underlying relational source(s), in a way that guarantees correct and complete query results with respect to the ontology and its entailments under well-defined fragments of OWL (typically, the OWL 2 QL fragment amenable to query rewriting). Recent extensions (e.g., MQ-MASTRO by Lenzerini et al. 2018) to the OBDA paradigm to support *metaquerying*, i.e., queries involving “classes of classes” (metaclasses) under well-defined semantics, may be also useful in our setting to support an alternative modeling where instance-of relationships are used in place of `execution_of` to link **T** and **P** instances, the latter playing at the same time the role of individuals (with respect to classes in **K** and the BPMN Ontology) and classes (with respect to **T** individuals).

While OBDA approaches build on the relational data model (mediated by an ontology), triplestore solutions like GraphDB, Virtuoso, and Blazegraph directly implement the RDF model and provide similar query answering facilities through a different mechanism, which employs Semantic Web technologies such as the SPARQL query language (Seaborne and Harris, 2013) and RDFS/OWL reasoning directly implemented on top of RDF. At the same time, their reliance on RDF does not force the use of a rigid schema and provides flexibility in changing or extending the model without losing stored data. These reasons and the capability of triplestores to address both our semantic and scale requirements justify their choice in our approach.

Finally, our modeling of activities, data objects, and their input/output relation both at the process model and execution levels can be related to efforts for defining general frameworks for modeling provenance information. For example, the standard PROV-O ontology (Sahoo et al., 2013) provides a set of classes and properties for representing and exchanging provenance metadata, which can be alternatively used in our context to denote data objects, activity executions, and their “was generated by” provenance relation from an high-level point of view. Therefore, an opportune mapping between corresponding Trace Ontology and PROV-O concepts (e.g., between `Traceable_Flow_Object` and PROV-O `Activity`) may be defined to align the two ontologies and permit PROV-O aware applications to consume Trace Ontology data.

9. Conclusions

This paper extends the state-of-the-art work on semantic process modeling and analysis by: (i) providing a semantic representation not only for business process activities but also for complex data artifacts manipulated by process activities; (ii) extending the OWL knowledge base also with execution traces; and (iii) providing support for the modeling and analysis of complex data-aware business processes. The approach has been assessed on a real case study and

the obtained results, besides suggesting the usability and usefulness of the tool support, also show the scalability of the approach in terms of storage, reasoning and PPI computation performance.

In the future, we plan to extend both the qualitative and the quantitative evaluation of the proposed approach, by also extending it to other real case studies. Moreover, we aim at further improving the tool support by taking advantage of the suggestions gathered through the interviews and to further optimize the ontology-based architecture based on the triplestore inferencing and querying approach.

Acknowledgments

This work was partially supported by “ProMo - A Collaborative Agile Approach to Model and Monitor Service-Based Business Processes”, a project funded by the Operational Programme “Fondo Europeo di Sviluppo Regionale” (FESR) 2007-2013 of the Province of Trento, Italy. The authors would like to thank their collaborators in the ProMo project, and in particular Michele Nori and Roberto Tiella, for their fruitful interactions that have helped with the development of the work here reported. The authors have no declarations of interest to report concerning this paper.

References

References

- Antunes, G., Bakhshandeh, M., Borbinha, J. L., Cardoso, J., Dadashnia, S., Di Francescomarino, C., Dragoni, M., Fettke, P., Gal, A., Ghidini, C., Hake, P., Khiat, A., Klinkmüller, C., Kuss, E., Leopold, H., Loos, P., Meilicke, C., Niesen, T., Pesquita, C., Péus, T., Schoknecht, A., Sheetrit, E., Sonntag, A., Stuckenschmidt, H., Thaler, T., Weber, I., Weidlich, M., 2015. The process model matching contest 2015. In: *Proceedings of the 6th International Workshop on Enterprise Modelling and Information Systems Architectures (EMISA)*. Vol. 248 of LNI. GI, pp. 127–155.
- Baader, F., Horrocks, I., Sattler, U., 2009. Description Logics. In: *Handbook on Ontologies*. Springer, pp. 21–43.
- Batoulis, K., Eid-Sabbagh, R., Leopold, H., Weske, M., Mendling, J., 2013. Automatic business process model translation with BPMT. In: *Proceedings of the Workshops of the 25th International Conference on Advanced Information Systems Engineering (CAiSE)*. pp. 217–228.
- Beheshti, S.-M.-R., Benatallah, B., Motahari-Nezhad, H. R., 2016. Scalable graph-based OLAP analytics over process execution data. *Distributed and Parallel Databases* 34 (3), 379–423.

