A Lightweight RDF Data Model for Business Process Analysis

Marcello Leida, Basim Majeed, Maurizio Colombo, and Andrej Chu

EBTIC (Etisalat BT Innovation Center), Khalifa University, P.O. Box 127788, Abu Dhabi, U.A.E.

Abstract. This article presents a lightweight data representation model designed to support real time monitoring of business processes. The model is based on a shared vocabulary defined using open standard representations (RDF) allowing independence and extremely flexible interoperability between applications. The main benefit of this representation is that it is transparent to the data creation and analysis processes; furthermore it can be extended progressively when new information is available. Business Process data represented with this data model can be easily published on-line and shared between applications. After the definition of the data model, in this article, we demonstrate that with the use of this representation it is possible to retrieve and make use of domain specific information without any previous knowledge of the process. This model is a novel approach to real-time process data representation and paves the road to a complete new breed of applications for business process analysis.

Keywords: business process, ontology, RDF, SPARQL, linked data, real time processing.

1 Introduction

Business Process Management (BPM) is increasingly taking a place as a critical area of information technology, helping businesses to leverage their resources for maximum benefit. It has been successfully applied at improving the performance of business processes:

- by automatically extracting process models from existing systems logs, event logs, database transactions, audit trail events or other sources of information [1,2];
- by allowing instant analysis of the business processes using interactive visual displays of the process workflow;
- by identifying specific case characteristics or trends that influence processing times [3].

However, existing BPM systems tend to limit the kind of data they can analyse and rigid on the data model itself. Moreover, most systems offer very limited real-time analysis capabilities. Furthermore, the kind of analysis that can be

P. Cudre-Mauroux, P. Ceravolo, and D. Gašević (Eds.): SIMPDA 2012, LNBIP 162, pp. 1–23, 2013.

performed is normally restricted to an inextensible and inflexible set of functions tied to the underlying model, besides others challenges identified by the IEEE Task Force on Process Mining [4].

The current technology typically requires the translation or import of data into the system before it can be analysed [1,2,3]. This imposes severe constraints on the way the data is represented and collected. The main limitation is however imposed by schemas that are neither flexible, nor easily extensible, meaning that new information cannot be easily captured by the system without off line modifications; moreover the high complexity of the model used makes such modification expensive. The scope for analysis and information extraction is hence severely limited.

In order to address these issues, in this article we present a lightweight and flexible business process representation model. Lightweight in the sense that the model presented is a very simple conceptualisation of a business process, that can be extended by domain specific conceptualisations. The model we present in this paper is an extremely reduced set of concepts allowing third party client to perform independent process analysis.

The approach is based on the Resource Description Framework (RDF) [5] standard that allows independence between the applications generating business process data and those consuming it. The approach is validated by a proof of concept prototype based on a set of SPARQL [6] queries that demonstrates that the applications can independently work without the need for exchanging domain specific information about the business process monitored.

The article is structured as follows: Section 2 introduces the actual state of the art; Section 3, the main section of the article, describes the basic data model used in our system and a use case scenario where the basic data model is extended by a process monitoring application with domain specific information. Then, Section 4 demonstrates how an analytical tool can perform specific analysis without previous knowledge of the process by using a set of SPARQL queries. Section 5 focuses on how the schema can be used for real time monitoring applications. Section 6 outlines open challenges that the use of such model can introduce. Finally, Section 7 presents final consideration and future work on this area.

2 Related Work

There is a very broad literature and commercial production related to business processes and their representation. However, in this section we will limit the scope to the data models that have been defined starting from requirements such as flexibility, extensibility and suitability for real-time analysis.

There are well known languages, such as BPMN[1] and BPEL[2] that are used to explicitly define a process and capture execution information for fully automated systems (e.g. web service orchestration). However in most cases integration of

[1] `http://www.omg.org/spec/BPMN/2.0/`
[2] `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`

BPEL engines in existing and on-going processes is a non-trivial undertaking and it is often an effort that enterprises prefer to avoid unless a minimum Return of Investment (ROI) is guaranteed. This effort might also be undesirable when the process is not formally captured and exists only in an idealized sense in the mind of the people involved; this is particularly the case for many SMEs.

Therefore, there are many situations where a process model is of very limited use or simply not available: in this case, the process model needs to be inferred from the information generated during its execution, following a bottom-up approach to build the process model from execution data. This is the case of many business process mining tools, such as Aperture [3], ProM [1] and Aris[7].

The data model underpinning Aperture is based on a flexible database model based on the concept ofprocess and task instances, with arbitrary number and type of attributes.

The MXML data format [2] and the Extensible Event Stream (XES) standard[3] used by ProM and Aris are, as for the Aperture model, general purpose, flexible and extensible business process representations.

However, the main issue with this approach is that they are strongly oriented toward a task-based representation, thus it is extremely difficult to analyse the flow from a different point of view. Moreover these models are based on a tree representation, while our model is based on a graph, which is expressively more powerful.

Ontology-based business process representation has also been defined: in [8], the authors define an ontology for semantically annotate business processes: the work focuses on the web service domain and the outcome is a set of ontologies and tools used to analyse the processes. The ontologies clearly describes the processes, the actors (also as organizations) involved, and all the tasks and constraints composing the processes. The target of this work is to use these ontologies to validate process executions by consistency checking on the processes represented as instances of the ontology.

Another example of business process models defined as ontologies is Web Service Modeling Ontology (WSMO) [9], which has been applied in the field of Web Service composition and execution. The approach is very similar to [8] and provides a set of ontologies to be used for business process validation.

The major issue with both approaches is that the process model is already defined by the ontology and the data is usually related to the concept by a mapping. This is an approach that is valid in case of fully automated processes (such as in a Web Service scenario), processes that can be easily deployed and that generate clean data, because these approaches require the data to be translated into the ontological format (if not generated already in the ontological format).

In our approach we do not define a process-specific ontology, rather the ontology is created dynamically starting from the data generated by process executions. Hence we can potentially represent any process without having to make any change to the data model. This particular situation is the one we plan to

[3] `http://www.xes-standard.org/`

challenge with the solution presented in this article: the data model presented is based on the Resource Description Framework (RDF) and is more naturally suited to represent continuously evolving and unpredictable knowledge associated with business processes.

3 An Extensible Business Process Data Model

As already reported in Section 2, the main limitation in actual business process analysis tools is the tight connection between the applications that capture the information about the process and the ones that analyse it, mainly due to the rigid data model used. Nowadays we face the need to deal with increasingly complex systems and monitoring tools; that have to be flexible and robust enough to process also information that is not present or unknown at the moment of deployment or system initialization. Next generation Business Intelligence (BI) systems will need to react to real-time events and predict the process behaviour in order to take corrective actions [10]. Moreover it is necessary to separate the data creation step from the analytical one in terms of meta-data dependency: departments that execute the processes are mostly unaware of the type of analysis that will be performed and they tend not to communicate small changes in the way and quantity of data they capture [3]. On the other side, also the analytical applications do not need to know how the information is captured, as long as it is made available on time and is compliant to a standard representation. Therefore, the need for a less constraining and less rigid model is arising. To this aim, it is important to use an extensible, flexible and publicly available data model. We identified the Resource Description Framework (RDF) [5] and RDF Schema (RDFS) [11] as a very elegant way to solve this issue.

