Enabling the Analysis of Cross-Cutting Aspects in Ad-Hoc Processes

Seyed-Mehdi-Reza Beheshti, Boualem Benatallah, and Hamid Reza Motahari-Nezhad

University of New South Wales, Sydney, Australia
{sbeheshti,boualem,hamidm}@cse.unsw.edu.au

Abstract. Processes in case management applications are flexible, knowledge-intensive and people-driven, and often used as guides for workers in processing of artifacts. An important fact is the evolution of process artifacts over time as they are touched by different people in the context of a knowledge-intensive process. This highlights the need for tracking process artifacts in order to find out their history (artifact versioning) and also provenance (where they come from, and who touched and did what on them). We present a framework, simple abstractions and a language for analyzing cross-cutting aspects (in particular versioning and provenance) over process artifacts. We introduce two concepts of *timed-folders* to represent evolution of artifacts over time, and *activity-paths* to represent the process which led to artifacts. The introduced approaches have been implemented on top of FPSPARQL, Folder-Path enabled extension of SPARQL, and experimentally validated on real-world datasets.

Keywords: Ad-hoc Business Processes, Case Management, Provenance.

1 Introduction

Ad-hoc processes, a special category of processes, have flexible underlying process definition where the control flow between activities cannot be modeled in advance but simply occurs during run time [9]. The semistructured nature of ad-hoc process data requires organizing process entities, people and artifacts, and relationships among them in graphs. The structure of process graphs, describing how the graph is wired, helps in understanding, predicting and optimizing the behavior of dynamic processes. In many cases, however, process artifacts evolve over time, as they pass through the business's operations. Consequently, identifying the interactions among people and artifacts over time becomes challenging and requires analyzing the cross-cutting aspects [12] of process artifacts. In particular, process artifacts, like code, has cross-cutting aspects such as versioning (what are the various versions of an artifact, during its lifecycle, and how they are related) and provenance [7] (what manipulations were performed on the artifact to get it to this point).

The specific notion of business artifact was first introduced in [23] and was further studied, from both practical and theoretical perspectives [17,13,5,8,6]. However, in a dynamic world, as business artifacts changes over time, it is important to

C. Salinesi, M.C. Norrie, and O. Pastor (Eds.): CAiSE 2013, LNCS 7908, pp. 51–67, 2013.
© Springer-Verlag Berlin Heidelberg 2013

be able to get an artifact (and its provenance) at a certain point in time. It is challenging as annotations assigned to an artifact (or its versions) today may no longer be relevant to the future representation of that artifact: artifacts are very likely to have different states over time and the temporal annotations may or may not apply to these evolving states. Consequently, analyzing evolving aspects of artifacts (i.e. versioning and provenance) over time is important and will expose many hidden information among entities in process graphs. This information can be used to detect the actual processing behavior and therefore, to improve the ad-hoc processes.

As an example, knowledge-intensive processes, e.g., those in domains such as healthcare and governance, involve human judgements in the selection of activities that are performed. Activities of knowledge workers in knowledge intensive processes involve directly working on and manipulating artifacts to the extent that these activities can be considered as artifact-centric activities. Such processes, almost always involves the collection and presentation of a diverse set of artifacts, where artifacts are developed and changed gradually over a long period of time. Case management [28], also known as case handling, is a common approach to support knowledge-intensive processes. In order to represent cross-cutting aspects in ad-hoc processes, there is a need to collect meta-data about entities (e.g., artifacts, activities on top of artifacts, and related actors) and relationship among them from various systems/departments over time, where there is no central system to capture such activities at different systems/departments. We assume that process execution data are collected from the source systems and transformed into an event log using existing data integration approaches [3].

In this paper, we present a novel framework for analyzing cross-cutting aspects in ad-hoc processes and show experimentally that our approach addresses the abovementioned challenges and achieves significant results. The unique contributions of the paper are:

- We propose a temporal graph model for representing cross-cutting aspects in ad-hoc processes. This model enables supporting timed queries and weaving cross-cutting aspects, e.g., versioning and provenance, around business artifacts to imbues the artifacts with additional semantics that must be observed in constraint and querying ad-hoc processes. In particular, the model allows: (i) representing artifacts (and their evolution), actors, and interactions between them through activity relationships; (ii) identifying derivation of artifacts over periods of time; and (iii) discovering timeseries of actors and artifacts in process graphs.
- We introduce two concepts of *timed-folders* to represent evolution of artifacts over time, and *activity-paths* to represent the process which led to artifacts.
- We extend FPSPARQL [3], a graph query language for analyzing processes execution, for explorative querying and understanding of cross-cutting aspects in ad-hoc processes. We provide a front-end tool for assisting users to create queries in an easy way and to visualize the proposed graph model and the query results.

The remainder of this paper is organized as follows: We fix some preliminaries in Section 2. Section 3 presents an example scenario in case management

applications. In Section 4 we introduce a data model for representing cross-cutting aspects in ad-hoc processes. In Section 5 we propose a query language for querying the proposed model. In Section 6 we describe the query engine implementation and evaluation experiments. Finally, we discuss related work in Section 7, before concluding the paper in Section 8.

2 Preliminaries

Definition 1. ['Artifact'] An artifact is defined as a digital representation of something that exists separately as a single and complete unit and has a unique identity. An artifact is a *mutable* object, i.e., its attributes (and their values) are able or likely to change over periods of time. An artifact Ar is represented by a set of attributes $\{a_1, a_2, ..., a_k\}$, where k represents the number of attributes.

