Organizing, Querying, and Analyzing Ad-hoc Processes' Data

Seyed Mehdi Reza Beheshti

A thesis in fulfilment of the requirements for the degree of

Doctor of Philosophy

THE UNIVERSITY OF NEW SOUTH WALES

SYDNEY · AUSTRALIA

School of Computer Science and Engineering

Faculty of Engineering

Supervisor: Prof. Boualem Benatallah

September 2012

Originality Statement

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the projects design and conception or in style, presentation and linguistic expression is acknowledged.

Signed .....

Date 6.Feb.2013

Copyright Statement

Authenticity Statement

I certify that the Library deposit digital copy is a direct equivalent of the final officially approved version of my thesis. No emendation of content has occurred and if there are any minor variations in formatting, they are the result of the conversion to digital format.

Signed ..... SMR Behbahani

Date ....6.Feb.2013..

Acknowledgements

Working at the School of Computer Science and Engineering at the University of New South Wales (UNSW) has been a great pleasure and a wonderful privilege.

In the first place, I would like to express my sincere appreciation and deep gratitude to my supervisor, Professor Boualem Benatallah, for his exceptional support, encouragement and guidance during the last three and a half years. Boualem taught me how to do high quality research and helped me think creatively. His truly incredible academic excellence and beautiful mind have made him as a constant oasis of ideas and passions in science, which has inspired and enriched my growth as a student, a researcher and a scientist. Moreover, I thank him for providing me with the opportunity to work with a talented team of researchers.

I gratefully thank my co-supervisor, Dr. Hamid Reza Motahari-Nezhad, for his outstanding and insightful comments on my work and friendly support in all stages of my study. Hamid is a passionate scientist, an excellent forward thinker, and an oasis of novel ideas, some of which has been inspiring for me in the conception of my scholarly research work.

My sincere thanks go to Dr. Sherif Sakr, my co-author. I thoroughly enjoyed his fruitful collaboration, and I gained invaluable skills by working with him. The joy and enthusiasm Sherif has for his research was contagious and motivational for me.

I am thankful to everyone in the Service-Oriented Computing (SOC) group at UNSW, especially Dr. Adnene Guabtni, Mohammad Allahbakhsh, and Mohammad Reza Nouri for their friendship, support and helpful comments. In addition, I would like to thank the PhD review panels and the anonymous reviewers who provided suggestions and helpful feedback on my publications.

I acknowledge the Australian Government, University of New South Wales, and the Faculty of Engineering at UNSW for providing scholarships (APA, EETS, SEA, and PRSS) to pursue doctoral studies. In addition, I would like to thank administra-

tive and technical staff members of the school of computer science and engineering at UNSW who have been kind enough to advise and help in their respective roles.

I am deeply and forever indebted to my parents and also my parents-in-law for their love and inspiration.

Last, but not least, I would like to dedicate this thesis to my wife Shirin and my son Barbod, for their love, patience, and understanding. They allowed me to spend most of the time on this thesis. They are my source of strength and without their countless support this thesis would have never been started.

Seyed-Mehdi-Reza Beheshti
Sydney, Australia
September 2012

To my wife Shirin and my son Barbod,
for their love, patience, and understanding

Abstract

Business processes are central to the operation of both public and private organizations. Recently, business world is getting increasingly dynamic as various technologies such as social media and Web 2.0 have made dynamic processes more prevalent. For example, outsourcing and the emphasis on customer service makes the use of complex, dynamic and often knowledge intensive activities an inevitable task.

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. In this dissertation, we investigate the problem of explorative querying, and analyzing of ad-hoc processes. Addressing this problem is challenging, as the information about process execution is scattered across several systems and data sources. Moreover, in many cases, there is no well-documented information on how this information is related to each other and to the overall business process of the enterprise. Enabling above-mentioned analysis requires a model and a query language for representing and querying process entities (e.g., events, artifacts, and actors), relationships among them, and the evolution of business artifacts over time. Moreover, the model should support multi-dimensional/-level views and analytics over ad-hoc processes data.

To address these challenges, we present a framework, simple abstractions and a language for the explorative querying and understanding of ad-hoc processes data from various user perspectives. We propose novel abstractions, *folder* and *path*, for facilitating the analysis of ad-hoc processes data by enabling process analysts to group related entities or find patterns among entities. We present FPSPARQL (Folder-, Path-enabled SPARQL) as a language and a set of new methods for organizing, indexing and querying ad-hoc processes. We then extend FPSPARQL for analyzing the evolution of process artifact, and for analyzing cross-cutting aspects in ad-hoc processes. We introduce two concepts of *timed-folder* to represent evolution of artifacts over time, and *activity-path* to represent the process which led to

artifacts. Finally, we propose a model, GOLAP, and extend FPSPARQL for online analytical processing on process graphs. The approaches presented in this dissertation have been implemented in prototype tools, and experimentally validated on synthetic and real-world datasets.

Dissertation Examiners:

- Prof. Schahram Dustdar, Vienna University of Technology, Austria
- Prof. Farouk Toumani, Blaise Pascale University, France

Publications

- Beheshti S.M.R., Benatallah B., Motahari-Nezhad H.R., and Sakr S., "A query language for analyzing business processes execution", 9th International Conference on Business Process Management (BPM), pages 281-297, Clermont-Ferrand, France, 2011. (ERA Rank: A)

- Beheshti S.M.R., Benatallah B., Motahari-Nezhad H.R., and Allahbakhsh M., "A Framework and a Language for On-Line Analytical Processing on Graphs", 13th International Conference on Web Information System Engineering (WISE), pages 213-227, Paphos, Cyprus, 2012. (ERA Rank: A)

- Beheshti S.M.R., Benatallah B., and Motahari-Nezhad H.R., "Enabling the Analysis of Cross-Cutting Aspects in Ad-hoc Processes", 25th International Conference on Advanced Information Systems Engineering (CAiSE), Valencia, Spain, 2013. (ERA Rank: A)

- Beheshti S.M.R., Benatallah B., Motahari-Nezhad H.R., and Lagares Lemos A., "FPSPARQL: A Query Engine for Explorative Querying and Analyzing Information Networks", 22th International World Wide Web Conference (WWW), Rio de Janeiro, Brazil, 2013. -submitted- (ERA Rank: A)

- Beheshti S.M.R., Benatallah B., and Motahari-Nezhad H.R., "FPSPARQL: A Graph Query Engine as a Service", Second Australasian Symposium on Service Research and Innovation, Sydney, Australia, 2012.

- Beheshti S.M.R., Motahari-Nezhad H.R., Benatallah B.: Temporal Provenance Model (TPM): Model and Query Language. CoRR abs/1211.5009 (2012).

- Beheshti S.M.R., Sakr S., Benatallah B., Motahari-Nezhad H.R.: Extending SPARQL to Support Entity Grouping and Path Queries. CoRR abs/1211.5817 (2012).

- Beheshti S.M.R., Benatallah B., Motahari-Nezhad H.R., "A Framework and a Language for Analyzing Cross-Cutting Aspects in Ad-hoc Processes", unsw-cse-tr-201228, University of New South Wales, 2012.

- Beheshti S.M.R., Benatallah B., Motahari-Nezhad H.R., and Allahbakhsh M., "Online Analytical Processing on Graphs (GOLAP): Model and Query Language", unsw-cse-tr-201214, University of New South Wales, 2012.

- Beheshti S.M.R., Benatallah B., and Motahari-Nezhad H.R., "An Artifact-Centric Activity Model for Analyzing Knowledge Intensive Processes", unsw-cse-tr-201210, University of New South Wales, 2012.

- Beheshti S.M.R., Benatallah B., Motahari-Nezhad H.R., and Sakr S., "FPSPARQL: A Language for Querying Semi-Structured Business Process Execution Data", unsw-cse-tr-1103, University of New South Wales, 2011.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Business processes are central to the operation of both public and private organizations. A business process consists of a set of coordinated tasks and activities employed to achieve a business objective or goal. Recently, business world is getting increasingly dynamic, as various technologies such as Internet and email have made dynamic processes more prevalent. Moreover, outsourcing[1] and the emphasis on customer service makes the use of complex, dynamic and often knowledge intensive activities an inevitable task. Consequently, in modern enterprises, BPs are realized over a mix of workflows, IT systems, Web services and direct collaborations of people to support such ad-hoc and dynamic activities.

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 [126, 121, 307]. In such cases, the process execution path can change in a dynamic and ad-hoc manner due to changing business requirements, dynamic customer needs, and people's growing skills. Examples of this, are the processes in the area of government, law enforcement, financial services, and telecommunications. Conventional workflows do not provide sufficient flexibility to reflect the nature of such processes. For example, working with ad-hoc processes allows organizations to be very flexible, which is a prerequisite for a

[1]Outsourcing is the process of contracting an existing business process which an organization previously performed internally to an independent organization, where the process is purchased as a service.

competitive process performance when working with varying business partners.

Analyzing and understanding such dynamic processes offers important information for the organization's management. This information can be used to detect the actual processing behavior and therefore, to improve the ad-hoc processes. However, understanding of business processes and analyzing BP execution data (e.g., logs containing events, interaction messages and other process artifacts) is difficult as the information about process execution is scattered across several systems and data sources. Moreover, in many cases, there is no well-documented information on how this information is related to each other and to the overall business process of the enterprise [252].

The main barrier for understanding ad-hoc processes is to identify the interactions among entities (e.g., process stakeholders and process artifacts) within BP execution data. In this context, most entities (structured or unstructured) in process logs are interconnected through rich semantic information, where entities and relationships among them can be modeled using graphs. Knowledge about business processes is often hidden in the relationships among entities in process graphs, i.e., BP execution data modeled using graphs. Understanding this hidden knowledge in terms of its scope and details is challenging specially as it is subjective: depend on the perspective of the process analyst.