- Beheshti, S.-M.-R., Benatallah, B., Motahari-Nezhad, H. R., Sakr, S., 2011. A query language for analyzing business processes execution. In: *Proceedings of the 9th International Conference on Business Process Management (BPM)*. Springer Berlin Heidelberg, pp. 281–297.
- Bertoli, P., Corcoglioniti, F., Di Francescomarino, C., Dragoni, M., Ghidini, C., Nori, M., Pistore, M., Tiella, R., 2014. Semantic-based process analysis. In: *Proceedings of the 13th International Semantic Web Conference (ISWC)*. Vol. 8797 of *Lecture Notes in Computer Science*. Springer International Publishing, pp. 228–243.
- Bertoli, P., Di Francescomarino, C., Dragoni, M., Ghidini, C., 2013a. Reasoning-based techniques for dealing with incomplete business process execution traces. In: *Proceedings of the 13th International Conference of the Italian Association for Artificial Intelligence (AI*IA)*. Vol. 8249 of *Lecture Notes in Computer Science*. Springer, pp. 469–480.
- Bertoli, P., Dragoni, M., Ghidini, C., Martufi, E., Nori, M., Pistore, M., Di Francescomarino, C., 2013b. Modeling and monitoring business process execution. In: *Service-Oriented Computing - 11th International Conference, ICSOC 2013, Berlin, Germany, December 2-5, 2013, Proceedings*. Vol. 8274 of *Lecture Notes in Computer Science*. Springer, pp. 683–687.
- Bertoli, P., Kazhamiakin, R., Nori, M., Pistore, M., 2012. SMART: Modeling and monitoring support for business process coordination in dynamic environments. In: *Proceedings of the Workshops of the 15th International Conference on Business Information Systems (BIS)*. Springer, pp. 243–254.
- Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., Xiao, G., 2017a. Ontop: Answering sparql queries over relational databases. *Semantic Web Journal* 8 (3), 471–487, semantic Web Journal outstanding paper award for 2016.
- Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., Savo, D. F., 2011. The MASTRO system for ontology-based data access. *Semant. Web* 2 (1), 43–53.
- Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R., 2009. Conceptual modeling for data integration. In: Borgida, A. T., Chaudhri, V. K., Giorgini, P., Yu, E. S. (Eds.), *Conceptual Modeling: Foundations and Applications – Essays in Honor of John Mylopoulos*. Vol. 5600. Springer Berlin Heidelberg, pp. 173–197.
- Calvanese, D., Kalayci, T. E., Montali, M., Santoso, A., 2017b. OBDA for log extraction in process mining. In: *Tutorial Lectures of the 13th International Summer School Reasoning Web – Semantic Interoperability on the Web*. Springer International Publishing, pp. 292–345.