Definition 2. ['Artifact Version/Instance'] An artifact may appear in many versions. A version v is an *immutable* deep copy of an artifact at a certain point in time. An artifact Ar can be represented by a set of versions $\{v_1, v_2, ..., v_n\}$, where n represents the number of versions. Each version v_i is represented as an artifact instance that exists separately and has a unique identity. Each version v_i consists of a snapshot, a list of its parent versions, and meta-data, such as commit message, author, owner, or time of creation.

Definition 3. ['Activity'] An activity is defined as an action performed on or caused by an artifact version, e.g., an action can be used to create, read, update, or delete an artifact version. We assume that each distinct activity does not have a temporal duration. A timestamp τ can be assigned to an activity.

Definition 4. ['Process'] A process is defined as a group of related activities performed on or caused by artifacts. A starting timestamp τ and a time interval d can be assigned to a process.

Definition 5. ['Actor'] An actor is defined as an entity acting as a catalyst of an activity, e.g., a person or a piece of software that acts for a user or other programs. A process may have more than one actor enabling, facilitating, controlling, and affecting its execution.

Definition 6. ['Artifact Evolution'] In ad-hoc processes, artifacts develop and change gradually over time as they pass through the business's operations. Consequently, *artifact evolution* can be defined as the series of related activities on top of an artifact over different periods of time. These activities can take place in different organizations/departments/systems and various actors may act as the catalyst of activities. Documentation of these activities will generate meta-data about actors, artifacts, and activity relationships among them over time.

Definition 7. ['Provenance'] Provenance refers to the documented history of an *immutable* object which tracks the steps by which the object was derived [7]. This documentation (often represented as graphs) should include all the information necessary to reproduce a certain piece of data or the process that led to that data [22].

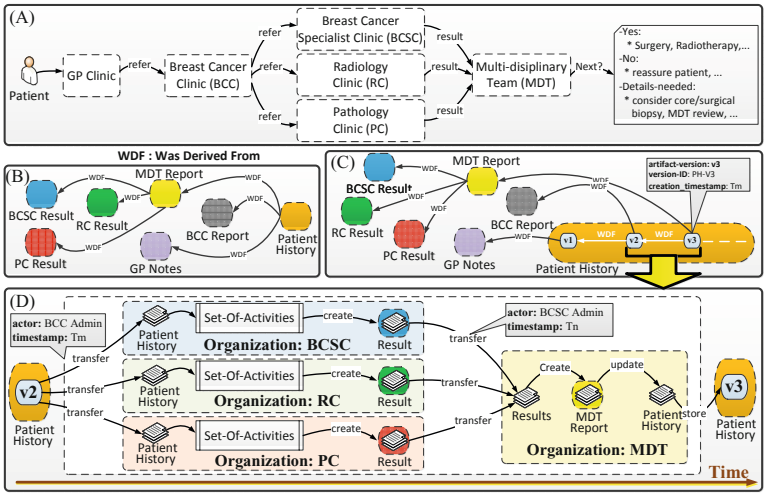

Fig. 1. Example case scenario for breast cancer treatment including a case instance (A), parent artifacts, i.e. ancestors, for patient history document (B) and its versions (C), and a set of activities which shows how version v_2 evolves into version v_3 over time (D).

3 Example Scenario: Case Management

To understand the problem, we present an example scenario in the domain of case management [28]. This scenario is based on breast cancer treatment cases in Velindre hospital [28]. Figure 1-A represents a case instance, in this scenario, where a General Practitioner (GP) suspecting a patient has cancer, updates patient history, and referring the patient to a Breast Cancer Clinic (BCC), where BCC refers the patient to Breast Cancer Specialist Clinic (BCSC), Radiology Clinic (RC), and Pathology Clinic (PC). These departments apply medical examinations and send the results to Multi-Disciplinary Team (MDT). Analyzing the results and the patient history, MDT will decide for next steps. During interaction among different systems and organizations a set of artifacts will be generated. Figure 1-B represents parent artifacts, i.e., ancestors, for patient history document, and Figure 1-C represents parent artifacts for its versions. Figure 1-D represents a set of activities which shows how version v_2 of patient history document develops and changes gradually over time and evolves into version v_3.

4 Representing Cross-Cutting Aspects

Time and Provenance. Provenance refers to the documented history of an *immutable* object and often represented as graphs. The ability to analyze provenance graphs is important as it offers the means to verify data products, to infer their quality, and to decide whether they can be trusted [15]. In a dynamic world, as data changes, it is important to be able to get a piece of data as it was, and its provenance graph, at a certain point in time. Under this perspective, the

provenance queries may provide different results for queries looking at different points in time. Enabling time-aware querying of provenance information is challenging and requires explicitly representing the time information and providing timed abstractions for time-aware querying of provenance graphs.

The existing provenance models, e.g., the open provenance model (OPM) [22], treat time as a second class citizen (i.e., as an optional annotation of the data) which will result in loosing semantics of time and makes querying and analyzing provenance data for a particular point in time inefficient and sometimes inaccessible. For example, the shortest path from a business artifact to its origin may change over time [26] as provenance metadata forms a large, dynamic, and time-evolving graph. In particular, versioning and provenance are important cross-cutting aspects of business artifacts and should be considered in modeling the evolution of artifacts over time.

4.1 AEM Data Model and Timed Abstractions

We propose an artifact-centric activity model for ad-hoc processes to represent the interaction between actors and artifacts over time. This graph data model (i.e., AEM: Artifact Evolution Model) can be used to represent the cross-cutting aspects in ad-hoc processes and to analyze the evolution of artifacts over periods of time. We use and extend the data model proposed in [3] to represent AEM graphs. In particular, AEM data model supports: (i) uniform representation of nodes and edges; (ii) structured and unstructured entities; (iii) *folder* nodes: A folder node contains a set of entities that are related to each other, i.e. the set of entities in a folder node is the result of a given query that requires grouping graph entities in a certain way. A folder can be nested and may have a set of attributes that describes it; and (iv) *path* nodes: A path node represents the results of a query that consists of one or more paths, i.e., a path is a transitive relationship between two entities showing a sequence of edges from the start entity to the end.