In this dissertation, we are interested in facilitating the analysis of process graphs by proposing a framework, simple abstractions and a language for the explorative querying and understanding of BP execution from various user perspectives. We provide facilities for analyzing cross-cutting aspects, e.g., versioning and provenance[2], and supporting timed queries in ad-hoc processes. Furthermore, we extend the proposed framework and query language to support analytics over ad-hoc process data.

The rest of this chapter is organized as follows. We first introduce concepts central to the work described in this dissertation. This is followed by a description of the key research issues tackled in this dissertation. Next, we summarize our contributions to the area. Finally, we describe the organization of this dissertation.

[2]Provenance refers to the documented history of an object (e.g. documents, data, and resources) or the documentation of processes in an object's lifecycle [93, 54].

1.1 Preliminaries

This section gives a brief introduction to the main topics of this dissertation, namely business processes and ad-hoc business processes.

1.1.1 Business Processes

A business process is a set of coordinated tasks and activities[3], carried out manually or automatically, to achieve a business objective or goal [141, 225, 10, 8, 252]. A business process is typically associated with a data flow, showing what data and how they are transferred among activities and tasks, and a control flow, showing the order in which activities and tasks are performed. Two types of business processes are recognized: private and public. Public business processes, can be shared with business partners (e.g., clients and suppliers) within an enterprise and can be used in the business-to-business integration (B2Bi) context [75]. On the contrary, private business processes, are internal to the enterprise, include execution details, and can be used in enterprise application integration (EAI) context [28].

In order to manage organization performance through BPs, set of methods, techniques, and tools, known as Business Process Management (BPM), are needed. In particular, BPM supports the design, enactment, management, and analysis of operational BPs, i.e., processes at the strategic level or processes that cannot be made explicit are excluded [8]. An example of operational BPs is the product shipping process. BPM enables organizations to be more efficient and capable of change, both in human and technological aspects, throughout a lifecycle.

The BPM lifecycle can be divided into four phases [8]: (i) design: in this phase, the process is (re-)designed and modeled; (ii) configuration: during this phase, a process-aware system, e.g., a workflow management system, is configured; (iii) enactment: in the process enactment phase the operational business process is executed; and (iv) diagnosis: in the diagnoses phase the process is monitored, analyzed, and process improvement approaches are proposed. Workflows and workflow man-

[3] An activity is the smallest unit of work, performed by executing a program, enacting a human or machine action or invoking another business process (known as sub-process) [10, 252].

agement systems (WfMS) were introduced to support BPs lifecycle.

The Workflow Management Coalition [96, 183] defines a workflow as "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules" and defines a Workflow Management System (WfMS) as "a system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants, and where required, invoke the use of IT tools and applications".

From the definition it can be seen that a WfMS only supports the BPM lifecycle from the process design to the process enactment phases. To address this shortcoming, Business Process Management Systems (BPMS) are introduced as an extension of classical WfMS focusing more on the diagnosis phase of the BPM lifecycle, i.e., monitoring, tracking, analysis and predication of business processes. In particular, BPMS can be defined as a generic software system that is driven by explicit process designs to enact and manage operational business processes [10].

Recently, many information systems in the enterprise have been implemented using Web services. Web service technology has become the preferred implementation technology for realizing the Service Oriented Architecture (SOA) paradigm [28, 268]. SOA is an architectural style that provides guidelines on how services are described, discovered and used. In SOA, software applications are packaged as "services", where services are defined to be standards-based, platform- and protocol-independent to address interactions in heterogeneous environments [28, 268].

In such information systems, business process analysis (BPA) over a wide range of information systems, services and softwares that implement the actual business processes of enterprises is required. In particular, there is a significant demand for approaches in an emerging area of BPA, called Business activity monitoring (BAM) [83, 140, 10]. BAM intends to provide real-time business performance indicators to improve the speed and effectiveness of business operations through discovering business process models from the process logs, where tracking and analyzing of process execution will be needed.

Designing and maintaining BAM applications is challenging, as business processes in modern enterprises are developed by different communities of practice, reside on different levels of abstractions, and handled by different IT systems [193, 137]. In particular, modern business processes, have flexible underlying process definition where the process execution path can change in a dynamic and ad-hoc manner. Next section gives an overview of such processes.

1.1.2 Ad-hoc Business Processes

Business world is getting increasingly dynamic. Various technologies such as Internet and email have made dynamic processes more prevalent. Outsourcing and the emphasis on customer service requires companies to continuously adapt their Process-Aware Information Systems (PAIS) [1, 125] in order to cope with the frequent and unprecedented changes in their business environment. Moreover, the phenomenon of performing online collaborative tasks has been recognized as a promising trend which is expected to grow due to the wide acceptance of Web 2.0 technologies and social systems [298, 313, 236]. For instance, in Gartner's top five BPM predictions for 2010, it has been identified as an overriding trend that business process management will be extended to include the management of unstructured work and data [192].

In practice, most of the business processes are conducted in an ad-hoc manner and do not rely on integrated software solutions. Instead, a mix of computerized systems and direct collaborations are often used. For instance, the processing of orders in a company may involve the accounting system, the stock database and personal productivity tools of the salesperson to follow up their customer and decide on discounts. Hence, the data relevant to a business process is then scattered across multiple systems with no integration between them, or they are spread across multiple documents stored in the personal folders of employees and exchanged by communication tools such as emails.

These challenges make the use of complex, dynamic and often knowledge intensive activities an inevitable task [170, 125]. In particular, such processes have flex-

ible underlying process definition where the control flow between activities cannot be modeled in advance but simply occurs during run time [126, 121, 307]. Consequently, the process execution path can change in a dynamic and ad-hoc manner due to changing business requirements, dynamic customer needs, and people's growing skills. In this dissertation, we use the term *ad-hoc* to refer to this category of processes.

Existing business process management tools (WfMSs and BPMSs introduced in Section 1.1.1) support well-structured [283] processes and do not provide sufficient flexibility to reflect the nature of ad-hoc processes, i.e., structured processes are fully prescribed how a future decision will be made. In particular, ad-hoc processes can be divided into two types: unstructured [307] and semi-structured [284]. An unstructured process is a process that can not be reduced to well-defined rules, unlike well-structured processes. A semi-structured process, or case-based processes, is a process which contains both structured and unstructured sub-processes.

Ad-hoc processes are complex not only because they are scattered across several systems and organizations, but also they require many different people (having lots of knowledge and experience) to collaborate to find the correct solution. Various technologies, e.g., customer relationship management (CRM) and content management systems (CMS) have been proposed for managing such dynamic and ad-hoc processes, however, they are not sufficient to address the key requirements of these types of processes: they are primarily driven by human participants reacting to changing context and do not follow a predetermined path. In particular, in ad-hoc processes, one process model would not serve the analysis purpose to understand the next step.

In order to analyze business process execution, process mining [2, 8, 68, 329] and querying [33, 47, 130, 134, 293] techniques received continuous attention in recent years, where most of these techniques relies on two main assumptions: (i) the availability of business processes specification that describe their operational logic, i.e., the execution flow; and (ii) the availability of business processes execution traces recorded in a standard formats. However, these two assumptions are not always valid [192].

The main barrier for understanding ad-hoc processes is to identify the interactions among entities (e.g., process stakeholders and process artifacts) within BP execution data. In this context, most entities (structured or unstructured) in process logs are interconnected through rich semantic information, where entities and relationships among them can be modeled using graphs. Knowledge about business processes is often hidden in the relationships among entities in process graphs, i.e., BP execution data modeled using graphs. In next section, we discuss key research issues in understanding ad-hoc processes.

1.2 Key Research Issues

In this section, we outline key research issues tackled in this dissertation. We intend to facilitate the analysis of process graphs for process analysts. We therefore separate research issues into three areas: understanding ad-hoc process data, cross-cutting aspects in ad-hoc processes, and business process analytics.

1.2.1 Understanding Ad-hoc Process Data

Business processes in modern enterprises are implemented over several applications and Web services, and the information about process execution is scattered across several data sources. Consequently, understanding of ad-hoc processes and analyzing BP execution data will be difficult due to the lack of documentation and especially as the process scope and how process events across these systems are correlated into process instances are subjective: depend on the perspective of the process analyst.

As an example, one analyst may want to understand the delays to the ordering process (the end-to-end from ordering to the delivery) for a specific customer, while another analyst is only considered with the packaging process for any orders in the shipping department. Certainly, one process model would not serve the analysis purpose for both situations. Rather there is a need for a process-aware querying approach that enables analysts to analyze the process events from their perspectives, for the specific goal that they have in mind, and in an explorative manner.

1.2.2 Cross-cutting Aspects in Ad-hoc Processes

Ad-hoc processes have flexible underlying process definition. The semi-structured 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 becomes challenging and requires analyzing the cross-cutting [204] aspects of process artifacts: we apply the aspect-oriented programming (AOP) [204] paradigm to the process artifacts in BP execution data, where AOP is used to add support for cross-cutting aspects to existing code without directly modifying that code [127]. In particular, process artifacts, like code, has cross-cutting aspects such as versioning and provenance. Analyzing these aspects will expose many hidden interactions among entities in process graphs.

For example, consider knowledge-intensive processes, e.g., those in domains such as healthcare and governance, which involve human judgements in the selection of activities that are performed. Such activities, 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. In this context understanding the evolution of artifacts over periods of time needs analyzing cross-cutting aspects of artifacts and supporting timed queries, in ad-hoc processes.

1.2.3 Business Process Analytics

In modern enterprises, businesses accumulate massive amounts of data from a variety of sources. Analytics, i.e., the discovery and communication of meaningful patterns in data, can help in understanding the business data with an eye to predicting and improving business performance in the future. In particular, business process analytics can facilitate the analysis of process graphs in a detailed and intelligent way through describing the applications of analysis, data, and systematic reasoning [74, 171]. Consequently, an analyst can gather more complete insights using techniques

such as modeling, summarizing, and filtering.