- Calvanese, D., Kalayci, T. E., Montali, M., Tinella, S., 2017c. Ontology-based data access for extracting event logs from legacy data: The onprom tool and methodology. In: *Business Information Systems*. Springer International Publishing, pp. 220–236.
- De Leoni, M., Maggi, F. M., van der Aalst, W. M. P., 2012. Aligning event logs and declarative process models for conformance checking. In: *Proceedings of the 10th International Conference on Business Process Management (BPM)*. Springer, pp. 82–97.
- Di Francescomarino, C., Ghidini, C., Rospocher, M., 2014a. Evaluating wiki collaborative features in ontology authoring. *IEEE Transactions on Knowledge and Data Engineering* 26 (12), 2997 – 3011.
- Di Francescomarino, C., Ghidini, C., Rospocher, M., Serafini, L., Tonella, P., 2008. Reasoning on semantically annotated processes. In: *Proceedings of the 6th International Conference on Service Oriented Computing (ICSOC)*. Springer, pp. 132–146.
- Di Francescomarino, C., Ghidini, C., Rospocher, M., Serafini, L., Tonella, P., 2009. Semantically-aided business process modeling. In: *Proceedings of the 8th International Semantic Web Conference (ISWC)*. Vol. 5823 of *Lecture Notes in Computer Science*. Springer, pp. 114–129.
- Di Francescomarino, C., Rospocher, M., Ghidini, C., Valerio, A., 2014b. The role of semantic annotations in business process modelling. In: *Proceedings of the 18th Enterprise Computing Conference (EDOC)*. IEEE, pp. 181 – 189.
- Dimitrov, M., Simov, A., Stein, S., Konstantinov, M., 2007. A BPMP based semantic business process modelling environment. In: *Proceedings of the 2nd International Workshop on Semantic Business Process and Product Lifecycle Management (SBPM) at the 4th European Semantic Web Conference (ESWC)*. Vol. 251 of *CEUR-WS*. pp. 1613–0073.
- Fill, H., 2017. SeMFIS: A flexible engineering platform for semantic annotations of conceptual models. *Semantic Web* 8 (5), 747–763.
- Ghidini, C., Di Francescomarino, C., Rospocher, M., Tonella, P., Serafini, L., 2012a. Semantics based aspect oriented management of exceptional flows in business processes. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 42 (1), 25 –37.
- Ghidini, C., Rospocher, M., Serafini, L., 2008. A formalisation of BPMN in description logics. Tech. Rep. TR 2008-06-004, FBK-irst, available at <https://arxiv.org/abs/2109.10716>.
- Ghidini, C., Rospocher, M., Serafini, L., 2012b. Modeling in a wiki with MoKi: Reference architecture, implementation, and usages. *International Journal On Advances in Life Sciences* 4 (3&4), 111–124.

- Guzzo, A., Joaristi, M., Rullo, A., Serra, E., 2021. A multi-perspective approach for the analysis of complex business processes behavior. *Expert Systems with Applications* 177, 114934.
- Hepp, M., Leymann, F., Domingue, J., Wahler, A., Fensel, D., 2005. Semantic business process management: A vision towards using Semantic Web services for business process management. In: *Proceedings of the IEEE International Conference on e-Business Engineering (ICEBE)*. IEEE, pp. 535–540.
- Hepp, M., Roman, D., 2007. An ontology framework for semantic business process management. In: *Proceedings of Wirtschaftsinformatik*. Universitaetsverlag Karlsruhe, pp. 423–440.
- ISO/IEC, 2001. ISO/IEC 9126. Software engineering – Product quality. Standard, ISO/IEC, <https://www.iso.org/standard/22749.html>.
- Koschmider, A., Oberweis, A., 2005. Ontology based business process description. In: *Proceedings of the Workshops of the 17th International Conference on Advanced Information Systems Engineering (CAiSE)*. Lecture Notes in Computer Science. Springer, pp. 321–333.
- Lee, Y.-H., Hu, P. J.-H., Tsao, W.-J., Li, L., 2021. Use of a domain-specific ontology to support automated document categorization at the concept level: Method development and evaluation. *Expert Systems with Applications* 174, 114681.
- Lenzerini, M., Lepore, L., Poggi, A., mar 2018. Metaquerying made practical for OWL 2 QL ontologies. *Information Systems*.
- Marin-Castro, H. M., Tello-Leal, E., 2021. An end-to-end approach and tool for bpmn process discovery. *Expert Systems with Applications* 174, 114662.
- Matassoni, M., Rospocher, M., Dragoni, M., Bouquet, P., 2014. TEX-OWL: a Latex-style syntax for authoring OWL 2 ontologies. In: *Proceedings of the Posters & Demonstrations Track of the 13th International Semantic Web Conference (ISWC)*. Vol. 1272 of *CEUR Workshop Proceedings*. CEUR-WS.org, pp. 357–360.
- Mendling, J., Leopold, H., Pittke, F., 2015. 25 challenges for semantic process modeling. *International Journal of Information Systems and Software Engineering for Big Companies* 1 (1), 78–94.
- Missikoff, M., Proietti, M., Smith, F., 2011. Querying semantically enriched business processes. In: *Proceedings of the 22nd International Conference on Database and Expert Systems Applications (DEXA)*, Part II. Springer Berlin Heidelberg, pp. 294–302.
- Object Management Group, 2008. Business Process Modeling Notation, v1.1 - specification. Standard, OMG, <http://www.omg.org/spec/BPMN/1.1/PDF>.