In this paper, we introduce two concepts of *timed* folders and *timed* paths, which help in analyzing AEM graphs. Timed folder and path nodes can show their evolution for the time period that they represent. In AEM, we assume that the interaction among actors and artifacts is represented by a directed acyclic graph $G_{(\tau_1,\tau_2)} = (V_{(\tau_1,\tau_2)}, E_{(\tau_1,\tau_2)})$, where $V_{(\tau_1,\tau_2)}$ is a set of nodes representing instances of artifacts in time, and $E_{(\tau_1,\tau_2)}$ is a set of directed edges representing activity relationships among artifacts. It is possible to capture the evolution of AEM graphs $G_{(\tau_1,\tau_2)}$ between timestamps τ_1 and τ_2.

4.2 AEM Entities

An entity is an object that exists independently and has a unique identity. AEM consists of two types of entities:

Artifact Version: Artifacts are represented by a set of instances each for a given point in time. Artifact instances considered as data objects that exist separately and have a unique identity. An artifact instance can be stored as a new

version: different instances of an entity for different points in time, departments, or systems may have different attribute values. An artifact version can be used over time, annotated by activity timestamps $\tau_{activity}$, and considered as a graph node whose identity will be the version unique ID and timestamps $\tau_{activity}$.

Timed Folder Node: We proposed the notion of folder nodes in [3]. A *timed* folder is defined as a timed container for a set of related entities, e.g., to represent artifacts evolution (Definition 6). Timed folders, document the evolution of a folder node by adapting a monitoring code snippet. A time-aware controller is used for creating a snippet and to allocate it to a timed folder node in order to monitor its evolution and update its content (details can be found in [2]). New members can be added to timed folders over time. Entities and relationships in a timed folder node are represented as a subgraph $F_{(\tau_1,\tau_2)} = (V_{(\tau_1,\tau_2)}, E_{(\tau_1,\tau_2)})$, where $V_{(\tau_1,\tau_2)}$ is a set of related nodes representing instances of entities in time added to the folder F between timestamps τ_1 and τ_2, and $E_{(\tau_1,\tau_2)}$ is a set of directed edges representing relationships among these related nodes. It is possible to capture the evolution of the folder $F_{(\tau_1,\tau_2)}$ between timestamps τ_1 and τ_2.

4.3 AEM Relationships

A relationship is a directed link between a pair of entities, which is associated with a predicate defined on the attributes of entities that characterizes the relationship. AEM consists of two types of relationships: activity and activity-path.

Activity Relationships: An activity is an *explicit* relationship that directly links two entities in the AEM graph, is defined as an action performed on or caused by an artifact version, and can be described by following attributes:

- *What* (i.e., type) and *How* (i.e., action), two *types* of activity relationships can be considered in AEM: (i) lifecycle activities, include *actions* such as creation, transformation, use, or deletion of an AEM entity; and (ii) archiving activities, include *actions* such as storage and transfer of an AEM entity;
- *When*, to indicate the timestamp in which the activity has occurred;
- *Who*, to indicate an actor that enables, facilitates, controls, or affects the activity execution;
- *Where*, to indicated the organization/department the activity happened;
- *Which*, to indicate the system which hosts the activity;
- *Why*, to indicate the goal behind the activity, e.g., fulfilment of a specific phase or experiment;

Activity-Path: Defined as an *implicit* relationship that is a container for a set of related activities which are connected through a path, where a path is a transitive relationship between two entities showing the sequence of edges from the starting entity to the end. Relationship can be codified using regular expressions in which alphabets are the nodes and edges from the graph [3]. We define an activity-path for each query which results in a set of paths between two nodes. Activity-paths can be used for efficient graph analysis and can be modeled using timed path nodes.

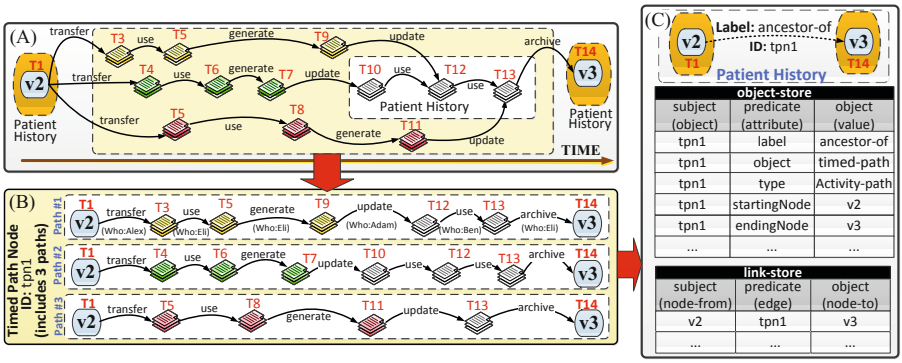

Fig. 2. Implicit/explicit relationships between versions v_2 and v_3 of patient history including: (A) activity edges; (B) activity-path; and (C) their representation/storage

We proposed the notion of path nodes in [3]. A *timed* path node is defined as a timed container for a set of related entities which are connected through *transitive* relationships. We define a timed path node for each change-aware query which results in a set of paths. New paths can be added to timed path nodes over time. Entities and relationships in a timed path node are represented as a subgraph $P_{(\tau_1,\tau_2)} = (V_{(\tau_1,\tau_2)}, E_{(\tau_1,\tau_2)})$, where $V_{(\tau_1,\tau_2)}$ is a set of related nodes representing instances of entities in time which added to the path node P between a time period of τ_1 and τ_2, and $E_{(\tau_1,\tau_2)}$ is a set of directed edges representing *transitive* relationships among these related nodes. It is possible to capture the evolution of the path node $P_{(\tau_1,\tau_2)}$ between a time period of τ_1 and τ_2. Figure 2 represents the implicit and explicit relationships between versions v_2 and v_3 of patient history (a sample folder node) document including: (A) activity edges; (B) constructed activity-path stored as a timed path node; and (C) representation and storage of the activity path. We use triple tables to store objects (object-store) and relationships among them (link-store) in graphs [2].