While existing analytics solutions, e.g., traditional on-line analytical processing (OLAP) techniques and tools, do a great job in collecting data and providing answers on known questions, key business insights remain hidden in the interactions among objects and data: most objects and data in the process graphs are interconnected, forming complex, heterogeneous but often semi-structured networks. In particular, traditional OLAP technologies were conceived to support multidimensional analysis, however, they cannot recognize patterns among process graph entities and analyzing multidimensional process graph data (from multiple perspectives and granularities) may become complex and cumbersome.

1.3 Contributions Overview

Our goal is to facilitate and simplify the analysis of process graphs. To achieve this goal, we propose a framework, simple abstractions and a language for: (i) organizing, indexing, and querying ad-hoc process data; (ii) representing cross-cutting aspects in ad-hoc processes; and (iii) supporting analytics over ad-hoc process data.

1.3.1 Organizing, Indexing, and Querying Ad-hoc Process Data

The first step of process analysis is gathering and integration of process execution data in a *process event log* from various, potentially heterogeneous, systems and services. We assume that execution data are collected from the source systems and transformed into an event log using existing data integration approaches [280], and we can access the event metadata and the payload content of events in the integrated process log. The next step is providing techniques to enable users define the relationships between process events. In particular, most entities (structured or unstructured) in process logs are interconnected through rich semantic information, where entities and relationships among them can be modeled using graphs, i.e., process graphs.

We present a framework, simple abstractions and a language for the explorative querying and understanding of process graphs from various user perspectives. We propose a query language, FPSPARQL [53, 52, 51] (a Folder-, Path-enabled extension of SPARQL [276]), for facilitating the analysis of process graphs based on the two concepts of folders and paths, which enable a process analyst to group related entities (e.g., process artifacts, events, and actors) in the process graph or find patterns among entities. Folders and paths can be stored to be used in follow-on analysis.

1.3.2 Representing Cross-cutting Aspects in Ad-hoc Processes

To represent cross-cutting aspects in ad-hoc processes, the focus should be on interactions among actors (i.e., people and services) and artifacts over time, where there is no central system to capture such activities at different systems/departments. This is challenging, as artifacts can be accessed/modified by different actors over time, various versions of artifacts can be generated in different sysems/departments, and each artifact version can be derived from various sources.

To address this challenge, we represent versioning and provenance as important cross-cutting aspects of business artifacts: analyzing these aspects will help in understanding ad-hoc processes. We propose a temporal graph model for supporting timed queries and representing the cross-cutting aspects of business artifacts. This model [51] 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. Moreover, 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. Finally, we extend FPSPARQL for querying evolution, derivation, and timeseries of artifacts.

1.3.3 Analytics Over Ad-hoc Process Data

To support analytics over ad-hoc process data, there are two major challenges to be addressed: (i) how to extend decision support on process graphs considering both data objects and the relationships among them: traditional OLAP technologies cannot recognize patterns among graph entities and, consequently, enabling users to analyze multidimensional graph data may become complex and cumbersome; and (ii) in process graphs, providing multiple views at different granularities is subjective: depends on the perspective of process analysts how to partition graphs and apply further operations on top of them.

To address these challenges, we propose a graph data model, GOLAP [52], for online analytical processing on process graphs. This data model enables extending decision support on multidimensional networks considering both data objects and the relationships among them as first class entities. We use the notions of folder and path nodes to support multi-dimensional and multi-level views over process graphs. We redefine OLAP data elements (e.g., cubes, dimensions, and measures) by considering the relationships among process graph entities as first class objects. Moreover, we extend FPSPARQL to support n-dimensional computations on process graphs.

1.3.4 Software Prototype

To address the above challenges, we have developed a software prototype for organizing, indexing, and querying ad-hoc process data. As mentioned earlier, we model process logs as a graph. In order to query this graph a graph query language is needed. Among languages for querying graphs, SPARQL [276] is an official W3C standard and based on a powerful graph matching mechanism. However, SPARQL does not support the construction and retrieval of subgraphs. Also paths are not first class objects in SPARQL [276, 181]. In order to analyze BPs event logs, we propose a graph processing engine, i.e. FPSPARQL [53, 52, 51] (a Folder-, Path-enabled extension of the SPARQL), to manipulate and query entities, and folder and path nodes. We support two levels of queries: (i) Entity-level Queries: at this level we use SPARQL to query entities in the process logs; and (ii) Aggregation-level

Queries: at this level we use FPSPARQL to construct and query folder and path nodes. The query engine is implemented in Java and supports two types of storage back-end:

- Relational Database System: As FPSPARQL core, we implemented a SPARQL-to-SQL translation algorithm based on the proposed relational algebra for SPARQL [103] and semantics preserving SPARQL-to-SQL query translation [91]. This algorithm supports *aggregate* and *keyword search* queries. We implemented the proposed techniques on top of this SPARQL engine. We developed four optimization techniques proposed in [90, 292, 91] to increase the performance of the query engine.

- Hadoop File System: we use Hadoop[4] data processing platforms to store and retrieve process graphs in Hadoop file system and to support cost-effective and Web-scale processing of process graphs. We use Apache-Pig[5], a high-level procedural language on top of Hadoop and the MapReduce[6] platform, for querying large process graph stored in Hadoop file system. As FPSPARQL core, we implemented a FPSPARQL-to-PigLatin translation algorithm based on the intermediate algebra for optimizing RDF [235] (Resource Description Framework) graph pattern matching on MapReduce proposed in [281, 206]. We use existing query optimization techniques [206, 188, 281] to generate the optimal query plan by reinterpreting certain join tree structures as grouping operations, i.e., to enable a greater degree of parallelism in join processing.

We have implemented a front-end tool for assisting users to create FPSPARQL queries in an easy way. In particular, FPSPARQL supports primitive graph queries, constructing folder/path nodes, applying further queries to constructed folder/path nodes, applying external tools and algorithms to graph, and supporting n-dimensional computations on graphs. The software prototype can be used to assist users in four

[4]Apache Hadoop [330] is an open source software framework that supports data-intensive distributed applications.

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

[6]MapReduce [106] is a programming model for processing large data sets, where a popular free implementation is Apache Hadoop.

steps: (i) preprocessing: generating graph models from process logs and providing indexing mechanisms; (ii) partitioning: organizing process graphs using folders and paths; (iii) analyzing: applying further operations to constructed partitions; and (iv) visualizing: supporting the exploration of folders, paths, and the result of queries.

1.4 Dissertation Organization

The remainder of this dissertation is organized as follows. We start with a discussion of the current state of the art in analyzing ad-hoc processes data in Chapter 2. We explain in more depth what business processes and ad-hoc business processes are, as well as different abstractions involved in analyzing process graphs, e.g., data space, data services, and business analytics, followed by a survey of existing work in these domains. We identify what we believe are important research issues in analyzing ad-hoc business processes.

After that, we present the details of our framework for analyzing business processes execution in Chapter 3. In this chapter, we first present a case study on process event logs. Then we present the proposed graph data model and the FPSPARQL query language followed by a detailed discussion on how we use the query language for analyzing the case study process log. Afterward we describe the query engine implementation and evaluation experiments. Finally, we discuss related work in domains such as NoSQL databases, RDF/SPARQL, querying process models and instances, and enterprise search before concluding the chapter.

Chapter 4 discusses how our model facilitates the representation and analyzing of cross-cutting aspects in ad-hoc processes. In order to understand cross-cutting aspects, we present an example scenario in case management applications. Then we represent cross-cutting aspects, versioning and provenance, and timed queries followed by a detailed discussion on how we extend our data model using timed abstractions. After that we present FPSPARQL extensions for querying cross-cutting aspects in ad-hoc processes. Next we describe the query engine implementation, for temporal extensions, and evaluation experiments. Finally, we discuss related work

in domains such as provenance and modeling/querying temporal graphs before concluding the chapter.

In Chapter 5, we present how our framework and query language can be used for on-line analytical processing on process graphs. After presenting an example scenario, in best practice processes applications, we propose a graph data model for online analytical processing on process graphs. We discuss how this data model enables extending decision support on multidimensional networks considering both data objects and the relationships among them. Then we redefine OLAP data elements (e.g., dimensions, measures, and cubes) by considering the relationships among process graph entities as first class objects. After that we present FPSPARQL extensions for supporting n-dimensional computations on process graphs. Next we describe the query engine implementation, for analytic extensions, and evaluation experiments. Finally, we discuss related work in domains such as on-line analytical processing (OLAP), on-line analytical processing on graphs, and analytics over ad-hoc process data before concluding the chapter.

Finally, in Chapter 6, we give concluding remarks of this dissertation and discuss possible directions for future work.

Chapter 2

Background and State-of-the-Art

In this dissertation, we investigate the problem of explorative querying and understanding of ad-hoc business processes execution. This chapter therefore presents central concepts and the current state-of-the-art in business processes, data services, and querying business processes. In business processes (Section 2.1), we discuss related work from structured to unstructured processes. We observe that in process-aware information systems processes are implemented over several information systems in an unstructured manner, and there is a need for a querying mechanism that enables analysts to analyze the process events from their perspectives for the specific goal that they have in mind and in an explorative manner.

In Section 2.2, we discuss data services as they provide rich metadata, expressive languages, and APIs to simplify accessing, querying, and analyzing of information over the Web and can be leveraged for organizing and managing ad-hoc process data. We discuss related work in querying business processes (Sections 2.3 and 2.4) in three categories: querying business process models, querying running instances of business processes, and querying execution logs of completed business processes (also known as *process mining*). We argue that our approach is complementary to process mining techniques as we enable grouping entities (e.g., events, artifacts, or people) in the process log that are then can be queried or considered as an input for process mining algorithms, providing an explorative and interactive manner. Moreover, we enable analyzing cross-cutting aspects of business artifacts, e.g., versioning and provenance,

and provide a framework for applying analytics to ad-hoc processes' data.