- Object Management Group, 2011. Business Process Model and Notation (BPMN) version 2.0. Standard, OMG, <http://www.omg.org/spec/BPMN/2.0/PDF>.
- Oliva, J. T., Lee, H. D., Spolaôr, N., Takaki, W. S. R., Coy, C. S. R., Fagundes, J. J., Wu, F. C., 2019. A computational system based on ontologies to automate the mapping process of medical reports into structured databases. *Expert Systems with Applications* 115, 37–56.
- Pedrinaci, C., Domingue, J., de Medeiros, A. K. A., 2008a. A core ontology for business process analysis. In: *Proceedings of the 5th European Semantic Web Conference (ESWC)*. Springer, pp. 49–64.
- Pedrinaci, C., Lambert, D., Wetzstein, B., van Lessen, T., Cekov, L., Dimitrov, M., 2008b. SENTINEL: A semantic business process monitoring tool. In: *Proceedings of the 1st International Workshop on Ontology-supported Business Intelligence (OBI)*. ACM, pp. 1:1–1:12.
- Pedrinaci, C., Markovic, I., Hasibether, F., Domingue, J., 2009. Strategy-driven business process analysis. In: *Proceedings of the 12th International Conference on Business Information Systems (BIS)*. Vol. 21 of *Lecture Notes in Business Information Processing*. Springer, pp. 169–180.
- Rospocher, M., Ghidini, C., Pammer, V., Serafini, L., Lindstaedt, S., 2009. MoKi: the modelling wiki. In: *Proceedings of the 4th Workshop on Semantic Wikis (SemWiki)*. Vol. 464 of *CEUR Workshop Proceedings*. CEUR-WS.org, pp. 113– 127.
- Rospocher, M., Ghidini, C., Serafini, L., 2014. An ontology for the Business Process Modelling Notation. In: *Proceedings of the 8th International Conference on Formal Ontology in Information Systems (FOIS)*. Vol. 267 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, pp. 133 – 146.
- Sahoo, S., Lebo, T., McGuinness, D., 2013. PROV-O: The PROV Ontology. Recommendation, W3C, <https://www.w3.org/TR/2013/REC-prov-o-20130430/>.
- Seaborne, A., Harris, S., 2013. SPARQL 1.1 Query Language. Recommendation, W3C, <https://www.w3.org/TR/2013/REC-sparql11-query-20130321/>.
- Sequeda, J. F., Arenas, M., Miranker, D. P., 2014. OBDA: Query rewriting or materialization? in practice, both! In: *The Semantic Web – ISWC 2014*. Springer International Publishing, pp. 535–551.
- Thomas, O., Fellmann, M., 2007. Semantic EPC: Enhancing process modeling using ontology languages. In: *Proceedings of the 2nd International Workshop on Semantic Business Process and Product Lifecycle Management (SBPM) at the 4th European Semantic Web Conference (ESWC)*. pp. 64–75.