5 Querying Cross-Cutting Aspects

FPSPARQL [3,4], a Folder-, Path-enabled extension of SPARQL, is a graph query processing engine which supports primitive graph queries and constructing/querying folder and path nodes. In this paper, we extend FPSPARQL to support timed abstractions. We introduce the *discover* statement which enables process analysts to extract information about facts and the relationship among them in an easy way. This statement has the following syntax:

```
discover.[ evolutionOf(artifact1,artifact2) | derivationOf(artifact) |
           timeseriesOf(artifact|actor) ];
filter( what(type),how(action),who(actor),where(location),which(system),when(t1,t2,t3,t4) );
where{  #define variables such as artifact, actor, and location. }
```

This statement can be used for discovering evolution of artifacts (using *evolutionOf* construct), derivation of artifacts (using *derivationOf* construct), and

timeseries of artifacts/actors (using *timeseriesOf* construct). The *filter* statement restrict the result to those activities for which the filter expression evaluates to true. Variables such as artifact (e.g., artifact version), type (e.g., lifecycle or archiving), action (e.g., creation, use, or storage), actor, and location (e.g., organization) will be defined in *where* statement. In order to support temporal aspects of the queries, we adapted the time semantics proposed in [31]. We introduce the special construct, 'timesemantic(fact, [t1, t2, t3, t4])' in FPSPARQL, which is used to represent the $fact$ to be in a specific time interval $[t1, t2, t3, t4]$. A fact may have no temporal duration (e.g., a distinct activity) or may have temporal duration (e.g., series of activities such as process instances). Table 1 represents FPSPARQL time semantics, adapted from [31]. The *when* construct will be automatically translated to *timesemantic* construct in FPSPARQL. Following we will introduce derivation, evolution, and timeseries queries.

5.1 Evolution Queries

In order to query the evolution of an artifact, case analysts should be able to discover activity paths among entities in AEM graphs. In particular, for querying the evolution of an AEM entity En, all activity-paths on top of En ancestors should be discovered. For example, considering the motivating scenario, Adam, a process analyst, is interested to see how version v_3 of patient history evolved from version v_2 (see Figure 2-A). Following is the sample query for this example.

```
1 discover.evolutionOf(?artifact1,?artifact2);
2 where{  ?artifact1 @id v2. ?artifact2 @id v3.
3         ?pathAbstraction @id tpn1.  ?pathAbstraction @label 'ancestor-of'.
4         ?pathAbstraction @description 'version evolution'. }
```

In this example, the *evolutionOf* statement is used to represent the evolution of version v_3 (i.e., variable '?artifact2') from version v_2 (i.e., variable '?artifact1'). The variable '?pathAbstraction' is reserved to identify the attributes for the path node to be constructed. Notice that, by specifying the 'label' attribute (line 3), the implicit relationship, with ID 'tpn1', between versions v_2 and v_3 will be added to the graph. It is possible to query the whole evolution of version v_3 by not considering the first parameter, e.g., in "evolutionOf(,?artifact2)". The attributes of variables '?artifact1' and '?artifact2' can be defined in the *where* clause. As illustrated in Figure 2-A, the result of this query will be a set of paths stored under an activity-path. Please refer to the extended version of the paper [2] to see the FPSPARQL translation of this query.

Table 1. FPSPARQL Time Semantics, adapted from [31]

Time Semantic	Time Range	Time Semantic	Time Range	Time Semantic	Time Range
in, on, at, during	[t,t,t,t]	after	[t,?,?,?]	till, until, by	[?,?,t,t]
since	[t,t,?,?]	before	[?,?,?,t]	between	[t,?,?,t]

5.2 Derivation Queries

In AEM graphs, derivation of an entity En can be defined as all entities which En found to have been derived from them. In particular, if entity En_b is reachable from entity En_a in the graph, we say that En_a is an ancestor of En_b. The result of a derivation query for an AEM entity will be a set of AEM entities, i.e., its ancestors. For example, Adam is interested to find all ancestors of version v_3 of patient history (see Figure 1-C) generated in radiology clinic before March 2011. Following is the sample query for this example.

```
1 discover.derivationOf(?artifact);  filter( where(?location), when(?,?,?,?t1) );
2 where{ ?artifact @id v3. ?location @name 'radiology'. ?t1 @timestamp '3/1/2011 @ 0:0:0'.}
```

In this example, *derivationOf* statement is used to represent the derivation(s) of version v_3 of patient history. Attributes of variable '?artifact' can be defined in the *where* clause. The *filter* statement is used to restrict the result to those activities, happened before March 2011 in radiology clinic. A sample graph result for this query has been depicted in Figure 1-C. Please refer to the extended version of the paper [2] to see the FPSPARQL translation of this query.