We present our observations (Section 2.5) in terms of structured and ad-hoc processes including understanding, correlating, querying, and exploring process execution data in an interactive manner. Finally, we summarize the chapter (Section 2.6) and highlight the remaining problems and gaps, and the need for querying mechanisms for understanding processes in environments that do not support the processes in a structured manner or with ad-hoc changes to the processes and/or with flexible processes.

2.1 Business Processes

A business process (BP) is a set of coordinated tasks and activities, carried out manually or automatically, to achieve a business objective or goal [141, 225, 10, 8, 252]. An activity is the smallest unit of work, performed by executing a program, enacting a human or machine action or invoking another business process (known as sub-process) [10, 252]. A business process is typically *structured* and, therefore, is associated with a data flow, showing what data and how they are transferred among activities and tasks, and a control flow, showing the order in which activities and tasks are performed.

Two types of business processes are recognized: public and private. Public business processes, can be shared with business partners (e.g., clients and suppliers) within an enterprise and can be used in the business-to-business integration (B2Bi) context [75]. On the contrary, private business processes, are internal to the enterprise, include execution details, and can be used in enterprise application integration (EAI) context [28]. In order to manage organization performance through BPs, set of methods, techniques, and tools, known as Business Process Management (BPM), are needed.

Business Processes Management (BPM) is a "generic software system that is driven by explicit process designs to enact and manage operational business processes" [10], where operational processes refer to repetitive business processes per-

formed by organizations in their daily operations, which are explicitly defined and modeled, e.g., the product shipping process. In this context, processes at the strategic decision-making level, which are performed by the enterprise management, or processes that are not explicit are excluded [252, 8]. For instance, if part of the business process is performed by some legacy systems or services that their processes are not explicitly defined, they are not covered.

Such systems should be process-aware and generic in the sense that it is possible to modify the processes they support. This challenge requires organizations to continuously adapt their Process-Aware Information Systems (PAIS) [1] in order to cope with the frequent and unprecedented changes in their business environment. This will enable end-users to be able to issue queries to monitor, analyze and understand the execution information of business processes and related business artifacts. In particular, BPM enables organizations to be more efficient and capable of change, both in human and technological aspects, throughout a *lifecycle*.

The BPM lifecycle can be divided into four phases [8]: (i) design: in this phase, the process is (re-)designed and modeled, where the designs are often graphical and the focus is on 'structured processes' that need to handle many cases [10]; (ii) configuration: during this phase, a process-aware system, e.g., a workflow management system, is configured; (iii) enactment: in the process enactment phase the operational business process is executed; and (iv) diagnosis: in the diagnoses phase the process is monitored, analyzed, and process improvement approaches are proposed.

In the design phase, a modeling tool or technique [302, 282, 189] (e.g., BPMN, CogNIAM, xBML, and EPC) will be used to specify the order of tasks in the business process. Process modeling tools typically support a graph-based modeling approach adopting a popular process modeling notation such as the Business Process Modeling Notation (BPMN) [155]. Process models created in the process design phase are usually too high level to be executed, and therefore, an analytical process model must be configured to an executable process model. Moreover, BPM suite software provides programming interfaces (e.g., BPEL, WS-CDL, and XPDL or other technologies including model-driven/service-oriented architecture) which allow enterprise applications to be built to leverage the BPM engine. Workflows and

workflow management systems (WfMS) were introduced to support BPs lifecycle.

The Workflow Management Coalition [96, 183] defines a workflow as "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules" and defines a Workflow Management System (WfMS) as "a system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants, and where required, invoke the use of IT tools and applications".

From the definition it can be seen that a WfMS only supports the BPM lifecycle from the process design to the process enactment phases. To address this shortcoming, Business Process Management Systems (BPMS) are introduced as an extension of classical WfMS focusing more on the diagnosis phase of the BPM lifecycle, i.e., monitoring, tracking, analysis and predication of business processes. In a BPMS, various resource classes, e.g., human or non-human, application or non-application, and individual or teamwork could contribute to tasks within a business process. Most of BPMSs focus on task-resource allocation in terms of individual human resources only [265, 282, 63], i.e., they model control flow dependencies between activities in structured business processes.

Currently, many workflow vendors are positioning their systems as BPM systems. In particular, workflow management systems [9, 302, 325] (e.g., Staffware, MQSeries, and COSA) can be used to integrate existing applications and support process changes by merely changing the workflow diagram [9, 8, 283]. Isolating the management of business processes in a separate component is also consistent with recent developments in the domain of web services, where many information systems in the enterprise have been implemented using Web services. Web service technology has become the preferred implementation technology for realizing the Service Oriented Architecture (SOA) paradigm [28, 268]. SOA is an architectural style that provides guidelines on how services are described, discovered and used. In particular, software applications are packaged as "services", where services are defined to be standard-based, platform- and protocol-independent to address interactions in

heterogeneous environments [28, 268].

Web services composition languages such as BPEL4WS, BPML, WSCI, XLANG, and WSFL can be used to glue services defined using WSDL[1] together. Web services provide standard specifications to simplify integration at lower levels of abstractions [82, 217], i.e., messaging, or at higher abstraction levels [250, 251, 252], i.e., service interfaces, business protocols, and also policies. In fact, what is provided at the higher levels of abstractions are languages to define the service specifications, i.e., its interface, business protocol, and policies. Enabling the analysis of service interactions, in the context of business process executions, and that of service integration, is a goal of enterprises today. In particular, business process analysis (BPA) over a wide range of information systems, services and softwares that implement the actual business processes of enterprises is required.

Business Process Analysis (BPA) is particularly concerned with the behavioral properties of enacted processes, e.g., at runtime, through *monitoring* BPs, or after execution, through using process *mining* or *querying* techniques [270]. BPA is typically structured around three different views [253, 270]: (i) process view: is concerned with the enactment of processes and is thus mainly focussed on the compliance of executed processes; (ii) resource view: is centered around the usage of resources within processes; and (iii) object view: focusses on business objects, e.g., inquiries, orders or claims, and analyze the life-cycle of these objects. These three views are populated with statistical information (e.g., the minimum, the average or the deviation of some parameter of interest) and correlations are typically established across them. Designing and maintaining BPA applications is challenging, as business processes in modern enterprises are developed by different communities of practice, reside on different levels of abstractions, and handled by different IT systems [193, 137].

[1]Proposed by IBM and Microsoft, and later published as a W3C note [279], WSDL is a general purpose XML language for describing the interface and also the implementation of Web services. The interface describes the operations that a service offers.

2.1.1 From Structured to Unstructured Processes

Business world is getting increasingly dynamic as various technologies such as social media and Web 2.0 have made dynamic processes more prevalent. For example, email communication about a process, instant messaging to get a response to a process related question, allowing business users to generate processes, and allowing front-line workers to update process knowledge (using new technologies such as process wikis) [313] makes the use of complex, dynamic and often knowledge intensive activities an inevitable task.

Such processes have flexible underlying process definition where the control flow between activities cannot be modeled in advance but simply occurs during run time [126, 121, 307]. In such cases, the process execution path can change in a dynamic and ad-hoc manner due to changing business requirements, dynamic customer needs, and people's growing skills. Examples of this, are the processes in the area of government, law enforcement, financial services, and telecommunications. In this dissertation, we use the term *ad-hoc* to refer to this category of processes.

Existing business process management tools (WfMSs and BPMSs) support well-structured [283] processes and do not provide sufficient flexibility to reflect the nature of ad-hoc processes: structured processes are fully prescribed how a future decision will be made. Ad-hoc processes can be divided into two types: unstructured and semi-structured. An unstructured process is a process that can not be reduced to well-defined rules [307], unlike well-structured processes. A semi-structured process, or *case-based* process, is a process which contains both structured and unstructured sub-processes [284].

While 'process-aware' information systems (PAIS) do a great job in increasing the productivity of organizations, it is known that their rigidity restricts their applicability [287, 141, 307]. Number of challenges for the next generation business process management have been discussed in [287]. For example, generation, recognition and application of reusable 'task patterns' and 'process patterns' suggested as an alternative to static workflows. Basic directions for the utilization of task-based approaches, to support users engaged in intensive and unstructured knowledge

work, have been discussed in [151]. A related approach has been proposed in [249] to address the problem of recommending activity steps in collaborative IT support systems, i.e., best practice processes.

An extended state of the art study in the area of flexible workflows and task management and a further approach for integrating ad-hoc and routine work is presented in [197]. Moreover, the gap between completely ad-hoc processes and rigid, predefined business processes have been discussed in [58]. These approaches provide frameworks for enabling delivery of process models, process fragments, and past cases for tasks where different stakeholders can enrich task resources and information. Moreover, they reveals major issues concerning business process flexibility and how it can be facilitated through interactive processes models [307].

Some approaches provide solutions for unstructured, e.g., in [185], and semi-structured, e.g., in [13], processes. They proposed to use document-based, task-based, and case-based techniques to manage the ad-hoc nature of such processes. Advanced techniques for building personal knowledge spaces and wiki-based collaborative document spaces are also discussed in these approaches. Some other techniques used email, which plays a central role for the exchange of tasks and task-related information in organizations [56, 156, 306]. Also, there are approaches, e.g., in [307], which support agile business processes focusing on email-based and human-to-human cooperation, where the collaboration flow determines the enterprise process flow.

Ad-hoc processes are complex not only because they are scattered across several systems and organizations, but also they require many different people (having lots of knowledge and experience) to collaborate to find the correct solution. *Case Management*, also known as case handling [314], defined as a common approach to support knowledge intensive processes. Moreover, various technologies, e.g., customer relationship management (CRM) and content management systems (CMS) have been proposed for supporting such dynamic and ad-hoc processes, however, they are not sufficient to address the key requirements of these types of processes: they are primarily driven by human participants reacting to changing context and do not follow a predetermined path.