- Tomasi, A., Marchetto, A., Di Francescomarino, C., Susi, A., 2012. reBPMN: Recovering and reducing business processes. In: Proceedings of the 28th IEEE International Conference on Software Maintenance (ICSM). pp. 666–669.
- van der Aa, H., del-Río-Ortega, A., Resinas, M., Leopold, H., Ruiz-Cortés, A., Mendling, J., A. Reijers, H., 2016. Narrowing the business-IT gap in process performance measurement. In: Proceedings of the 28th International Conference on Advanced Information Systems Engineering (CAiSE). Vol. 9694. Springer, pp. 543–557.
- Verbeek, H. M. W., Buijs, J. C. A. M., van Dongen, B. F., van der Aalst, W. M. P., 2010. XES, XESame, and ProM 6. In: Selected Extended Papers of the Information Systems Evolution - CAiSE Forum of the 22nd International Conference on Advanced Information Systems Engineering (CAiSE). Vol. 72 of Lecture Notes in Business Information Processing. Springer, pp. 60–75.
- W3C OWL Working Group, 2012. OWL 2 Web Ontology Language document overview (second edition). W3C recommendation, W3C, <http://www.w3.org/TR/2012/REC-owl2-overview-20121211/>.
- Weber, I., Hoffmann, J., Mendling, J., 2008. Semantic business process validation. In: Proceedings of the 3rd International Workshop on Semantic Business Process Management (SBPM) at the 5th European Semantic Web Conference (ESWC). Vol. 472 of CEUR Workshop Proceedings. CEUR-WS.org.
- Weller, T., Maleshkova, M., 2016. Adaptive semantic process modeling tool. In: Joint Proceedings of the Posters and Demos Track of the 12th International Conference on Semantic Systems (SEMANTiCS) and the 1st International Workshop on Semantic Change & Evolving Semantics (SuCCESS). Vol. 1695 of CEUR Workshop Proceedings. CEUR-WS.org.
- Wetzstein, B., Ma, Z., Filipowska, A., Kaczmarek, M., Bhiri, S., Losada, S., Lopez-Cobo, J.-M., Cicurel, L., 2007. Semantic business process management: A lifecycle based requirements analysis. In: Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM) at the 3rd European Semantic Web Conference (ESWC). Vol. 251 of CEUR Workshop Proceedings. CEUR-WS.org, pp. 1613–0073.
- Wong, P. Y., Gibbons, J., 2009. A relative timed semantics for BPMN. Electronic Notes in Theoretical Computer Science 229 (2), 59–75.
- Xiao, G., Calvanese, D., Kontchakov, R., Lembo, D., Poggi, A., Rosati, R., Zakharyashev, M., 7 2018. Ontology-based data access: A survey. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI). International Joint Conferences on Artificial Intelligence Organization, pp. 5511–5519.

Appendix A. Interviews to experts

In the following we report the main questions asked to the experts during the interviews:

Usability

1. Do you think that the MoKiP interface is *easy to understand*? Did you find it easy retrieving what you were looking for in the tool? Which problems did you find? Do you have suggestions to make the MoKiP interface easier to understand?
2. Do you think that the MoKiP interface is *easy to learn*? How much time did it take for you to learn how to use MoKiP? Which problems did you find? Do you have suggestions to make the MoKiP interface easier to learn?
3. Do you think that the MoKiP interface is *easy to use*? Which problems did you find? Do you have suggestions to make the MoKiP interface easier to use?

Usefulness

1. What are the main strengths you found (and the functionalities you appreciated most) in MoKiP when carrying out your work (e.g., positively affecting efficiency and quality of the work)? Why?
2. What are the main limits you found (and the functionalities you disliked most) in MoKiP when carrying out your work (e.g., negatively affecting efficiency and quality of the work)? Why?