5.3 Timeseries Queries

In analyzing AEM graphs, it is important to understand the timeseries, i.e., a sequence of data points spaced at uniform time intervals, of artifacts and actors over periods of time. To achieve this, we introduce *timeseriesOf* statement. The result of artifact/actor timeseries queries will be a set of artifact/actor over time, where each artifact/actor connected through a 'happened-before' edge. For example, Adam is interested in Eli's activities on the patient history document between timestamps τ_1 and τ_{15}. Following is the sample FPSPARQL query for this example.

```
1 discover.timeseriesOf(?actor); filter(when("T1",?,?,"T15")); where{ ?actor @id Eli-id. }
```

In this example, *timeseriesOf* statement is used to represent the timeseries of Eli, i.e., the variable '?actor'. Attributes of variable ?*actor* can be defined in the *where* clause. Considering the path number one in Figure 2-B, where Eli did activities on top of patient history document on τ_5, τ_9, and τ_{14}, Figure 3 represents the timeseries of Eli for the this query. Please refer to the extended version of the paper [2] to see the FPSPARQL translation of this query.

Fig. 3. Eli's Timeseries for acting on patient history between τ_1 and τ_{15}

5.4 Constructing Timed Folders

To construct a *timed folder* node, we use FPSPARQL's *fconstruct* statement proposed in [3]. We extend this statement with '@timed' attribute. Setting the value of attribute *timed* to *true* for the folder, will assign a monitoring code snippet to this folder. The code snippet is responsible for updating the folder content over time: new members can be added to timed folders over time. For example, considering Figure 1-C, a timed folder can be constructed to represent a patient history document. Following is a sample query for this example.

```
1  fconstruct X14-patient-history as ?med-doc select ?version
2  where { ?med-doc @timed true. ?med-doc @type artifact.
3   ?med-doc @description 'history for patient #X14'.
4   ?version @isA entityNode. ?version @patient-ID X14. }
```

In this example, variable '?med-doc' represents the folder node to be constructed (line 1). This folder is of type 'artifact' (line 2). Setting the attribute *timed* to *true* (line 2) will force new artifacts having the patient ID 'X14' (line 4) to be added to this folder over time. The attribute 'description' used to describe the folder (line 34. The variable '?version' is an AEM entity and represents the patient history versions to be collected. Attribute 'patient-ID' (line 4) indicate that the version is related to the patient history of the patient having the id 'X14'. Please refer to the extended version of the paper [2] for more details.

6 Implementation and Experiments

Implementation. The query engine is implemented in Java. Implementation details, including architecture and graphical representation of the query engine can be found in [2]. Moreover, we have implemented a front-end tool to assist process analysts in two steps: (i) Query Assistant: we provided users with a front-end tool (Figure 4-A) to generate AEM queries in an easy way. Users can easily drag entities (i.e., artifacts and actors) in the activity panel. Then they can drag the operations (i.e., evolution, derivation, or timeseries) on top of selected entity. The proposed templates (e.g., for evolution, derivation, and timeseries queries) will be automatically generated; and (ii) Visualizing: we provided users with a timeline like graph visualization (Figure 4-B) with facilities such as zooming in and zooming out.

Experiments. We carried out the experiments on three time-sensitive datasets: (i) The real life log of a Dutch academic hospital[1], originally intended for use in the first Business Process Intelligence Contest (BPIC 2011); (ii) e-Enterprise Course[2], this scenario is built on our experience on managing an online project-based course; and (iii) Supply Chain Management log[3]. Details about this datasets can be found in [2]. The preprocessing of the log is an essential step in gaining

[1] http://data.3tu.nl/repository/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54
[2] http://www.cse.unsw.edu.au/~cs9323
[3] http://www.ws-i.org

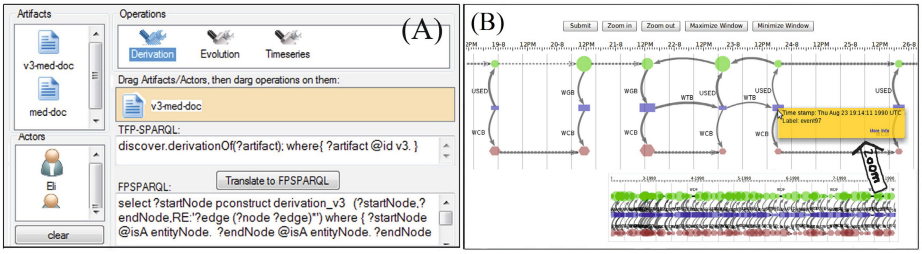

Fig. 4. Screenshots of front end tool: (A) Query assistant tool; and (B) graph visualization tool: to visualize AEM graphs

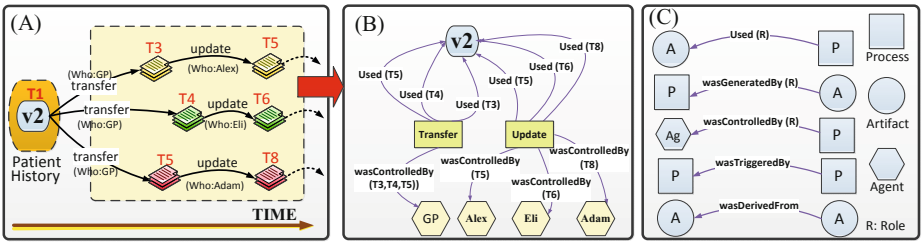

Fig. 5. A sample AEM graph for the hospital log (A), a sample OPM graph generated from a part of AEM graph (B), and open provenance model entities/relationships (C)

meaningful insights and it can be time consuming. For example, the log of a Dutch academic hospital contains 1143 cases and 150291 events referring to 624 distinct activities. We extracted various activity attributes both at the event level and at the case level, e.g., 11 diagnosis code, 16 treatment code, and 16 attributes pertaining to the time perspective. Afterward, we generate the AEM graph model, out of these extracted information. In particular, a system needs to be provenance-aware [7] to automatically collect and maintain the information about versions, artifacts, activities (and its attributes such as type, who, when).