RDF is a standard vocabulary definition which is at the basis of the Semantic Web vision, it is composed of three elements: *concepts*, *relations* between concepts and *attributes* of concepts. These elements are modelled as a labelled oriented graph [12], defined by a set of triples `<s,p,o>` where `s` is *subject*, `p` is *predicate* and `o` is *object*. Formally a graph G can be defined as:

$$G \equiv (U \cup B \cup L) \times U \times (U \cup B \cup L)$$

where:

- U is an infinite set of constant values (called URI references) these have their well-defined semantics provided as an example by the RDF and RDFS vocabularies;
- B is an infinite set of identifiers (called Blank nodes) which identify instantiation of concepts. Elements in this set do not have a defined semantic;
- L is an infinite set of values (called Literals). Elements in this set do not have a defined semantic.

The elements of a triple `<s,p,o>` are respectively: $s \in (U \cup B \cup L)$, $p \in U$, and $o \in (U \cup B \cup L)$. The elements in U represents elements of the schema as in Figure 1, while the elements in B and L are used to represent instances.

An RDF graph is defined by a list of triples and adding new information reduces to append new triples to the list. It is therefore easy to understand why such representation can provide big benefits for real time business process analysis: data can be appended on the fly to the existing one and it will become part of the graph, available for any analytical application, without need for reconfiguration or any other data preparation steps as we will see in Section 5.

RDF is an extremely generic data representation model and it can be used in any domain. The level of flexibility and extensibility offered by the RDF model however requires high computational power in order to query the information in the graph. A system which is able to store and query an RDF graph is called a triple store. It is out of the scope of this article to go in details of the features of a triple store; in order to get a detailed overview the reader should refer to [13] [14] for some comparative surveys. In order to get some comparative measures for query answering times of the main triples stores please refer to [15].

RDF and RDFS standard vocabularies allow external application to query data through SPARQL query language [6]. SPARQL is a standard query language for RDF graphs based on conjunctive queries on triple patterns, which identify paths in the RDF graph. SPARQL is supported by most of the triples stores available.

RDF and RDFS provide a basic set of semantics that is used to define concepts, sub-concepts, relations, attributes, and can be extended easily with any domain-specific information. In the specific case of this article, the application domain is related to Business Processes analysis and in order to provide a point of contact for the data generation and analytical applications, it is important to agree on a common vocabulary defining the set of semantics of this domain: a simple representation allowing a high level of flexibility and at the same time providing a set of elements (concepts, relations and attributes) that are meaningful in the business process analysis domain. We identified as the most generic business process the conceptual model represented in Figure 1.

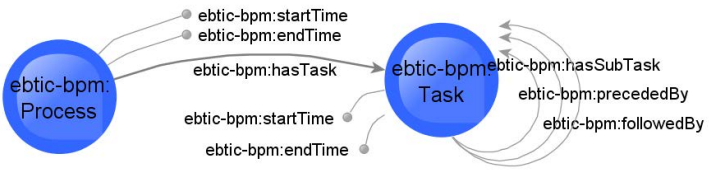

Fig. 1. The conceptual model representing the basic business process representation

The main elements are the concepts: *ebtic-bpm:Process* that represents the business process and *ebtic-bpm:Task*, representing the activity that composes the processthe process. They both have a basic set of attributes representing the beginning and termination of an activity: *ebtic-bpm:startTime* and *ebtic-bpm:endTime*. A relation between these two basic concepts defines the simplest way to represent the workflow of a business process execution: *ebtic-bpm:hasTask*

is defined from the *ebtic-bpm:Process* to the *ebtic-bpm:Task* concepts and represents the set of tasks belonging to a process. Finally, a set of recursive relations are defined on the *ebtic-bpm:Task* concept: *ebtic-bpm:followedBy*, *ebtic-bpm:precededBy* and *ebtic-bpm:hasSubTask*. They respectively indicate which tasks precede and follow a given one and which tasks are subtasks of a given one. The *ebtic-bpm:followedBy* and *ebtic-bpm:precededBy* relations are used to build the process workflow while the *ebtic-bpm:hasSubTask* is useful in order to define stages of execution and to ease the analysis and structural organisation of complex processes. It is important also to underline the fact that the representation of Figure 1 does not contain any data, it provides only a conceptual schema that the process monitoring applications will extend and instantiate as described in Section 3.1.

The conceptual model of Figure 1 can be formalised by a graph G defined by the set of triples in Table 1[4]. The elements in the conceptual model of Figure 1 are defined as a vocabulary EBTIC-BPM that extends the set U which already contains the vocabularies RDF and RDFS.

Table 1. The list of triples that defines the basic Business process representation with elements from EBTIC-BPM vocabulary

s	p	o
ebtic-bpm:hasTask	rdfs:range	ebtic-bpm:Task
ebtic-bpm:hasTask	rdfs:domain	ebtic-bpm:Process
ebtic-bpm:precededBy	rdfs:domain	ebtic-bpm:Task
ebtic-bpm:precededBy	rdfs:range	ebtic-bpm:Task
ebtic-bpm:followedBy	rdfs:domain	ebtic-bpm:Task
ebtic-bpm:followedBy	rdfs:range	ebtic-bpm:Task
ebtic-bpm:hasSubTask	rdfs:domain	ebtic-bpm:Task
ebtic-bpm:hasSubTask	rdfs:range	ebtic-bpm:Task
ebtic-bpm:endTime	rdfs:range	xs:dateTime
ebtic-bpm:endTime	rdfs:domain	ebtic-bpm:Process
ebtic-bpm:endTime	rdfs:domain	ebtic-bpm:Task
ebtic-bpm:startTime	rdfs:range	xs:dateTime
ebtic-bpm:startTime	rdfs:domain	ebtic-bpm:Task
ebtic-bpm:startTime	rdfs:domain	ebtic-bpm:Process

The RDF graph resulting by these triples is displayed in Figure 2: the nodes represents the elements from the $(U \cup B \cup L)$, the number of the connections represents the triple identifiers, and the label of connection identifies the triple part: the label `SP` is the part of the triple connecting *subject* to *object* and the `PO` is the part of the triple connecting *predicate* to *object*.

According to **Linked Data** [16] principles the `ebtic-bpm` RDF graph will be made available on-line and advertised as a publicly available schema. So that business process data can be published and shared using this representation. This will allow consumer application to be able to process any business process defined according to this extensible schema, as described in Section 4.

[4] The elements with xs: preamble are part of the XML Schema vocabulary (`http://www.w3.org/XML/Schema`).

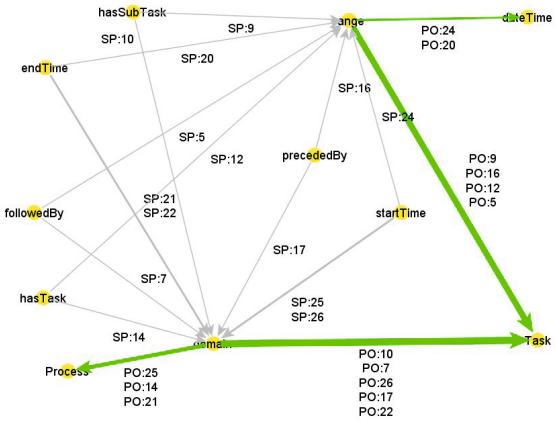

Fig. 2. The RDF graph resulting by the triples in Table 1

3.1 Extending the EBTIC-BPM Vocabulary

The conceptual model defined by the triples in Table 1 is generic and does not provide any domain specific information that can be used for a meaningful business process analysis; this information needs to be provided independently by the applications that monitor the process execution and defined as an extension of the EBTIC-BPM vocabulary. We will not go into the details of this application since it depends on the specific deployment configuration, as an example this application could be deployed on a SOA environment and monitor the message exchange on an Enterprise Service Bus (ESB), it could monitor a set of log files, it could be defined as a trigger in a database and so on. What is relevant is that the information captured by this application has to be interpreted and translated into triples.