Appendix B. Data Population Without Inference Retraction

We analyze here the data population performance of the MoKiP **system** and the **baseline system** in the alternative (and less realistic) setting where a trace is stored only when completed. In this setting, reasoning is strictly monotonic and inferences have never to be retracted. This permits the **baseline system** to complete the tests also when using the OWL 2 RL inference regime.

Figures B.18a and B.18b show the average execution time of a trace update operation, which corresponds here to the time needed to store a full trace. As for the setting considered in Section 7.2, execution times are lower for the **baseline system** and for RDFS, and they slightly increase with data size. Inference times of the MoKiP **system** are approximately constant except for the smallest data sizes with OWL 2 RL, where inference is faster. As inference times should not depend in principle by data size, this behavior can be possibly explained by a lower overall load of the test machine for small data sizes that improve inference times, this effect progressively disappearing as the data size increases and the inference time stabilizes around a constant value (730 ms).

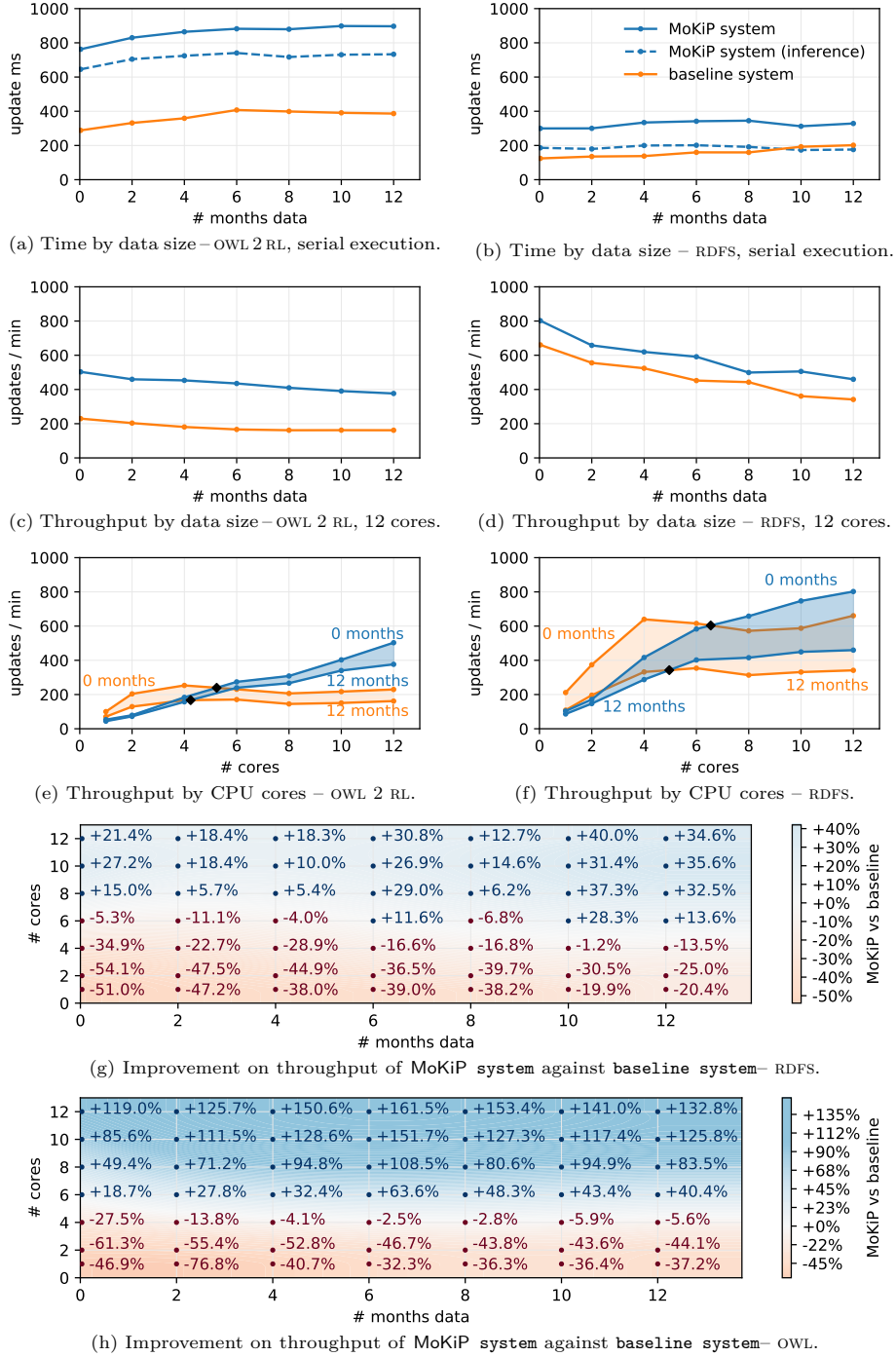


Figure B.18: Population time and throughput of MoKiP and baseline systems with OWL2 RL and RDFS, different data sizes and CPU cores, storing full traces without retracting inferences.

Figures B.18c and B.18d show the population throughput as a function of data size, when using all the 12 CPU cores. Similarly to the results reported in Section 7.2, throughput is better for RDFS, decreases with data size, and always exceeds the minimum requirement set in the nationwide Birth Management scenario (1 full trace update per minute), permitting us to positively answer **RQ_{TS1}** also in this alternative setting, for what concerns data population. The **MoKiP system** consistently outperforms the **baseline system**, both for RDFS and for OWL 2 RL that was not comparable in the setting of Section 7.2.

Figures B.18e and B.18f show the population throughput as a function of the number of available CPU cores, for data sizes of 0 months and 12 months (throughputs for intermediate data size lie in colored areas). Also in this alternative setting, the **MoKiP system** outperforms the **baseline system** when enough cores (~ 6) are used, especially for RDFS, with the throughput of the **MoKiP system** being more insensitive to data size for OWL 2 RL.