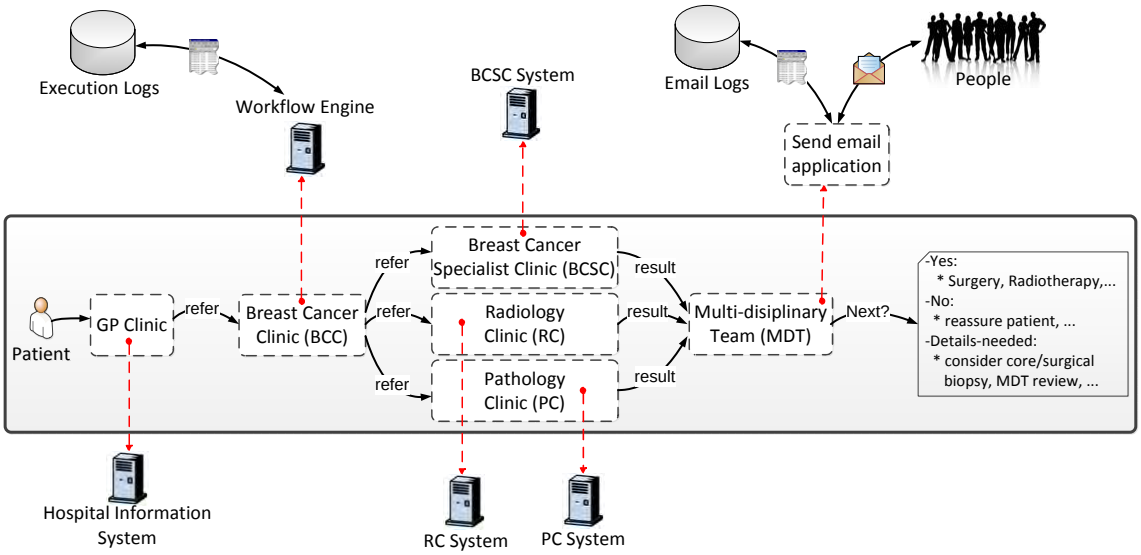


Figure 2.1: An example of the ad-hoc business process execution in an enterprise.

To understand the problem, Figure 2.1 illustrates an example scenario in the domain of semi-structured (case-based) processes. This scenario is based on breast cancer treatment cases in Velindre hospital [314]. In this scenario, a General Practitioner (GP) suspecting a patient has cancer, update patient history using hospital information system, and referring the patient to a Breast Cancer Clinic (BCC). BCC checks the patients history and requests assessments such as an examination, imaging, fine needle aspiration, and core biopsy using a workflow engine. Therefore, the workflow system refers patient to Breast Cancer Specialist Clinic (BCSC), Radiology Clinic (RC), and Pathology Clinic (PC), where these departments use their own systems, apply medical examinations, and send the results to Multi-Disciplinary Team (MDT). The results are gathered and send to MDT team members (e.g., surgeon oncologist, radiologist, pathologist, clinical and medical oncologist, and a nurse) through a send mail application. Analyzing the results and the patient history, MDT members will decide for next steps, e.g., in case of positive findings, non-surgical (Radiotherapy, Chemotherapy, Endocrine therapy, Biological therapy, or Bisphosphonates) and/or surgical options will be considered.

As illustrated in the scenario, most of the processes are conducted in an ad-hoc manner and do not rely on integrated software solutions. Instead, a mix of computerized systems and direct collaborations are used. For instance, the processing of orders in the hospital information system may involve the accounting system and

personal productivity tools of the laboratories to follow up patients. Hence, the data relevant to a business process is then scattered across multiple systems, e.g., RC, BCSC, and PC System, with no integration between them, or they are spread across multiple documents stored in the personal folders of GPs and exchanged by communication tools such as emails. Moreover, the process itself might be only partially specified or not specified at all. This yield to the fact that in many situations, stakeholders can be aware of processes but they are not able to track or understand it. Therefore, under such conditions, *organizing*, *indexing*, and *querying* ad-hoc processes data becomes of a great practical value but clearly a very challenging task as well.

Organizing Ad-hoc Process Data. In order to organize the ad-hoc process data, the first step is gathering and integration of process execution data in a *process event log* from various, potentially heterogeneous, systems and services. In general, this step involves several phases [280, 178]: (i) data analysis: is required to detect errors and inconsistencies of heterogeneous event logs; (ii) definition of transformation workflow and mapping rules: depending on the number of data sources and their degree of heterogeneity of the data, a large number of data transformation and cleaning steps may have to be executed; (iii) verification: the correctness and effectiveness of a transformation workflow/definitions should be tested and evaluated; and (iv) transformation: data transformations deal with schema/data translation and integration, and with filtering and aggregating data to be stored in the integrated process log.

The next step is providing techniques to identify entities (e.g., process stakeholders and process artifacts) and the interactions among them within such integrated process logs. In this context, most entities (structured or unstructured) in process logs are interconnected through rich semantic information, where entities and relationships among them can be modeled using graphs. Since graphs form a complex and expressive data type, there is a need for methods to organize and index the graph data. Existing database models, including the relational model, lack native support for advanced data structures such as graphs. In particular, as the graph data increases in size and complexity, it becomes important that it is managed by a

database system. There are several approaches for managing graphs in a database. A line of related work extended a commercial RDBMS engine, e.g., Oracle provides a commercial DBMS for modeling and manipulating graph data [18], and some other works used general purpose relational tables to support graph structure data, e.g., triplestores.

Triplestores, a special purpose database for the storage and retrieval of RDF [235] (Resource Description Framework) data, are optimized for the storage and retrieval of a large number of short statements in the form of subject-predicate-object, like "patient @age 35" or "Adam created patient-history", which are called triples. Much work has been done to support efficient data access on triplestores [19, 29]. While these approaches do a great job in storing and managing graph data, they fail to deliver needed performance to answer large graph queries.

A new stream of work used MapReduce [106] for processing huge amounts of unstructured data in a massively parallel way. Hadoop [330], the open source implementation of MapReduce, provides a distributed file system (i.e., HDFS[2]) and a high level language for data analysis, i.e., Pig[3]. For example, a new stream of work [199, 98, 228, 187, 297] used Hadoop for large scale graph storage and mining. They store and retrieve large number of triples in Hadoop file system.

Indexing, and Querying Ad-hoc Process Data. In order to query process graphs a graph query language is needed. A number of graph query languages have been proposed in the literature. For example GOOD [163] and GraphDB [161] are query languages which have the root in object-oriented databases. GraphQL [174] extended the relational algebraic operators, including selection, cartesian product, and set operations, to graph structures. SPARQL query language [276] is designed to efficiently query RDF data. A SPARQL query may consist of triple patterns, conjunctions, disjunctions, and optional patterns. The SPARQL query processor will search for sets of triples that match the triple patterns. GraphGrep [144] and GIndex [337], uses structural characteristics of the graph, e.g., paths and frequent patterns, to facilitate the indexing and query processing. These approaches used

[2]http://hadoop.apache.org/
[3]http://pig.apache.org/

as preprocessing filters, which remove irrelevant graphs from the underlaying data. Another work [338] identified substructures in the underlaying graphs in order to facilitate indexing. Some other works [350, 259] use the tree structures in the underlaying graph to facilitate searching and indexing.

While existing query processing solutions do a great job in querying and indexing graph data, still many challenges remain to be addressed. For example, process data is becoming increasingly large. Parallel processing, e.g., using MapReduce framework, can be used to handle such large data, however, many graph algorithms are very difficult to be parallelized [234]. Some works [206, 188, 281] focused on query optimization techniques for the Hadoop Pig to store triples and querying graphs. In addition to Pig, there are several high-level language and environments for advanced MapReduce-like systems, including SCOPE [85], Sawzall [273], and Sphere [154]. As another challenge, graph queries are becoming extremely complected: queries against a complex ontology are often lengthy, regardless to the graph query language to be used. To answer these challenges, keyword search [324] and mining methods [226], have been used to simplify queries and to semi-automate the query formation. Moreover, data services have been proposed to simplify accessing, querying, and analyzing of information, e.g., related to processes.

2.2 Data Services

In the enterprise world, data services play an important role in SOA architectures [79, 286, 164, 80]. For example, when an enterprise wishes to controllably share data (e.g., structured data such as relational tables, semi-structured information such as XML documents, and unstructured information such as commercial data from online business sources) with its business partners, via the Internet, it can use data services to provide mechanisms to find out which data can be accessed, what are the semantics of the data, and how the data can be integrated from multiple enterprises. In particular, data services are "software components that address these issues by providing rich metadata, expressive languages, and APIs for service consumers to send queries and receive data from service providers" [80].

A Web service, i.e., a method of communication between two electronic devices over the Web [28], can be specialized, as a data service, to encapsulate a wide range of data-centric operations, where these operations need to offer a semantically richer view of their underlying data in order to use or integrate entities returned by different data services [80]. Microsoft's WCF data-services framework[4], which enables the creation and consumption of OData services for the web, and Oracle's ODSI[5], which provides a wide array of data services designed to improve data access from disparate data sources for a wide range of clients, are two of a number of commercial frameworks that can be used to achieve this goal.

In this context, SOA applications will often need to invoke a service to obtain data, operate locally on that data, and then notify the service of changes that the application wishes to make to the data. Consequently, standards activity is needed in the context of data services. For example, the Open SOA Collaborations Service Data Objects (SDO) specification [286] addresses these needs by defining client-side programming models, e.g., for operating on data retrieved from a data service and for XML serializing objects, and their changes for transmission back to a data service [79]. In particular, the use of data is bound to various rules imposed by data owners and the (data) consumers should be able to find and select relevant data services as well as utilize the data 'as a service'.