The process monitor will act as a bridge between the specific execution environment of the process to monitor and the triple store where the triples will be directed to; in order to make accessible the information to other applications through the SPARQL interface of the triples store as described more in detail in section 3.2.

As an example let us consider the process of building a product A; this product is composed by a set of components that need to be assembled. Once the product is assembled, its correct behaviour is verified by a testing activity. It is possible to capture the semantic of this process by defining a vocabulary Product-A (PA) that enriches the basic process model EBTIC-BPM as represented by the conceptual model in Figure 3. This vocabulary is not a rigid process model definition such as BPML but just the list of attributes, relations and tasks that *may* appear during the execution of the process.

Modelling the information as an RDF graph allows monitoring applications to provide also an incomplete vocabulary or even not to provide any vocabulary at all during start time, and update it when a new concept, task or an attribute appears during the execution of the process; hence also the process definition

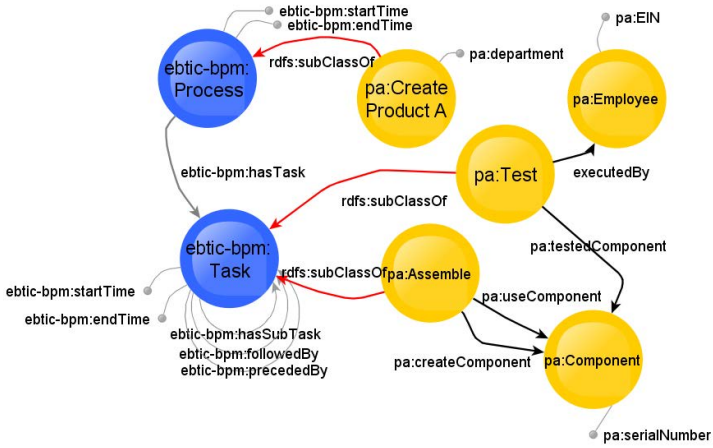

Fig. 3. The EBTIC-BPM conceptual model extended with domain specific concepts

is provided on-line. This is an important assumption that allows our approach to adapt to situations where the process is not known a priori and where its behaviour is not predictable or cannot be bounded. Updating the vocabulary in this model is equivalent to appending the related triples to the graph and there is no constraint of any type on when these triples should appear in the graph. As it is also possible to notice from Figure 4, this action is transalted in adding nodes and relations to the graph.

The domain specific vocabulary is composed by a set of concepts, relations and attributes:

- *pa:CreateProductA*: this concept extends the concept *ebtic-bpm:Process* using the *rdfs:subClassOf* relation and its semantic indicates the type of process (creation of a product of type A).
  This concept has a domain specific attribute *pa:department* which represents the identifier of the department that executed the process.
- *pa:Assemble*: this concept extends the concept *ebtic-bpm:Task* (with the *rdfs:subClassOf* relation) and represents the activity of assembling a set of components (indicated by the relation *pa:useComponent*) into a new component (represented by the relation *pa:createComponent*).
- *pa:Component*: this concept represents the component that is used/created by the assembling task. This concept has an attribute *pa:serialNumber* which indicates the serial number of the component.
- *pa:Test*: this concept extends the concept *ebtic-bpm:Task* and represents the activity of testing the final component (indicated by the *pa:testedComponent* relation). As represented by the *pa:executedBy* relation, this activity is carried out by an employee.
- *pa:Employee*: this concept represents the employee which performs the test of the final product. This concept has an attribute *pa:EIN* which represents the identification number of the employee.

Table 2. The list of triples that extends the EBTIC-BPM vocabulary with domain specific information

s	p	o
pa:CreateProductA	rdfs:subClassOf	ebtic-bpm:Process
pa:Assemble	rdfs:subClassOf	ebtic-bpm:Task
pa:Test	rdfs:subClassOf	ebtic-bpm:Task
pa:Assemble	rdfs:domain	pa:useComponent
pa:useComponent	rdfs:range	pa:Component
pa:Assemble	rdfs:domain	pa:createComponent
pa:createComponent	rdfs:range	pa:Component
pa:Test	rdfs:domain	pa:testedComponent
pa:testedComponent	rdfs:range	pa:Component
pa:Test	rdfs:domain	pa:executedBy
pa:executedBy	rdfs:range	pa:Employee
pa:CreateProductA	rdfs:domain	pa:department
pa:department	rdfs:range	xs:String
pa:Component	rdfs:domain	pa:serialNumber
pa:serialNumber	rdfs:range	xs:Integer
pa:Employee	rdfs:domain	pa:EIN
pa:EIN	rdfs:range	xs:Integer

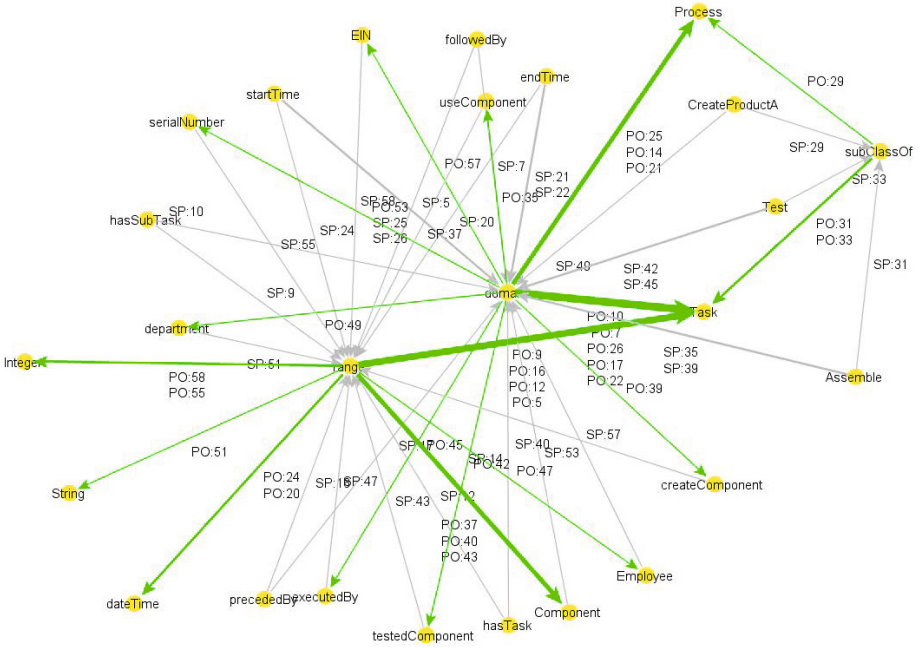

Fig. 4. The RDF graph resulting by adding to the existing graph the the triples in Table 2

This domain specific information is used to extend the basic process model by appending the triples in Table 2 to the ones present in the triple store (the triples in Table 1 that will be present after the bootstrap procedure). Figure 4 shows the complete RDF graph resulting by the union of the triples in Table 1 and in Table 2.