Finally, Figures B.18g and B.18h report the percentage throughput improvement (or worsening) of the **MoKiP system** against the **baseline system**, depending on data size and number of CPU cores. As for the default setting considered in Section 7.2, the **baseline system** is competitive only when few CPU cores and/or small data sizes are considered. Improvements of the **MoKiP system** are sensibly higher for OWL 2 RL, suggesting that although the **baseline system** is here applicable to OWL 2 RL, the storage solution of the **MoKiP system** is to be preferred also in this case. All these results permit us to positively answer **RQ_{TS2}** also in this alternative setting.

Appendix C. SPARQL Definition of Example PPIs

We report here the SPARQL listings of the example PPIs referenced in Section 7.2, which are expressed in terms of the classes and properties of the ontological model described in Sections 3 and 4.

All the SPARQL queries reported below share the following namespace declarations, where **bpmn** refers to the BPMN Ontology (core Process Ontology), **domain** refers to the Domain Ontology (Core Domain Ontology and Domain-Specific Ontology), **trace** refers to the Core Trace Ontology, while **ofn** is the prefix associated to the custom SPARQL functions provided by GraphDB and here used to manipulate temporal quantities:

```
PREFIX bpmn: <http://dkm.fbk.eu/index.php/BPMN_Ontology#>
PREFIX domain: <https://dkm.fbk.eu/#>
PREFIX trace: <http://dkm.fbk.eu/Trace_Ontology#>
PREFIX ofn: <http://www.ontotext.com/sparql/functions/>
```

PPI.1. Avg. time per process execution spent by a specific municipality **M**.

```
SELECT (AVG(ofn:asMillis(?trace_end - ?trace_start)) AS ?average_time)
WHERE {
  SELECT ?trace (MIN(?activity_start) AS ?trace_start) (MAX(?activity_end) AS ?trace_end)
  WHERE {
    ?bpd a bpmn:business_process_diagram, domain:BirthManagement;
    bpmn:has_business_process_diagram_pools / bpmn:has_pool_participant_ref <M>.
    ?trace trace:execution_of ?bpd; trace:has_process_trace_flow_object ?executed_activity.
```

```

        ?executed_activity trace:initial_start_dateTime ?activity_start;
        trace:final_end_dateTime ?activity_end.
    }
    GROUP BY ?trace
}