Data as a service, or DaaS, is based on the concept that the data can be provided on demand to the user regardless of geographic or organizational separation of provider and consumer [320]. In particular, data services are created to integrate as well as to service-enable a collection of data sources. These services can be used in mashups [341], i.e., Web applications that are developed starting from contents and services available online, to use and combine data from two or more sources to create new services. In particular, data services will be integral for designing, building, and maintaining SOA applications [79]. For example, Oracle's ODSI supports the creation and publishing of collections of interrelated data services, similar to *dataspaces*.

[4]http://msdn.microsoft.com/en-us/data/bb931106
[5]http://docs.oracle.com/cd/E13162_01/odsi/docs10gr3/

Dataspaces, are an abstraction in data management that aim to manage large number of diverse interrelated data sources in enterprises in a convenient, integrated, and principled fashion. Dataspaces are different from data integration approaches in a way that they provide base functionality over all data sources, regardless of how integrated they are. For example, a dataspace can provide keyword search over its data sources, then more sophisticated operations (e.g., mining and monitoring certain sources) can be applied to queried sources in an incremental, pay-as-you-go fashion [166]. These approaches does not consider the business process aspects per se, however, they can be leveraged for organizing and managing ad-hoc process data.

DataSpace Support Platforms (DSSPs), have been introduced as a key agenda for the data management field and to provide data integration and querying capabilities on (semi-)structured data sources in an enterprise [166, 294]. For example, SEMEX [77] and Haystack [200] systems extract personal information from desktop data sources into a repository and represent that information as a graph structure where nodes denote personal data objects and edges denote relationships among them.

The design and development of DSSPs have been proposed in [135]. In particular, a DSSP [166, 294, 135]: (i) helps to identify sources in a dataspace and inter-related identified resources. A DSSP is required to support all the data in the dataspace rather than leaving some out, as with DBMSs; (ii) offers basic searching, querying, updating, and administering mechanisms over resources in a dataspace, including the ability to introspect about the contents. However, unlike a DBMS, a DSSP is not in full control of its data, as same data may also be accessible and modifiable through an interface native to the system hosting the data; (iii) does not require full semantic integration of the sources in order to provide useful services: there is not a single schema to which all the data conforms and the data resides in a multitude of host systems; (iv) offers a suite of interrelated (data integration and querying) services in order to enable developers focusing on the specific challenges of their applications. Queries to a DSSP may offer varying levels of service, as sometimes individual data sources are unavailable and best-effort or approximate answers can be produced at the time of the query; and (v) provides mechanisms for

enforcing constraints and some limited notions of consistency and recovery, i.e., to create tighter integration of data in the space as necessary.

Motivating applications for DSSPs includes scenarios in which related data are scattered across several systems and data sources, e.g., personal information management systems [115, 124], which are used to acquire, organize, maintain, retrieve and use information items (e.g., desktop documents, web pages and email messages) accessed during a person's lifetime, and scientific data management systems [150], which are used for record management for most types of analytical data and documentation which ensures long-term data preservation, accessibility, and retrieval during a scientific process.

In order to search and query dataspaces, a new formal model of queries and answers should be specified. This is challenging as answers will come from multiple sources and will be in different data models and schemas. Moreover, unlike traditional querying/answering systems, a DSSP can also return sources, i.e., pointers to places where additional answers can be found. Some works [165, 237] presented semantic mappings techniques to reformulate queries from one schema to another in data integration systems. Another line of related work [61, 160] focused on ranking answers in the context of keyword queries to handle the heterogeneity of resources. Some other works, e.g., in [224], focused on finding relevant information sources in large collections of formally described sources.

In dataspaces, a significant challenge is to answer historical queries which applied to heterogenous data. A line of research proposed techniques for modeling and analyzing provenance [93] (also known as lineage and pedigree), uncertainty and inconsistency of the heterogenous data in dataspaces [331, 191]. Many provenance models [93, 136, 248, 305] have been presented, motivated by notions such as influence, dependence, and causality in such systems. Moreover, the relationship between uncertainty and provenance discussed in [331].

Dataspaces are large collections of heterogeneous and partially unstructured data, and therefore, indexing support for queries that combine keywords and the structure of the data can be challenging. For example, in [116], authors proposed an indexing technique for dataspaces to capture both text values and structural infor-

mation using an extended inverted list. Their proposed framework extend inverted lists that capture attribute information and associations between data items, i.e., to support robust indexing of loosely-coupled collections of data in the presence of varying degrees of heterogeneity in schema and data. Another indexing system [109], designed to provide entity search capabilities over datasets as large as the entire 'Web of Data'. Their approach supports full-text search, semi-structural queries and top-k query results while exhibiting a concise index and efficient incremental updates. Challenges in implementing a scalable and high performance system for searching semi-structured data objects over a large heterogeneous and decentralized infrastructure have been discussed in [108], where an indexing methodology for semi-structured data have been introduced.

Recently, new class of data services designed for providing data management in the cloud [326]: the cloud is quickly becoming a new universal platform for data storage and management. In practice, data warehousing, partitioning and replication are well-known strategies to achieve the availability, scalability, and performance improvement goals in the distributed data management world. Moreover, database-as-a-service proposed as an emerging paradigm for data management in which a third party service provider hosts a database as a service [164]. Data services can be employed on top of such cloud-based storage systems to address challenges such as availability, scalability, elasticity, load balancing, fault tolerance, and heterogenous environments in data services. For example, Amazon Simple Storage Service (S3) is an online public storage Web service offered by Amazon Web Services[6].

A growing number of organizations have begun turning to various types of non-relational, *NoSQL* (not only SQL), databases such as Google Bigtable [87], Yahoo PNUTS [101], and Amazon Dynamo [107]. NoSQL is a broad class of low-cost and high performance database management systems and proposed to address RDBMSs shortcomings: ever-increasing needs for scalability and new advances in Web technology, which requires facilitating the implementation of applications as a distributed and scalable services, have created new challenges for RDBMSs [308, 218, 84, 326]. Such databases are designed to be very scalable and reliable and they consists of

[6]http://aws.amazon.com/

thousands of servers geographically distributed all over the world.

Major research challenges for providing heterogenous data management, e.g. using data services, need [80, 79, 269, 164]: (i) a dynamically reconfigurable runtime architectures, distributed service components and resources should be leveraged to create an optimal architectural configuration to both a particular users requirements and the application characteristics; (ii) an end-to-end security solutions, a full system approach to test end-to-end security solutions at both the network and application level is required; (iii) the infrastructure support for data and process integration, uniform consistent access to all heterogenous data should be provided, i.e., irrespective of the data format, source, or location; and (iv) the analytic support for the discovery and communication of meaningful patterns in (process execution) data, e.g., *business analytics*.

Business Analytics [241, 35, 210], is the family of methods and tools that can be applied to process execution data in order to support decision-making in organizations by analyzing the behavior of completed processes (i.e., Process Controlling [254]), evaluating currently running process instances (i.e., Business Activity Monitoring [83, 140, 10]), or predicting the behavior of process instances in the future (i.e., Process Intelligence [89]). In particular, the intent of process analytics can be motivated by performance, to shorten the reaction time of decision makers to events that may affect changes in process performance, or compliance considerations, to establish the adherence of process execution with governing rules and regulations [241].

In enterprises, sources for process analytics data includes activities, stakeholders, and business related artifacts (and data) which are scattered across several systems and data sources. In particular, business process analytics might include events from multiple processes, data sources outside the organization, and events from non-process-centric information systems [241, 210]. Existing works on business analytics focused more on exploration of new knowledge and investigative analysis using broad range of analysis capabilities, including: (i) trend analysis, provide techniques to explore data and track business developments; (ii) what-if analysis, in which scenarios with capabilities for reorganizing, reshaping and recalculating

data is of high interest; and (iii) advanced analysis, provide techniques to uncover patterns in businesses and discover relationships among important elements in an organization's environment.

In order to apply analytics to business data, data sources (e.g., operational databases and documents) should be streamed into data warehouse servers (e.g., relational DBMS or MapReduce engine). This can be done using ETL [322] (extract transform load) or complex event processing engines [230]. Multi-tier servers (e.g., OLAP servers, enterprise search engines, data mining engines, text analytic engines, and reporting servers) can be used on top of data warehouses for converting process execution data into knowledge and to support business users with decision making process. To understand the generated knowledge, set of interactions between business users and expert business analytics is inevitable. For example, most analytic tools are designed for quantitative analysts, not the broader base of business users who need the output to be translated into language and visualizations (appropriate for business needs). Set of front-end applications (e.g., spreadsheets, dashboards, and querying approaches) can be used to address this challenge. Moreover, visual query interface and storytelling techniques [295] can be used to facilitate the understanding of business analytics results.

Several challenges have been introduced to characterize the gap between relevant analytics and users strategic business needs [210, 241, 89] including: (i) cycle time: the time needed for the overall cycle of collecting, analyzing, and acting on enterprise data. Business constraints may impose limits on reducing the overall cycle time; (ii) analytic time and expertise: the time needed for analyzing generated knowledge from business data. Sometimes there are specific expertise necessary to analyze the result; (iii) business goals and metrics: various measurements such as cycle times, service-level variability, and even customer comments on a particular process can be used to understand process analytics. For example, crowdsourcing, i.e., a process that involves outsourcing tasks to a distributed group of people [114, 26, 27], systems can be leveraged to understand the analytics results; and (iv) goals for data collection and transformations: once metrics are identified, appropriate data must be collected and transformed into business data warehouses.