3.2 Architecture of a Sample Deployment

A typical deployment of a system based on the data model described so far is represented in Figure 5. One or more process monitor is deployed as a non invasive application and used to monitor the process activity; non invasive means that the deployment of the process monitor will require none or very limited change in the existing system. The process monitor creates and maintains the extension of the EBTIC-BPM vocabulary while capturing process execution data: this extension need to be present in the triple store as well as the EBTIC-BPM vocabulary: it can be provided during system bootstrap procedure or created on the fly, by the process monitor itself by analysing the process execution data, and

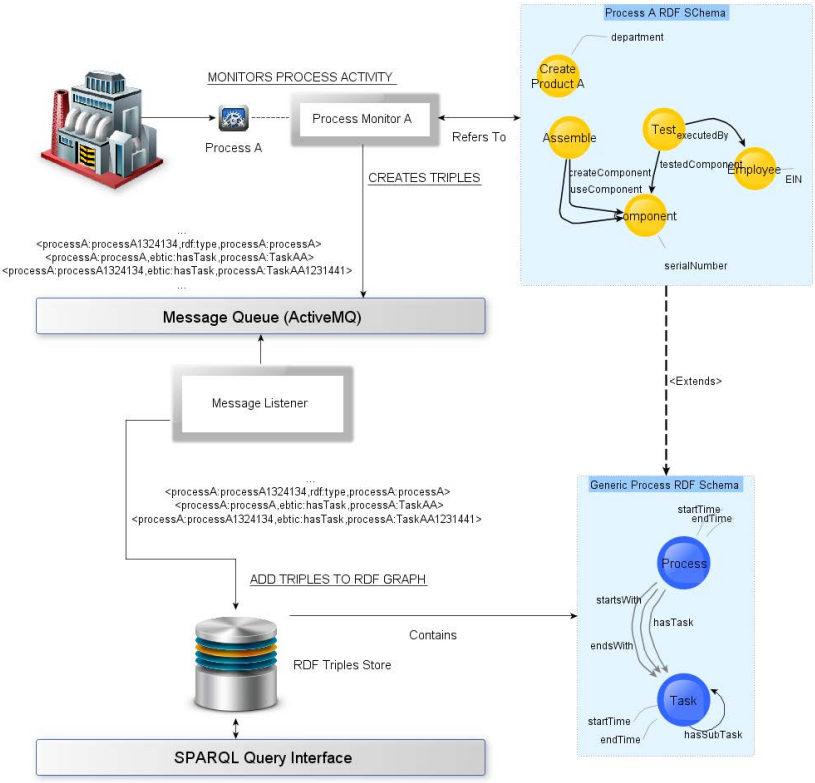

Fig. 5. A typical deployment of the system

Table 3. An example of a log file contining process activity

TimeStamp	TransactionBegin	ID	TN	Dep.	SN	EIN
2012-03-14T03:55:46	2012-03-14T03:18:56	01	Assemble	DBX	23442	
2012-03-14T04:15:06	2012-03-14T04:01:56	01	Assemble	DBX	21232	
2012-03-14T04:55:26	2012-03-14T04:18:56	01	Assemble	DBX	21232	
2012-03-14T05:15:16	2012-03-14T04:56:58	01	Test	DBX	21232	760506653
2012-03-14T06:55:26	2012-03-14T06:18:56	01	Assemble	DBX	21232	
2012-03-14T06:58:25	2012-03-14T07:15:26	01	Test	DBX	21232	760506653

Table 4. The triples defining one instance of execution of a process, as created by a monitoring application

s	p	o
pa:01	rdf:type	pa:CreateProductA
pa:01	pa:department	DBX"^^xsd:string
pa:01	ebtic-bpm:startTime	2012-03-14T03:18:56+04:00"^^xsd:date
pa:0045	rdf:type	pa:Assemble
pa:0045	ebtic-bpm:startTime	2012-03-14T03:18:56+04:00"^^xsd:date
pa:0098	rdf:type	pa:Component
pa:0098	pa:serialNumber	23442"^^xsd:integer
pa:0045	pa:createComponent	pa:0098
pa:0045	ebtic-bpm:endTime	2012-03-14T03:55:46+04:00"^^xsd:date
pa:01	ebtic-bpm:hasTask	pa:0045
pa:0045	ebtic-bpm:followedBy	pa:0046
pa:0046	ebtic-bpm:followedBy	pa:0047
pa:0047	ebtic-bpm:precededBy	pa:0046
pa:0056	ebtic-bpm:precededBy	pa:0049
pa:0046	rdf:type	pa:Assemble
pa:0046	ebtic-bpm:startTime	2012-03-14T04:01:56+04:00"^^xsd:date
pa:0099	rdf:type	pa:Component
pa:0099	pa:serialNumber	21232"^^xsd:integer
pa:0046	pa:useComponent	pa:0098
pa:0046	pa:createComponent	pa:0099
pa:0046	ebtic-bpm:endTime	2012-03-14T04:15:06+04:00"^^xsd:date
pa:0047	rdf:type	pa:Assemble
pa:01	ebtic-bpm:hasTask	pa:0046
pa:0047	ebtic-bpm:startTime	2012-03-14T04:18:56+04:00"^^xsd:date
pa:0047	pa:useComponent	pa:0099
pa:0048	ebtic-bpm:precededBy	pa:0047
pa:0047	ebtic-bpm:followedBy	pa:0048
pa:0048	ebtic-bpm:followedBy	pa:0049
pa:0049	ebtic-bpm:followedBy	pa:0056
pa:0046	ebtic-bpm:precededBy	pa:0045
pa:0049	ebtic-bpm:precededBy	pa:0048
pa:0047	pa:createComponent	pa:0099
pa:0047	ebtic-bpm:endTime	2012-03-14T04:55:26+04:00"^^xsd:date
pa:0048	rdf:type	pa:Test
pa:01	ebtic-bpm:hasTask	pa:0049
pa:0048	ebtic-bpm:startTime	2012-03-14T04:56:58+04:00"^^xsd:date
pa:0048	pa:testedComponent	pa:0099
pa:00129	rdf:type	pa:Employee
pa:00129	pa:EIN	760506653"^^xsd:integer
pa:0048	pa:executedBy	pa:00129
pa:0048	ebtic-bpm:endTime	2012-03-14T05:15:16+04:00"^^xsd:date
pa:0049	rdf:type	pa:Assemble
pa:01	ebtic-bpm:hasTask	pa:0047
pa:0049	ebtic-bpm:startTime	2012-03-14T06:18:56+04:00"^^xsd:date
pa:0049	pa:useComponent	pa:0099
pa:0049	pa:createComponent	pa:0099
pa:0049	ebtic-bpm:endTime	2012-03-14T06:55:26+04:00"^^xsd:date
pa:0056	rdf:type	pa:Test
pa:01	ebtic-bpm:hasTask	pa:0048
pa:0056	ebtic-bpm:startTime	2012-03-14T06:58:25+04:00"^^xsd:date
pa:0056	pa:testedComponent	pa:0099
pa:0056	pa:executedBy	pa:00129
pa:01	ebtic-bpm:hasTask	pa:0056
pa:0056	ebtic-bpm:endTime	2012-03-14T07:15:26+04:00"^^xsd:date
pa:01	ebtic-bpm:endTime	2012-03-14T07:15:26+04:00"^^xsd:date

submitted gradually to the triple store. When the process monitor intercept process activity information, it creates the triples representing such activity and continuously sends the triples to a message queue (for example Apache ActiveMQ[5]).