```

PPI.2. Total number of Registration Request documents filled from January, 1st, 2018.

```

SELECT (COUNT(DISTINCT ?data_object) AS ?total)
WHERE {
    ?trace trace:execution_of / a bpmn:business_process_diagram, domain:BirthManagement;
    trace:has_process_trace_data_object ?data_object.
    ?data_object trace:execution_of / a domain:RegistrationRequest;
    ^domain:hasOutput / trace:final_end_dateTime ?activity_end.
    FILTER (?activity_end >= "2018-01-01T00:00:00Z"^^xsd:dateTime)
}

```

PPI.3. Percentage of times in which the flow followed is the one which passes first through the APSS pool and then through the municipality one.

```

SELECT ((COUNT(DISTINCT ?trace_apss) / COUNT(DISTINCT ?trace)) AS ?percentage)
WHERE {
    ?trace trace:execution_of / a bpmn:business_process_diagram, domain:BirthManagement.
    OPTIONAL {
        ?trace trace:has_process_trace_flow_object / trace:execution_of / a bpmn:activity,
        domain:ShowInHospital.
        BIND (?trace AS ?trace_apss)
    }
}

```

PPI.4. Average time spent and number of executions of optional activities by each public office involved in the process. *See Listing 1.*

PPI.5. Number of times in which the municipality sends to SAIA a request without FC.

```

SELECT (COUNT (DISTINCT ?executed_activity) AS ?total)
WHERE {
    ?bpd a bpmn:business_process_diagram, domain:BirthManagement;
    bpmn:has_business_process_diagram_pools [
        bpmn:has_pool_participant_ref / a domain:Municipality;
        bpmn:has_pool_process_ref / bpmn:has_process_graphical_elements ?bpd_activity
    ], [
        bpmn:has_pool_participant_ref / a domain:SAIA;
        bpmn:has_pool_process_ref / bpmn:has_process_graphical_elements ?bpd_event
    ].
    ?bpd_event a bpmn:message_intermediate_event.
    ?bpd_activity a bpmn:activity;
    ^bpmn:has_message_flow_source_ref / bpmn:has_message_flow_target_ref ?bpd_event.
    ?executed_activity trace:execution_of ?bpd_activity; domain:hasOutputField ?field.
    ?field trace:execution_of / a domain:AP5Request.PersonalDetails.CF.
    FILTER NOT EXISTS { ?field domain:value ?value }
}

```

PPI.6. Last event of trace T.

```

SELECT ?executed_activity
WHERE {
  <T> trace:has_process_trace_flow_object ?executed_activity.
  ?executed_activity trace:final_end_dateTime ?activity_end; trace:execution_of / a bpmn:activity.
  FILTER NOT EXISTS {
    <T> trace:has_process_trace_flow_object ?executed_activity2.
    ?executed_activity2 trace:final_end_dateTime ?activity2_end;
    trace:execution_of / a bpmn:activity.
    FILTER (?activity2_end > ?activity_end).
  }
}

```

PPI.7. Cycle time of trace T.

```

SELECT (ofn:asMillis(MAX(?activity_end) - MIN(?activity_start)) AS ?time)
WHERE {
  <T> trace:has_process_trace_flow_object ?executed_activity.
  ?executed_activity trace:initial_start_dateTime ?activity_start; trace:final_end_dateTime ?activity_end.
}

```

PPI.8. Does the trace T pass through the activity labeled with “Present at the hospital”?

```

ASK {
  <T> trace:has_process_trace_flow_object / trace:execution_of / a
  bpmn:flow_object, domain:ShowInHospital.
}

```