The ability for an organization to take all its capabilities and convert them into knowledge, requires analyzing data about their customers and their suppliers. The wide adoption of customer relationship management (CRM) and supply chain management (SCM) techniques has allowed enterprises to fully interface and integrate their demand and supply chains. Trkman et al. [319] discussed the impact of business analytics on supply chain performance through investigating the relationship between analytical capabilities in the plan, source, make, and deliver area of the supply chain. Moreover, online analytical processing (OLAP) techniques can be used for business reporting for sales, marketing, management reporting, budgeting and forecasting, and financial reporting. In particular, OLAP servers can be used to expose the multidimensional view of business data to applications or users and enable the common business intelligence operations, e.g., filtering, aggregation, drill-down and pivoting.

In addition to traditional OLAP servers, newer 'in-memory' business intelligence engines [275] are appearing that exploit todays large main memory sizes to dramatically improve performance of multidimensional queries. For example, HYRISE [157] and HyPer [202] are both recent academic main-memory DBMSs for mixed (OLTP and BI) workloads. Hyper creates virtual memory snapshots by duplicating pages on demand when BI queries conflict with OLTP queries. HYRISE seems to be an offline analytical tool for deciding the proper grouping of columns and physical layout to optimize performance for a given mixed workload. Also, the Blink project [39] proposed for fast processing of business intelligence queries in mere seconds, regardless of the database size, with an extremely low total cost of ownership and recently has been incorporated into IBM accelerator products. A theoretical framework for understanding the role of business analytics in obtaining performance gains has been proposed in [301]. In particular, the authors defined and used dynamic business analytics capability (DBAC) to utilize the operational, and other, data available in organizations.

Recently, engines based on the MapReduce paradigm, which originally built for analyzing Web documents and Web search query logs [106], are now being targeted for enterprise analytics [89]. These approaches can be used to address challenges in

real-time business analytics where the goal is to reduce the latency between when operational data is acquired and when analytics over that data is possible. Data platforms based on the MapReduce paradigm and its variants have attracted strong interest in the context of the 'Big Data' [231] challenge in enterprise analytics, i.e., Big Data can be considered as a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools and has the following characteristics: (i) volume: is the primary attribute of the big data and can be quantified by counting (terabytes of) records, transactions, tables, or files; (ii) variety: the big data contains a greater variety of (structured, unstructured, and semi-structured) data sources including datasets, logs, and clickstreams [219]; and (iii) velocity: big data can be described by its velocity or speed, e.g., batch, near time, real time, and streams. For example, projects like MRShare [262], Nectar [159], Scope [85], and Starfish [179] have been built on top of Hadoop to provide a self-tuning system for big data analytics.

2.3 Querying Business Processes Models and Instances

Large organizations often run hundreds or even thousands of business processes. In order to manage such large collections of business processes, BP model repositories, i.e., repositories that are structured according to a process-specific conceptual model and/or that have a process-specific interface [340], have been proposed. Moreover, set of abstract models have been introduced to describe process models, e.g., (i) finite state machines (FSM): a model for process description composed of a finite number of states (and transitions between them) and messages; (ii) process graphs: a common formalism for modeling temporal and logical dependencies in process models (e.g., BPMN [155]); and (iii) workflow-Net (WF-Net): a special class of Petri Net [257] used for modeling processes.

In this context, extracting knowledge from existing processes, e.g., retrieving relevant process models in such repositories, to better design new processes is a

challenging task and have been addressed through querying business processes [340, 33, 47, 130, 134, 293, 48, 246, 274]. In particular, querying of business processes can be divided into three categories [33]: (i) querying business process models: can help business analysts to search for certain patterns within enterprise repository of BPs; (ii) querying running instances of business processes: can help in monitoring the status of running processes, tracing the progress of execution, and making ad-hoc queries about a status of a certain process; and (iii) querying execution logs of completed business processes (also known as business *process mining*): can help in extracting information from event logs to capture the business process as it is being executed.

The main limitation of these approaches is that they assume an ideal world in which: (i) the business process models are pre-defined and available; and (ii) the execution of the business processes is achieved through a business process management system (e.g., BPEL) where the execution traces should comply with the defined process models. In this section we discuss querying process models and instances and in Section 2.4 we present the current state-of-the-art in process mining.

Querying BP Models. Several approaches have been proposed for process matching and retrieval problem in various application domains such as Web services discovery and integration [57, 321], retrieving scientific workflows [145, 45], retrieving business processes in repository [112], auto-completion mechanism for modeling processes [129], and version management [258, 215]. These approaches for matching process models are based on different abstract models introduced earlier, e.g., FSM, process graphs, and WF-Net. In particular, the comparison of two process models can be done from two perspectives: (i) structural [153, 102, 133]: this approach consider only the process topologies (i.e., the type of the control flow specified in the process model); and (ii) behavioral [334, 258, 117]: in this approach the process execution semantics are considered by comparing the traces (i.e., an execution of a process instance starting from initial state and ending into a final state) generated by the two processes.

Query languages for process retrieval [59, 47, 33, 304, 299, 205] enable comparing a process model with a set of process models in a repository. Bernstein et al. [59]

proposed a service retrieval approach that captures service semantics using process models, and applies a pattern-matching algorithm to find the services with the behavior of user interest. BP-QL [47] proposed to query business processes expressed in BPEL [130]. BP-QL is a graphical language where queries are modeled using process patterns and allow to retrieve paths inside a single process graph and to zoom across process graphs of the used services (in case of compound activities).

BPMN-Q [33] is oriented to query generic process modeling concepts using Business Process Modeling Notation [155] (BPMN). In particular, BPMN-Q provides a visual interface, supports the navigation of process structures to answer queries, supports the notion of paths between nodes in the process graph, and provide techniques to modify the results of queries to create new queries to support the iterative nature for the querying scenarios. In this context, the result of the BPMN-Q query can be either the whole process model containing a match to the query or only the matching part. Moreover, the authors proposed the semantic expansion of BPMN-Q queries in [34] which supports various vocabularies which may express the same concept.

Shen et al. [304] proposed a behavior model for web services using automata and logic formalisms, where a web service process is modeled as an extended nondeterministic finite automaton, i.e., the automaton contains states for sending/receiving messages and states for performing internal activities. They defined the behavior pattern as a regular expression over the set of activities and messages. Moreover, they proposed a query language, to express temporal and semantic properties on service behaviors, and query evaluation algorithms, to improve the query performance using optimization approach such as RE-tree and some heuristics. VisTrails [299] proposed as a scientific workflow and provenance management system that provides an intuitive interface for querying data flows, i.e., a data flow is composed of modules which define specific operations and connections which specify the conceptual flow of data between modules. The main idea behind VisTrails is to leverage query-by-example and visualization techniques where analogies are used as first-class operations to guide scalable interactions.

Unlike the above approaches, which proposed dedicated process query language, the work presented in [205] investigated the use of similarity measures in a pro-

cess ontology retrieval task using the iSPARQL framework. In particular, to enable querying similar entities in semantic web knowledge bases, iSPARQL extends SPARQL [276] graph query language with similarity operators. Several other studies focused on the comparison of semantic business processes either for retrieval, discovery, matchmaking, or process alignment. For example, the works in [45, 129] proposed an approach to semantically align business processes originally represented as Petri nets [257]. The authors employed similarity measures from different categories to measure the affinity between elements of Petri nets. Klusch et al. [207] presented an approach to perform hybrid semantic Web service matchmaking using both semantic similarity measures and logic-based reasoning techniques.

Another line of related work proposed indexing and filtering techniques [86, 233, 339] for efficient retrieval of process models in large repositories. The RE-tree [86] is an index structure for regular expressions based on R-tree [162] (i.e., a dynamic index structure for spatial searching) in which each node entry is a finite state automata. RE-tree have been constructed to accelerate the search process. Mahleko et al. [233] proposed an approach for indexing and matching business process represented as finite state machines, where the process matchmaking is reduced to the intersection of state machines. Yan et al. [339] proposed a technique for improving the speed of business process similarity search through matching a query process model against a collection of process models specified as process graphs. The proposed feature-based filtering technique is used to efficiently estimate model similarities and classify them as relevant, irrelevant or potentially relevant to a query model.

Querying BP Instances. Querying running instances of business processes [48, 246, 274] represents another flavor of querying processes. It can be considered as a tool in the hand of administrator of a business process enactment engine to monitor the status of running processes and trace the progress of execution. Such queries can be used to discover many problems such as: detecting the occurrence of deadlock situations or recognizing unbalanced load on resources. Querying BP instances is in flavor of Business Activity Monitoring (BAM) [83, 140, 10]. BAM intends to provide real-time business performance indicators to improve the speed and effectiveness of business operations through discovering business process models, where tracking and

analyzing of process execution will be needed. Available solutions for BAM, e.g., Oracle BAM [264], focused on processing real-time events at the middle-ware level using event processing systems [111].

Several approaches [48, 49, 41, 43, 42, 246, 274, 64, 245] have been proposed in literature for specifying monitoring directives externally to the BPEL processes and for supporting the run-time monitoring of these directives. For example, a BP-Mon monitoring query [48, 49] is represented as a BPEL process that capture interesting events of monitored processes, e.g., to monitor the bid cancelation events of a bidding process, where a manager could be interested in tracking users who make cancelations too often. BP-Mon allows users to visually define monitoring tasks and associated reports, using a simple intuitive interface, similar to those used for designing BPEL processes. In particular, monitoring the execution of BPs for frequent and interesting patterns is critical for enforcing business policies and meeting efficiency and reliability goals in enterprises [48].

The focus in [41] is on the specification of properties which may span over multiple executions of BPEL processes and that aggregate information about all these executions. The proposed architecture separates the BPEL execution engine and the monitoring engine. Baresi et al. [43] focused on the specification of monitoring directives that can be activated and de-activated for each process execution. The actual monitoring of these directives is performed by weaving them into the process they belong to. Detailed comparison of these two approaches ([41] and [43]), e.g., in terms of the events they are able to monitor or the level of integration of process execution and monitoring, have been presented in [44]. The approach presented in [42] investigated how to monitor dynamic service compositions with respect to contracts expressed via assertions on services. They represented dynamic compositions as BPEL processes which can be monitored at run-time to check whether individual services comply with their contracts.