[5] http://activemq.apache.org/

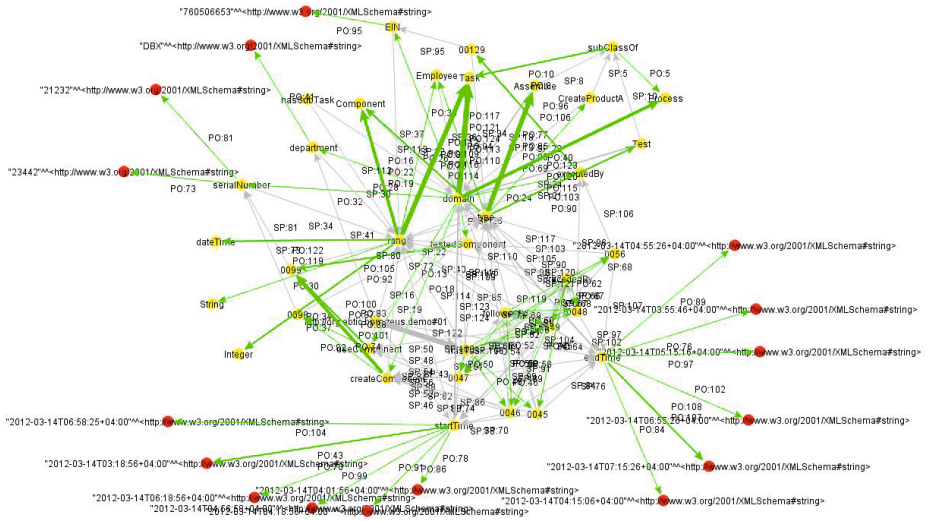

Fig. 6. The RDF graph resulting by the union of the triples of Table 1, 2 and 4

A message listener continuously receives the triples sent to the message queue by the various process monitors and inserts the triples into the triple store. The triple store exposes a SPARQL query service allowing external applications to query the continuously updated process execution graph.

As discussed, the monitoring application is responsible of creating the extensions of the EBTIC-BPM vocabulary and once a process instance is executing, the process monitor captures this information and creates the corresponding triples to be sent to the triple store.

For the sake of simplicity we will not consider in this paper named graphs or contexts of a triple. We assume the triples created by the process monitors will contribute to compose the same RDF graph. We will demonstrate that even without the use of contexts we can distinguish between different process models in the same graph. Introducing named graphs or contexts will add separation between RDF graphs, but the approach described is still valid.

In this deployment, the process monitoring application generates a flow of triples representing the process activity. The logic used to translate process activity into triples is contained in the Process Monitor and it varies depending on the process. Imagine a log file continuously updated by a workflow management system containing process activity as the on in Table 3.

Every line in the log represent a process activity; every time a line is added to the log, the process monitor will analyse the line and crate the respective triples. The final list of triples representing the log execution of Table 3 is reported in Table 4.

Figure 6 shows the complete RDF graph containing all the nodes and connections representing the execution of a process; as it is possible to notice, already with only one process execution the graph starts to become complex.

Once the triples are present in the triple store, it will be possible for other applications such as business process mining or monitoring applications to extract and analyse business process information through the SPARQL interface.

The EBTIC-BPM vocabulary introduced in section 3 provides a generic business process domain vocabulary that allows to define SPARQL queries to be used for data discovery process, but it is necessary to ensure that the concepts, relations and attributes in Figure 1 are always present in the graph when the process monitors become operative. Therefore we assume that the system will be initialised by a bootstrap procedure that will insert the triples of Table 1 in the triple store.

4 A SPARQL-Based Discovery Process

As a proof of concept application we developed a business process visualisation tool, which is based on a set of predefined SPARQL queries defined with the assumption that the EBTIC-BPM vocabulary is present in the graph. Hence the queries are independent with respect to the domain specific information of the process to analyse.

This application connects to a SPARQL endpoint and is used to display business process flows under different point of views. It has been developed from a set of recurrent requirements gathered with the experience our team developed from down-streaming of the business process mining tool [3]. These requirements were focused on a more flexible visualisation of the process workflow and the possibility to monitor real time process executions. The use of the data model described in this article allows to fulfil both requirements successfully.

This tool will initially display the list of domain specific processes that are available in the triple store, which the user can select and obtain the instances of that process, inspect their attributes, and list of tasks with their attributes, related concepts, and subtasks. Also by clicking on a process instance the application will display the workflow diagram of the selected instance of execution. All the interactions of the user with the application are translated into SPARQL queries that are submitted to the triple store. This application processes the results and presents them to the user according to the selected action.

These SPARQL queries are defined only with knowledge of EBTIC-BPM vocabulary, but we will now demonstrate that the tool can visualise and make use of domain specific information without prior knowledge of it.

Once the application is started, a set of SPARQL queries is executed in order to obtain some information about the processes that are present in the store; the following query:

```
SELECT ?process                                                    (1)
WHERE { ?process rdfs:subClassOf ebtic-bpm:Process.}
```

returns the concepts that extend the *ebtic-bpm:Process* concept. Each of these concepts represents a different process type that is present in the RDF graph, in case of the current example the results in the variable `?process` will contain

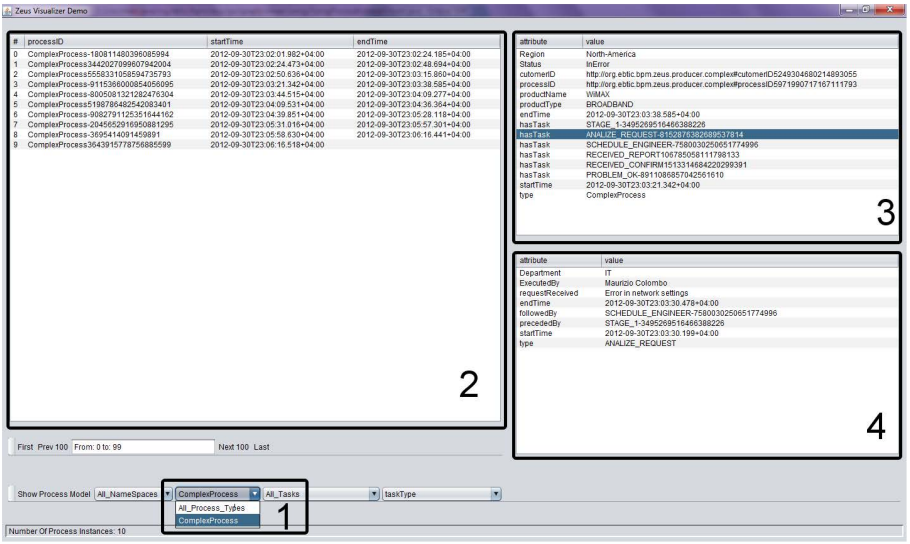

Fig. 7. The main interface of the business process visualisation tool

`pa:CreateProductA`. The results of this query will be displayed in a list (area 1 in Figure 7) and the user will be able to select which type of processes to analyse. Once the user selects an entry in the process list, a second query will be executed:

```
SELECT ?processID ?startTime ?endTime                                    (2)
WHERE { ?processID rdf:type pa:CreateProductA.
        ?processID ebtic-bpm:startTime ?startTime.
        OPTIONAL { ?processID ebtic-bpm:endTime ?endTime.}}
```

this query returns information (`?processID ?startTime ?endTime`) about the instances of the process *pa:CreateProductA*, in case the user selected such process from the list populated with the results of the previous query. In case the user did not select any specific process type then the query that will be submitted will be slightly different:

```
SELECT ?processID ?startTime ?endTime                                    (3)
WHERE { ?processID rdf:type ?process.
        ?process rdfs:subClassOf ebtic-bpm:Process.
        ?processID ebtic-bpm:startTime ?startTime.
        OPTIONAL { ?processID ebtic-bpm:endTime ?endTime.}}
```

this modified version combines the query 1, used to retrieve the concepts extending the basic process concept and the query 2 which is used to retrieve the instances of a specific process concept, thus the query will return information

about all the process instances that are present in the graph; the query 3 can be slightly simplified in case the triple store supports RDFS inference.

The results of this second query will be used to populate a table in the graphical interface of the application (area 2 in Figure 7), where the user will be able to see the list of instances of the selected process (`pa:CreateProductA` in this case). The `OPTIONAL` keyword indicates that the ending time of a process may not be present, in case the process is still executing (as it is possible to notice in the last row of the table in area 2 in Figure 7). In case `OPTIONAL` is not present the results will contain only process instances that have terminated.