We have compared our approach with that of querying open provenance model (OPM) [22]. We generated two types of graph models, i.e., AEM and OPM, from proposed datasets. The AEM graphs generated based on the proposed model in Section 4.1. The OPM graphs generated based on open provenance model specification [22]. Figure 5, represents a sample AEM graph (Figure 5-A) for the hospital log, a sample OPM graph generated from a part of AEM graph (Figure 5-B), and open provenance model entities and relationships (Figure 5-C). Both AEM and OPM graphs for each datasets loaded into FPSPARQL query engine. We evaluated the performance and the query results quality using the proposed graphs.

Performance. We evaluated the performance of evolution, derivation, and time-series queries using *execution time* metric. To evaluate the performance of queries, we provided 10 evolution queries, 10 derivation queries, and 10 timeseries queries.

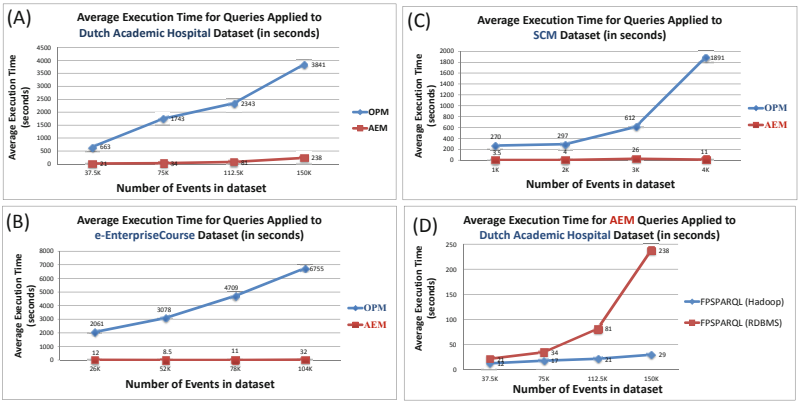

Fig. 6. The query performance evaluation results, illustrating the average execution time for applying evolution, derivation, and timeseries queries on AEM and OPM graphs generated from: (A) Dutch academic hospital dataset; (B) e-Enterprise course dataset; (C) SCM dataset; and (D) the evaluation results, illustrating the performance analysis between RDBMS and Hadoop applied to Dutch academic hospital dataset.

These queries were generated by domain experts who were familiar with the proposed datasets. For each query, we generated an equivalent query to be applied to the AEM graphs as well as the OPM graphs for each dataset. As a result, a set of historical paths for each query were discovered. Figure 6 shows the average execution time for applying these queries to the AEM graph and the equivalent OPM graph generated from each dataset. As illustrated in Figure 6 we divided each dataset into regular number of events, then we generated AEM and OPM graph for different sizes of datasets, and finally we ran the experiment for different sizes of AEM and OPM graphs. We sampled different sizes of the graphs very carefully and based on related cases (patients in the log hospital, projects in the e-Enterprise project, and products in the SCM log) to guarantee the attributes of generated graphs. The evaluation shows the viability and efficiency of our approach.

FPSPARQL queries can be run on two types of storage back-end: RDBMS and Hadoop. We also compare the performance of query plans on relational triple-stores and Hadoop file system. All experiments were conducted on a virtual machine, having 32 cores and 192GB RAM. Figure 6-D illustrates the performance analysis between RDBMS and Hadoop for queries (average execution time) in Figure 6-A applied to Dutch academic hospital dataset. Figure 6-D shows an almost linear scalability between the response time of FPSPARQL queries applied to Hadoop file system and the number of events in the log.

Quality. The quality of results is assessed using classical *precision* metric which is defined as the percentage of discovered results that are actually interesting. In this context, interestingness is a subjective matter in its core, and our approach is to have statistical metrics and thresholds on what is not definitely interesting, and the results are presented to user for subjective assessment of their relevance, depending on what they are looking for. Therefore, for evaluating the

interestingness of the result, we asked domain experts who had the most accurate knowledge about the datasets and the related processes to analyze discovered paths and identify what they considered relevant and interesting. We evaluated the number of discovered paths for all the queries (in performance evaluation) and the number of relevant paths chosen by domain experts. As a result of applying queries to AEM graphs generated from all the datasets, 125 paths were discovered and examined by domain experts, and 122 paths (precision=97.6%) considered relevant. And as a result of applying queries to OPM graphs generated from all the datasets, 297 paths discovered, examined by domain experts, and 108 paths (precision=36.4%) considered relevant.

Discussion/Tradeoffs/Drawbacks. Cross-cutting aspects in ad-hoc processes differs from other forms of meta-data because they are based on the relationships among objects. Specifically for aspects such as provenance and versioning, it is the ancestry relationships that form the heart of ad-hoc processes' data. Therefore, the proposed AEM model considers the issue of paths and cycles among objects in ad-hoc processes' data. Evaluation shows that the path queries applied to the OPM graph resulted in many irrelevant paths and also many cycles discovered in the OPM graph: these cycles hide the distinction between ancestors and descendants. Conversely, few cycles and irrelevant paths have been discovered in the AEM model. Moreover, to increase the performance of path queries in AEM graphs, we implemented an interface to support various graph reachability algorithms such as all-pairs shortest path, transitive closure, GRIPP, tree cover, chain cover, and Sketch [2].