At this point the user, by interacting with this list can select one process instance, as an example `pa:01`; this action will fire another query:

```
SELECT ?attribute ?value                                    (4)
WHERE { pa:01 ?attribute ?value.
        FILTER (?attribute != rdf:type)}
```

which will populate another table with the names and values of all the attributes of the selected process and the list of its tasks (area 3 in Figure 7).

It is important to stress on the fact that also in this case the query will return all the process attributes and tasks of that specific process execution, without any knowledge of the process meta-data. The `FILTER` keyword is used to remove the attributes of type `rdf:type` from the results if they are not meaningful for the final user. The use of `pa:01` in the first triple pattern is used to ensure that the results returned are relevant to the selected process instance.

The user can now select a row in the resulting table, this action will execute the query 4 modified by replacing the `pa:01` element with the value of the row selected by the user. In case the value represents a task instance, the results of this query will populate a table with the names and the values of all the attributes of the selected task (area 4 in Figure 7)

A more interesting behaviour occurs when the user double clicks on one process instance to obtain the process workflow. This action will be converted by the application into the following SPARQL query:

```
SELECT ?PID ?startTime ?endTime ?taskID                      (5)
       ?taskType ?follBy ?precBy
WHERE { ?PID ebtic-bpm:hasTask ?taskID.
        ?taskID rdf:type ?taskType.
        ?PID ebtic-bpm:startTime ?startTime.
OPTIONAL {?PID ebtic-bpm:endTime ?endTime.}.
OPTIONAL { ?taskID ebtic-bpm:followedBy ?follBy.}.
OPTIONAL { ?taskID ebtic-bpm:precededBy ?precBy.}.
FILTER (?PID = pa:01)}
```

where `pa:01` is the selected process instance.

The output of query 5, in case the RDF graph contains the triples of Table 1, 2 and 4 is reported in Table 5

Table 5. A sample result set from the execution of query 5

PID	startTime	endTime	taskID	taskType	follBy	precBy
pa:01	2012-03-14T03:18:56+04:00	2012-03-14T07:15:26+04:00	pa:0045	pa:Assemble	pa:0046	
pa:01	2012-03-14T03:18:56+04:00	2012-03-14T07:15:26+04:00	pa:0046	pa:Assemble	pa:0047	pa:0045
pa:01	2012-03-14T03:18:56+04:00	2012-03-14T07:15:26+04:00	pa:0047	pa:Assemble	pa:0048	pa:0046
pa:01	2012-03-14T03:18:56+04:00	2012-03-14T07:15:26+04:00	pa:0048	pa:Test	pa:0049	pa:0047
pa:01	2012-03-14T03:18:56+04:00	2012-03-14T07:15:26+04:00	pa:0049	pa:Assemble	pa:0056	pa:0048
pa:01	2012-03-14T03:18:56+04:00	2012-03-14T07:15:26+04:00	pa:0056	pa:Test		pa:0049

The result of this query is processed by a method that returns a graphical representation of the process built using the information in the `?followedBy` and `?precededBy` variables. The method generates a standard GraphML [17] document that can be consumed by any graphical library supporting such standard (Jung[6], yFiles[7], jsPlumb[8], D3[9] just to cite few of them). The output will be returned to the user in a new window as in the example of Figure 8. Our tool makes use of the yFiles library to display the GraphML document. All the workflow diagrams presented in this paper has been generated by our tool; in the figures the window frame has been cropped out.

The process workflow in Figure 8 is an interactive object that the user can manipulate: as an example switching the task-based view by selecting alternative ways to present the process. This can be done at *process* or *task* level.

4.1 Process Level View Interaction

This aspect refers to the interaction of the user with the graphical representation of the global process workflow. The right click of the user on any point of the window (which is not a task), is translated in a query that is used to retrieve all the attributes and objects related to all the tasks present in the workflow. This is an important feature of this approach because it ensures that the user will only be able to choose between task attributes that are present in the process instance the user is interacting with. This is encoded in the following query:

```
SELECT DISTINCT ?attribute                                    (6)
WHERE { pa:01 ebtic-bpm:hasTask ?taskID.
?taskID rdf:type ?Task.
?Task rdfs:subClassOf ebtic-bpm:Task.
?Task rdfs:domain ?attribute.}
```

The results of this query will populate a pop-up box where the user can select an element that will be used to switch the process workflow from a task-based representation to an attribute-based representation, according to the attribute selected. In case of process level view interaction, if the selected attribute is not present in a task the usual task-based representation will be used for that task.

[6] http://jung.sourceforge.net/
[7] http://www.yworks.com/en/products_yfiles_about.html
[8] http://jsplumb.org
[9] http://d3js.org/

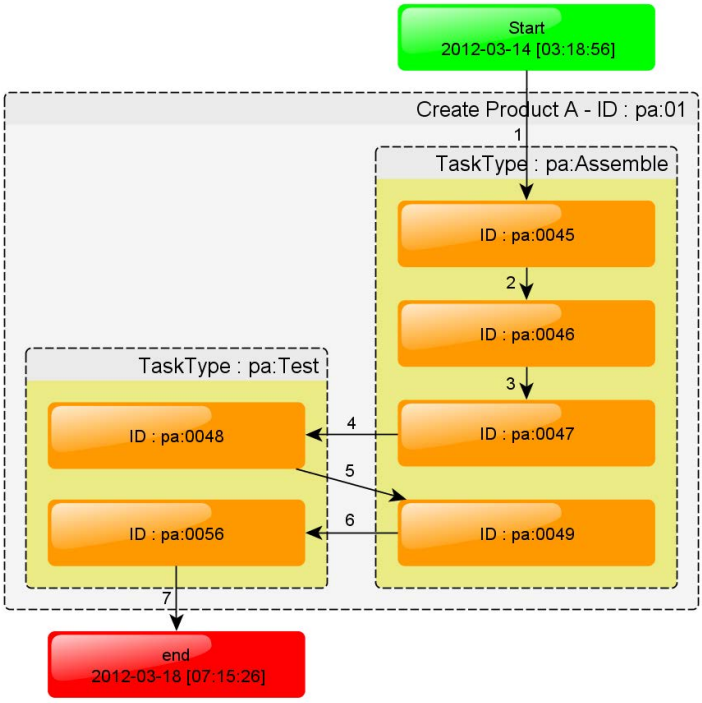

Fig. 8. The process workflow of a process instance

This behaviour is automatically encoded in the query that will be fired once the user selects an attribute from the pop-up list. The query is similar to the previous one, with one additional variable `?alternativeName` that will be used to replace the task name in case the attribute used to switch the workflow is present in the task. For example, if the user selects the element `pa:createComponent` from the pop up list; the SPARQL query that will be execute will be:

```
SELECT ?PID ?startTime ?endTime ?taskID                    (7)
       ?taskType ?follBy ?precBy ?altName
WHERE { ?PID ebtic-bpm:hasTask ?taskID.
        ?taskID rdf:type ?taskType.
        ?PID ebtic-bpm:startTime ?startTime.
OPTIONAL {?PID ebtic-bpm:endTime ?endTime.}.
OPTIONAL { ?taskID ebtic-bpm:followedBy ?follBy.}.
OPTIONAL { ?taskID ebtic-bpm:precededBy ?precBy.}.
OPTIONAL { ?taskID pa:createComponent ?altName.}.
FILTER (?PID = pa:01)}
```

The difference with query 5 is that there is an additional optional triple pattern created using the attribute selected by the user (`pa:createComponent` in this case) linked to the task instance (by the `?taskID` variable). This additional pattern will bind the results to the `?alternativeName` variable in case the path in the RDF graph exists.

The output of query 7, in case the RDF graph contains the triples of Table 1, 2 and 4 is reported in Table 6.

Table 6. A sample result set from the execution of query 5

PID	startTime	endTime	taskID	taskType	follBy	precBy	altName
pa:01	2012-03-14T07:...	2012-03-14T07:...	pa:0045	pa:Assemble	pa:0046		pa:0098
pa:01	2012-03-14T07:...	2012-03-14T07:...	pa:0046	pa:Assemble	pa:0047	pa:0045	pa:0099
pa:01	2012-03-14T07:...	2012-03-14T07:...	pa:0047	pa:Assemble	pa:0048	pa:0046	pa:0099
pa:01	2012-03-14T07:...	2012-03-14T07:...	pa:0048	pa:Test	pa:0049	pa:0047	
pa:01	2012-03-14T07:...	2012-03-14T07:...	pa:0049	pa:Assemble	pa:0056	pa:0048	pa:0099
pa:01	2012-03-14T07:...	2012-03-14T07:...	pa:0056	pa:Test		pa:0049	

The results will be processed to create the GraphML document representing the workflow in Figure 9. The algorithm used to generate the GraphML document will use the value in `?alternativeName`, if present, to represents the task name, using also a different colour to fill the element in the workflow.

4.2 Task Level View Interaction

This second aspect refers to the recursive interaction of the user with single task type attributes in order to modify the graphical representation of the process workflow.

Imagine a situation for which the user is interested in a certain attribute of a specific task type and another attribute of another task type.

The user, by right clicking on a task, will fire a query very similar to query 6 which will be used to retrieve all the attributes and objects related to that specific task. This is done by replacing the variable `?Task` in the query with the specific task (`pa:createComponent`), consequently narrowing the set of results.

These results are used to populate a pop-up box as, described above, and the user can select an attribute that will be used to replace the traditional task-based representation for that specific task.

The action of clicking on an attribute will be translated to a query conceptually similar to query 7, with the difference that this time we need to make wide use of the `UNION` construct in order to compose the results of different queries: one query that will be focused on the alternative value referred to the task selected by the user, while a second query that will take care of the tasks that are

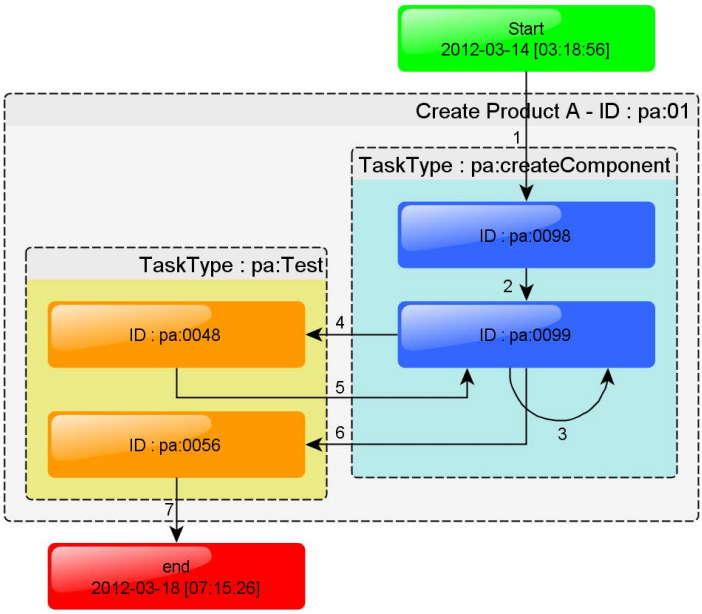

Fig. 9. The process workflow using switched view by the id of the component created by the task

not of the type selected by the user, which will be displayed with the traditional task-based view. This behaviour is defined by the following query:

```
SELECT ?PID ?startTime ?endTime ?taskID                              (8)
       ?taskType ?follBy ?precBy ?altName
WHERE {{ ?PID ebtic-bpm:hasTask ?taskID.
         ?taskID rdf:type ?taskType.
         ?PID ebtic-bpm:startTime ?startTime.
OPTIONAL {?PID ebtic-bpm:endTime ?endTime.}.
OPTIONAL { ?taskID ebtic-bpm:followedBy ?follBy.}.
OPTIONAL { ?taskID ebtic-bpm:precededBy ?precBy.}.
OPTIONAL { ?taskID pa:createComponent ?altName.}.
FILTER (?PID = pa:01 && ?taskType = pa:Assemble)}
UNION { ?PID ebtic-bpm:hasTask ?taskID.
         ?taskID rdf:type ?taskType.
         ?PID ebtic-bpm:startTime ?startTime.
OPTIONAL {?PID ebtic-bpm:endTime ?endTime.}.
OPTIONAL { ?taskID ebtic-bpm:followedBy ?follBy.}.
OPTIONAL { ?taskID ebtic-bpm:precededBy ?precBy.}.
FILTER (?PID = pa:01 && ?taskType != pa:Assemble)}}
```

This action can be iteratively repeated by the user for every task type present in the workflow diagram, so that each different task type can be represented using a different attribute.

The actions of the user are translated into additional queries that are connected to the one defined so far by the UNION keyword as shown in query 9.

```
SELECT ?PID ?startTime ?endTime ?taskID                              (9)
       ?taskType ?follBy ?precBy ?altName
WHERE {{ ?PID ebtic-bpm:hasTask ?taskID.
         ?taskID rdf:type ?taskType.
         ?PID ebtic-bpm:startTime ?startTime.
OPTIONAL {?PID ebtic-bpm:endTime ?endTime.}.
OPTIONAL { ?taskID ebtic-bpm:followedBy ?follBy.}.
OPTIONAL { ?taskID ebtic-bpm:precededBy ?precBy.}.
OPTIONAL { ?taskID pa:createComponent ?altName.}.
FILTER (?PID = pa:01 && ?taskType = pa:Assemble)}
UNION { ?PID ebtic-bpm:hasTask ?taskID.
         ?taskID rdf:type ?taskType.
         ?PID ebtic-bpm:startTime ?startTime.
OPTIONAL {?PID ebtic-bpm:endTime ?endTime.}.
OPTIONAL { ?taskID ebtic-bpm:followedBy ?follBy.}.
OPTIONAL { ?taskID ebtic-bpm:precededBy ?precBy.}.
OPTIONAL { ?taskID pa:executedBy ?altName.}.
FILTER (?PID = pa:01 && ?taskType = pa:Test)}
UNION { ?PID ebtic-bpm:hasTask ?taskID.
         ?taskID rdf:type ?taskType.
         ?PID ebtic-bpm:startTime ?startTime.
OPTIONAL {?PID ebtic-bpm:endTime ?endTime.}.
OPTIONAL { ?taskID ebtic-bpm:followedBy ?follBy.}.
OPTIONAL { ?taskID ebtic-bpm:precededBy ?precBy.}.
FILTER (?PID = pa:01 && ?taskType != pa:Assemble
            && ?taskType != pa:Test)}}
```

The output of query 9, in case the RDF graph contains the triples of Table 1, 2 and 4 is reported in Table 7.

The results will be processed to create the GraphML document representing the workflow in Figure 10.

This set of queries demonstrates how the visualiser application is able to extract and make use of domain specific information to create useful user interfaces and rich process workflows, by automatically creating SPARQL queries with only the initial knowledge of the EBTIC-BPM vocabulary. As previously pointed out

Table 7. A sample result set from the execution of query 6.