AEM model requires pattern matching over sequences of graph edges as well as pattern matching against the labels on graph edges, where the support for full regular expressions over graph edges is important. Moreover, AEM model requires the uniform representation of nodes and edges, where this representation encodes temporal data into versions while fully retaining the temporal information of the original data. Even though this may seem a bloated representation of the graph, however, this will guarantee the (provenance) graph to be acyclic, but risks leading to large quantities of data. This tradeoff is similar to the tradeoffs for versioning, but it enables users to have reproducible results. In terms of versioning, versions can be created implicitly each time more information is added to an existing artifact.

7 Related Work

We study the related work into three main areas: artifact-centric processes, provenance, and modeling/querying temporal graphs.

Artifact-Centric Processes. Knowledge-intensive processes almost always involve the collection and presentation of a diverse set of artifacts and capturing the human activities around artifacts. This, emphasizes the artifact-centric nature of such processes where *time* becomes an important part of the equation. Many approaches [17,13,5,8,6] used business artifacts that combine data and

process in a holistic manner and as the basic building block. Some of these works [17,13,8] used a variant of finite state machines to specify lifecycles. Some theoretical works [6,5] explored declarative approaches to specifying the artifact lifecycles following an event oriented style. Another line of work in this category, focused on modeling and querying artifact-centric processes [20,30,11]. In [20,30], a document-driven framework, proposed to model business process management systems through monitoring the lifecycle of a document. Dorn et.al. [11], presented a self-learning mechanism for determining document types in people-driven ad-hoc processes through combining process information and document alignment. Unlike our approach, these approaches assumed a predefined document structure or they presume that the execution of the business processes is achieved through a BPM system (e.g., BPEL) or a workflow process.

Another related line of work is artifact-centric workflows [5] where the process model is defined in terms of the lifecycle of the documents. Some other works [25,9,10,27], focused on modeling and querying techniques for knowledge-intensive tasks. Some of existing approaches [25] for modeling ad-hoc processes focused on supporting ad-hoc workflows through user guidance. Some other approaches [9,10,27] focused on intelligent user assistance to guide end users during ad-hoc process execution by giving recommendations on possible next steps. All these approaches focused on user activities and guide users based on analyzing past process executions. Unlike these approaches, in our model (AEM), actors, activities, artifacts, and artifact versions are first class citizens, and the evolution of the activities on artifacts over time is the main focus.

Provenance. Many provenance models have been presented in a number of domains (e.g., databases, scientific workflows and the Semantic Web), motivated by notions such as influence, dependence, and causality. The existing provenance models, e.g., the open provenance model (OPM) [22], treat time as a second class citizen (i.e., as an optional annotation of the data) which will result in loosing semantics of time and makes querying and analyzing provenance data for a particular point in time inefficient and sometimes inaccessible. Discovering historical paths through provenance graphs forms the basis of many provenance query languages [18,15,32]. In ProQL [18], a query takes a provenance graph as an input, matches parts of the input graph according to path expression and returns a set of paths as the result of the query. PQL [15] proposed a semi-structured model for handling provenance and extended the Lorel query language for traversing provenance graph. NetTrails [32] proposed a declarative platform for interactively querying provenance data in a distributed system. In our approach, we introduce an extended provenance graph model to explicitly represent time as an additional dimension of provenance data.

Modeling/Querying Temporal Graphs. In recent years, a plethora of work [16,19,26] has focused on temporal graphs to model evolving, time-varying, and dynamic networks of data. Ren et al. [26] proposed a historical graph-structure to maintain analytical processing on such evolving graphs. Moreover, authors in [19,26] proposed approaches to transform an existing graph into a

similar temporal graph to discover and describe the relationship between the internal object states. In our approach, we propose a temporal artifact evolution model to capture the evolution of time-sensitive data where this data can be modeled as temporal graph. We also provide abstractions and efficient mechanisms for time-aware querying of AEM graphs.

Approaches for querying graphs (e.g., [1,14,24,29]) provide temporal extensions of existing graph models and languages. Tappolet et al. [29] provided temporal semantics for RDF graphs. They proposed τ-SPARQL for querying temporal graphs. Grandi [14] presented another temporal extension for SPARQL, i.e. T-SPARQL, aimed at embedding several features of TSQL2 [21] (temporal extension of SQL). SPARQL-ST [24] and EP-SPARQL [1] are extensions of SPARQL supporting real time detection of temporal complex patterns in stream reasoning. Our work differs from these approaches as we enable registering time-sensitive queries, propose timed abstractions to store the result of such queries, and enable analyzing the evolution of such timed abstractions over time.

8 Conclusion and Future Work

In this paper, we have presented an artifact-centric activity model (AEM) for ad-hoc processes. This model supports timed queries and enables weaving cross-cutting aspects, e.g., versioning and provenance, around business artifacts to imbues the artifacts with additional semantics that must be observed in constraint and querying ad-hoc processes. Two concepts of timed folders and activity-paths have been introduced, which help in analyzing AEM graphs. We have extended FPSPARQL [3,4] to query and analyze AEM graphs. To evaluate the viability and efficiency of the proposed framework, we have compared our approach with that of querying OPM models. As future work, we are weaving the timed abstractions with our work on on-line analytical processing on graphs [4] to support business analytics. Moreover, we plan to employ interactive graph exploration and visualization techniques to design a visual query interface.