PID	startTime	endTime	taskID	taskType	follBy	precBy	altName
pa:01	2012-03-14T03:...	2012-03-14T07:...	pa:0045	pa:Assemble	pa:0046		pa:0098
pa:01	2012-03-14T03:...	2012-03-14T07:...	pa:0046	pa:Assemble	pa:0047	pa:0045	pa:0099
pa:01	2012-03-14T03:...	2012-03-14T07:...	pa:0047	pa:Assemble	pa:0048	pa:0046	pa:0099
pa:01	2012-03-14T03:...	2012-03-14T07:...	pa:0049	pa:Assemble	pa:0056	pa:0048	pa:0099
pa:01	2012-03-14T03:...	2012-03-14T07:...	pa:0048	pa:Test	pa:0049	pa:0047	pa:00129
pa:01	2012-03-14T03:...	2012-03-14T07:...	pa:0056	pa:Test		pa:0049	pa:00129

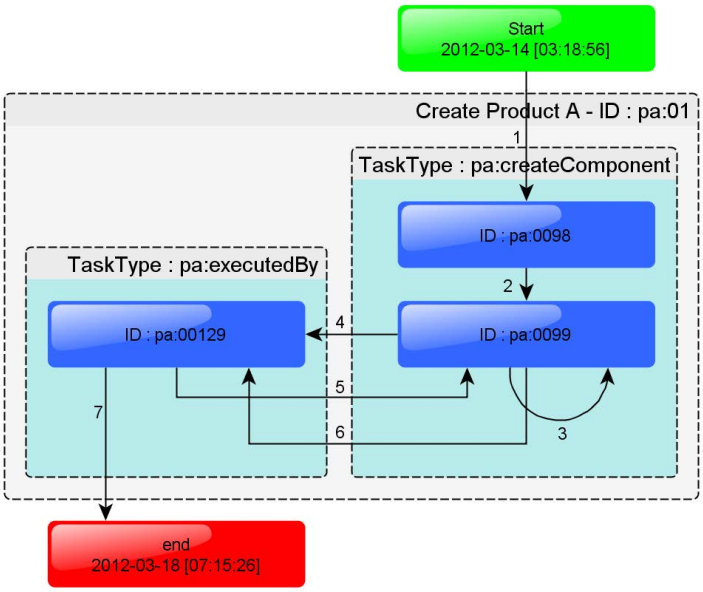

Fig. 10. The process workflow using switched view by used component

this vocabulary is the basic set of information shared between applications that monitor business process execution and submit the triples to the triple store, and the applications that make use of such information.

5 Real Time Aspects

Another important aspect of the business process model presented in this article is the possibility to continuously add information to the graph. This is an important feature of RDF representation, enabled by the fact that every triple is a valid RDF piece of information that identify nodes and connections in the RDf graph. Another important feature of RDF is that both schema and instance-level information is stored in the same graph and that a SPARQL query can return any point in the graph.

Answering queries in SPARQL can be simply translated into identifying all the paths in the graph that satisfy the query pattern. Hence, continuously adding

nodes and connection to the graph generates new paths in the graph that can match queries previously executed.

EBTIC[10] developed a triple store that allows client applications to register queries, so that the client is notified with a new result whenever there is data inserted in the system closing a path that satisfies the registered query. In case the triple store does not support such a feature, a timer can be defined and the queries to monitor can be executed continuously providing the difference between the results at time *t* and *t-1* to the client.

The queries that have been presented in the previous section can be registered in the triple store as continuous queries and the application will be notified with every new result. Assuming that the process monitor will continuously intercept process execution data and translate it into triples, the visualisation application is able to monitor the processes in real-time.

6 Open Problems

One of the major issues that at the moment prevents us from obtaining a fully flexible solution to business process discovery with absolute transparency between process monitors and the graph, is the assumption that all the process monitors operating on the same extended process model (for example the one defined in Section 3.1) need to agree on the process model specific vocabulary and they are not allowed to modify it without mutual agreement with all the other process monitors. This is a requirement we have to impose at the moment, but in future the idea is that every process model is independent and that they can freely modify the process model vocabulary. The RDF graph ideally will automatically adapt and modify the incoming triples in order to adapt to the evolved model. However this is an open problem that will require further research in the field of collaborative ontology evolution.

7 Conclusions

In this article we presented an extremely extensible and flexible data representation model oriented towards real time business process monitoring and discovering; the model is based on Resource Description Framework (RDF) standard that allows independence between applications that generate business process data and applications that consume it. The process is represented as a labelled oriented graph defined by a set of triples as defined in Section 3.

Applications that monitor business processes will extend this basic set of information with their domain specific one at run-time by entering the corresponding triples in the triple store.

We finally demonstrated that this approach allows process discovery and analysis of domain specific extensions that may also be created at run time by third party applications just with the use of SPARQL queries. Future work on this

[10] `http://www.ebtic.org`

direction will be to develop a set of non-invasive monitoring and analytical applications that will allow us to deploy and test this approach within an enterprise-scale environment.

References

1. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering 16(9), 1128–1142 (2004)
2. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow mining: a survey of issues and approaches. Data Knowl. Eng. 47, 237–267 (2003)
3. Taylor, P., Leida, M., Majeed, B.: Case study in process mining in a multinational enterprise. In: Aberer, K., Damiani, E., Dillon, T. (eds.) SIMPDA 2011. LNBIP, vol. 116, pp. 134–153. Springer, Heidelberg (2012)
4. van der Aalst, W.M.P., et al.: Process mining manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM Workshops 2011, Part I. LNBIP, vol. 99, pp. 169–194. Springer, Heidelberg (2012)
5. Hayes, P., McBride, B.: Resource description framework (rdf). Recommendation, W3C (2004)
6. Prud'hommeaux, E., Seaborne, A.: Sparql query language for rdf. Recommendation, W3C (2008)
7. Fischer, M.: Aris process performance manager. In: Bause, F., Buchholz, P. (eds.) MMB, pp. 307–310. VDE Verlag (2008)
8. Weber, I., Hoffmann, J., Mendling, J.: Semantic business process validation. Technical report (2008)
9. Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D.: Wsmo - web service modeling ontology. In: DERI Working Draft 14, vol. 1, pp. 77–106. Digital Enterprise Research Institute (DERI), IOS Press, BG Amsterdam (2005)
10. Azvine, B., Cui, Z., Nauck, D.D., Majeed, B.: Real time business intelligence for the adaptive enterprise. In: 8th IEEE International Conference on E-Commerce Technology - 3rd IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, CEC-EEE 2006. IEEE Computer Society, Washington, DC (2006)
11. Brickley, D., Guha, R., McBride, B.: Rdf vocabulary description language 1.0: Rdf schema. Recommendation, W3C (2004)
12. Carroll, J.J., Bizer, C., Hayes, P., Stickler, P.: Named graphs. Web Semant. 3(4), 247–267 (2005)
13. Rohloff, K., Dean, M., Emmons, I., Ryder, D., Sumner, J.: An evaluation of triple-store technologies for large data stores. In: Meersman, R., Tari, Z. (eds.) OTM-WS 2007, Part II. LNCS, vol. 4806, pp. 1105–1114. Springer, Heidelberg (2007)
14. Shi, H., Maly, K., Zeil, S., Zubair, M.: Comparison of ontology reasoning systems using custom rules. In: Proceedings of the International Conference on Web Intelligence, Mining and Semantics, WIMS 2011, pp. 16:1–16:9. ACM, New York (2011)
15. Bizer, C., Schultz, A.: Berlin sparql benchmark (bsbm). Benchmark, Freie Universität Berlin (2011)
16. Bizer, C., Cyganiak, R., Heath, T.: How to publish Linked Data on the Web (2007)
17. Brandes, U., Eiglsperger, M., Herman, I., Himsolt, M., Marshall, M.S.: GraphML progress report, structural layer proposal. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, pp. 501–512. Springer, Heidelberg (2002)