References

1. Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: EP-SPARQL: a unified language for event processing and stream reasoning. In: WWW (2011)
2. Beheshti, S.M.R., Benatallah, B., Motahari Nezhad, H.R.: A framework and a language for analyzing cross-cutting aspects in ad-hoc processes. Technical Report UNSW-CSE-TR-201228, University of New South Wales (2012)
3. Beheshti, S.-M.-R., Benatallah, B., Motahari-Nezhad, H.R., Sakr, S.: A query language for analyzing business processes execution. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 281–297. Springer, Heidelberg (2011)
4. Beheshti, S.-M.-R., Benatallah, B., Motahari-Nezhad, H.R., Allahbakhsh, M.: A framework and a language for on-line analytical processing on graphs. In: Wang, X.S., Cruz, I., Delis, A., Huang, G. (eds.) WISE 2012. LNCS, vol. 7651, pp. 213–227. Springer, Heidelberg (2012)

5. Bhattacharya, K., Gerede, C.E., Hull, R., Liu, R., Su, J.: Towards formal analysis of artifact-centric business process models. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 288–304. Springer, Heidelberg (2007)
6. Bhattacharya, K., Hull, R., Su, J.: A data-centric design methodology for business processes. In: Handbook of Research on Business Process Modeling, pp. 503–531 (2009)
7. Cheney, J., Chiticariu, L., Tan, W.C.: Provenance in databases: Why, how, and where. Found. Trends Databases 1, 379–474 (2009)
8. Cohn, D., Hull, R.: Business artifacts: A data-centric approach to modeling business operations and processes. IEEE Data Eng. Bull. 32(3), 3–9 (2009)
9. Dorn, C., Burkhart, T., Werth, D., Dustdar, S.: Self-adjusting recommendations for people-driven ad-hoc processes. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 327–342. Springer, Heidelberg (2010)
10. Dorn, C., Dustdar, S.: Supporting dynamic, people-driven processes through self-learning of message flows. In: Mouratidis, H., Rolland, C. (eds.) CAiSE 2011. LNCS, vol. 6741, pp. 657–671. Springer, Heidelberg (2011)
11. Dorn, C., Marín, C.A., Mehandjiev, N., Dustdar, S.: Self-learning predictor aggregation for the evolution of people-driven ad-hoc processes. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 215–230. Springer, Heidelberg (2011)
12. Dyreson, C.E.: Aspect-oriented relational algebra. In: EDBT, pp. 377–388 (2011)
13. Gerede, C.E., Su, J.: Specification and verification of artifact behaviors in business process models. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 181–192. Springer, Heidelberg (2007)
14. Grandi, F.: T-SPARQL: a TSQL2-like temporal query language for RDF. In: International Workshop on Querying Graph Structured Data, pp. 21–30 (2010)
15. Holland, D.A., Braun, U., Maclean, D., Muniswamy-Reddy, K.K., Seltzer, M.: Choosing a data model and query language for provenance. In: IPAW (2008)
16. Holme, P., Saramäki, J.: Temporal networks. CoRR, abs/1108.1780 (2011)
17. Hull, R.: Artifact-centric business process models: Brief survey of research results and challenges. In: Meersman, R., Tari, Z. (eds.) OTM 2008, Part II. LNCS, vol. 5332, pp. 1152–1163. Springer, Heidelberg (2008)
18. Karvounarakis, G., Ives, Z.G., Tannen, V.: Querying data provenance. In: SIGMOD. ACM (2010)
19. Kostakos, V.: Temporal graph. Physica A: Statistical Mechanics and its Applications 388(6), 1007–1023 (2009)
20. Kuo, J.: A document-driven agent-based approach for business processes management. Information and Software Technology 46(6), 373–382 (2004)
21. Mitsa, T.: Temporal Data Mining, 1st edn. Chapman & Hall/CRC (2010)
22. Moreau, L., Clifford, B., Freire, J., Futrelle, J., Gil, Y., Groth, P.T., Kwasnikowska, N., Miles, S., Missier, P., Myers, J., Plale, B., Simmhan, Y., Stephan, E.G., Van den Bussche, J.: Van den J. Bussche. The open provenance model core specification (v1.1). Future Generation Comp. Syst. 27(6), 743–756 (2011)
23. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Systems Journal 42(3), 428–445 (2003)
24. Perry, M., et al.: SPARQL-ST: Extending SPARQL to support spatiotemporal queries. In: Geospatial Semantics and the Semantic Web, pp. 61–86 (2011)
25. Reijers, H.A., Rigter, J.H.M., Aalst, W.M.P.V.D.: The case handling case. Int. J. Cooperative Inf. Syst. 12(3), 365–391 (2003)
26. Ren, C., Lo, E., Kao, B., Zhu, X., Cheng, R.: On querying historical evolving graph sequences. VLDB 4(11), 727–737 (2011)

27. Schonenberg, H., Weber, B., van Dongen, B.F., van der Aalst, W.M.P.: Supporting flexible processes through recommendations based on history. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 51–66. Springer, Heidelberg (2008)
28. Swenson, K.D., et al.: Taming the Unpredictable Real World Adaptive Case Management: Case Studies and Practical Guidance. Future Strategies Inc. (2011)
29. Tappolet, J., Bernstein, A.: Applied temporal RDF: Efficient temporal querying of RDF data with SPARQL. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E. (eds.) ESWC 2009. LNCS, vol. 5554, pp. 308–322. Springer, Heidelberg (2009)
30. Wang, J., Kumar, A.: A framework for document-driven workflow systems. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 285–301. Springer, Heidelberg (2005)
31. Zhang, Q., Suchanek, F.M., Yue, L., Weikum, G.: TOB: Timely ontologies for business relations. In: WebDB (2008)
32. Zhou, W., et al.: NetTrails: a declarative platform for maintaining and querying provenance in distributed systems. In: SIGMOD, pp. 1323–1326 (2011)