PQL [246] is a SQL-based query language with a main focus on querying biological pathways, where the representation of a single enactment of a workflow process is called a process instance and the execution of a process instance considered as execution of a set of activity instances. In this context, every activity instance that

is an atomic activity is performed by one workflow participant, i.e., a resource that may participate in process execution is called a workflow participant which adapted to proposed workflow process definition and instance metamodel and a workflow control data. The workflow process metamodel defines workflow entities, their relationships and basic attributes. In PQL, the similarly to the process instance and the behavior of an activity instance is represented by state diagram and stored as activity instance state entities.

Pistore et al. [274] presented an approach based on model checking to monitor the composition as well as the execution of web services. Starting with an abstract BPEL specification, they focused on a model checking approach to find a composition that reaches the goals as well as synthesizing a monitoring component that checks whether the external services behave as specified in the protocol. Bianculli et al. [64] proposed a process monitoring approach based on an algebraic specification language focusing on monitoring functionality of conversational services, whose behavior depends on the local state resulting from the client-service interaction. Furthermore, a model-driven methodology for a top-down development of a process-oriented IT support based on a SOA introduced in [245]. The proposed approach is based on a SOA techniques, considered monitoring requirements for business process controlling, and introduced metamodels for the specification of process performance indicators in conjunction with the necessary monitoring. In next section, we discuss the current state-of-the-art in querying execution logs of completed business processes, i.e., *process mining*.

2.4 Process Mining

In order to analyze process execution data, querying execution logs of completed business processes (i.e., process mining [2, 8]) received continuous attention in research. The goal of process mining is to simplify process queries and to semi-automate the query formation in order to easily establish links between the actual processes, their data, and the process models. In particular, process mining helps in discovering and improving real processes by extracting knowledge from event logs

through using process modeling/analysis, machine learning, and data mining techniques. The main concern of these approaches is to reverse engineer the definitions of business process models from execution logs of information system components. Moreover, depending on how much details the log gives, they can provide statistics about many aspects of the business processes such as: the average duration of process instances or the average resource consumptions.

Recently, the IEEE Task Force on Process Mining released a manifesto describing guiding principles and challenges in process mining [6], where the goal is to increase the maturity of process mining as a new tool to improve the (re)design, control, and support of operational business processes. In particular, process mining challenges include [6, 11, 3, 8, 2]:

- mining hidden and duplicate tasks: one of the basic assumptions of process mining is that each event is registered in the log. Consequently, it is challenging to find information about tasks that are not recorded. Moreover, the presence of duplicate tasks is related to hidden tasks and refers to the situation that one can have a process model with two nodes referring to the same task.

- loops: in a process it may be possible to execute the same task multiple times, i.e., this typically refers to a loop in the corresponding process model.

- temporal properties: the temporal metadata (e.g., event timestamps) can be used for adding time information to the process model or to improve the quality of the discovered process model.

- mining different perspectives: understanding process logs in terms of its scope and details is challenging specially as it is subjective: depend on the perspective of the process analyst.

- dealing with noise and incompleteness: the log may contain noise (e.g., incorrectly logged information) and can be incomplete (e.g., the log does not contain sufficient information to derive the process).

- gathering data from heterogeneous sources: in modern enterprises, information about process execution is scattered across several systems and data sources.

- visualization techniques: helps presenting the results of process mining in a way that people actually gain insight in the process.

- delta analysis: is used to compare the two process models and explain the differences. It can be useful as process models can be descriptive or normative.

Tree types of process mining are recognized [2, 4, 6]: (i) discovery: this technique takes an event log and produce a model without using any a priority information. For example, the α-algorithm [12] takes an event log and produce a Petri net [257] model which explains the behavior recorded in the log; (ii) conformance: in this technique an existing process model is compared with an event log of the same process. Conformance checking can be used to check if reality, as recorded in the log, conforms to the model and vice versa. For example, the conformance checking algorithm proposed in [289] can be used to quantify and diagnose deviations; and (iii) enhancement: this technique can be used to extend or improve an existing process model using information about the actual process recorded in some event log. In this context, two types of enhancement are recognized: repair, can be used to modify the model to better reflect reality, and extension, can be used to add a new perspective to the process model by cross-correlating it with the log.

A recent book [2] and surveys [4, 8, 11] discuss the entire process mining spectrum from process discovery to operational support. Moreover, growing number of software vendors added process mining functionality to their tools. For example, ProM [7] offers a wide range of tools related to process mining and process analysis. In particular, ProM is a workflow discovery prototype tool that offers some of above approaches. Agrawal et al. [20] proposed an approach to apply process mining in the context of workflow management systems and to address the problem of model construction. Datta [104] proposed algorithms for the discovery of business process models. Also, similar approaches in the the context of software engineering processes have been addressed in [100]. Herbst [177] presented a learning algorithm that is capable of inducing concurrent workflow models. The proposed approach focused on processes containing duplicate tasks and presented a specialization-based technique for discovering sequential model of process logs represented using HMM (Hidden

Markov Model).

Conformance checking techniques [17, 256] are used to relate events in the log to activities in the model. Adriansyah et al. [17] presented techniques to measure the conformance of an event log for a given process model. The approach quantifies conformance and provides intuitive diagnostics and has been implemented in the ProM framework. Munoz-Gama et al. [256] presented an approach to enrich the process conformance analysis for the precision dimension. Some other examples of approaches focused on precision for: measuring the percentage of potential traces in the process model that are in the log [152], comparing two models and a log to see how much of the first models behavior is covered by the second [238], comparing the behavioral similarity of two models without a log [118], and using minimal description length to evaluate the quality of the model [78].

Enhancement techniques heavily rely on the relationship between elements in the model and events in the log. These relationships may be used to: (i) replay the event log on the model, e.g., bottlenecks can be identified by replaying an event log on a process model while examining the timestamps [2]; (ii) modify the model to better reflect reality; and (iii) add a new perspective to the process model by cross-correlating it with the log. Subramanian et al. [309] proposed an approach for enhancing BPEL engines with facilities that permit satisfying self-healing requirements. Moreover, the concept of self-healing as a part of autonomic computing has been proposed in [203], where self-healing systems will automatically detect, diagnose, and repair localized problems resulting from failures. A diagnostic reasoning techniques and diagnosis-aware exception handlers for exception handling proposed in [32]. Also, a framework for providing a proxy-based solution to BPEL, as an approach for dynamic adaptation of composite Web services, presented in [132].

2.5 Observations

Understanding, analyzing, and ultimately improving business processes is a goal of enterprises today. As discussed in this chapter, most related work in the area of analyzing business process execution assumes well defined processes, however,

business world is getting increasingly dynamic and there are cases where the process execution path can change in a dynamic and ad-hoc manner. Current state-of-the-art in querying business processes does not provide sufficient techniques for the analysis of ad-hoc process data. For example, some of the basic assumptions of existing BPs querying techniques is that each event should be registered in the log, the BP models should be pre-defined and available, and the execution traces should comply with the defined process models.

In particular, the understanding of business processes and analyzing BP execution data is difficult due to the lack of documentation and especially as the process scope and how process events across these systems are correlated into process instances are subjective: depend on the perspective of the process analyst. Consequently, there is a need for querying approaches that enables analysts to analyze the process events from their perspectives, for the specific goal that they have in mind, and in an explorative manner.

Moreover, most objects and data in the integrated process logs are interconnected, forming complex, heterogeneous but often semi-structured networks and can be modeled using graphs. Understanding modern business processes entails identifying the relationships among entities in process graphs. Viewing process logs as a network, process graphs, and studying systematically the methods for mining such networks, of events, actors and process artifacts, is a promising frontier in database and data mining research: process mining provides an important bridge between data mining and business process modeling and analysis [6].

There are many studies on the analysis of graphs, such as network measures [327], statistical behavior study [260], modeling of trend and dynamic and temporal evolution of networks [221, 184, 211], clustering [310, 277], ranking [312, 311], and similarity search [338]. All these approaches can be leveraged for mining and analyzing process graphs. Moreover, for effective discovery of ad-hoc process knowledge it is important to enhance process data by various data mining methods, i.e., to help data cleaning/integration, trustworthiness analysis, role discovery, and ontology discovery, which in turn help improving business processes.

To address these challenges, set of works [252, 291, 290] focused on the correla-

tion discovery between events in process logs, i.e., event correlation is the process of finding relationships between events that belong to the same process execution instance. In particular, the problem of event correlation can be seen as related to that of discovering functional dependency [229, 272] in databases. These works are complementary to process mining techniques as they enable grouping events in the log into process instances that are then input to process mining algorithms.

Some other related works, focused on converting process execution data into knowledge to support the decision making process [241, 35, 210]. They presented a family of methods and tools for developing new insights and understanding of business performance based on collection, organization, analysis, interpretation, and presentation of ad-hoc process data. While existing analytics solutions, e.g., OLAP techniques and tools, do a great job in collecting data and providing answers on known questions, key business insights remain hidden in the interactions among objects and data. Existing approaches [169, 317, 92, 349, 278, 208, 198, 131], in on-line analytical processing on graphs, took the first step by supporting multi-dimensional and multi-level queries on graphs, however, much work needs to be done to make OLAP heterogeneous networks a reality [168].

In our approach, we focus on providing a framework, simple abstractions and a language for the explorative querying and understanding of process graphs from various user perspectives. The framework caters for lifecycle activities important for wide range of processes, from unstructured to structured, including understanding, analyzing, correlating, querying, and exploring process execution data in an interactive manner. Moreover, the framework provides techniques for applying existing mining algorithms and analytics to process data. We propose a query language for facilitating the analysis of process graphs based on the two concepts of folders and paths, which enable a process analyst to group related entities in the graph or find patterns among entities. We use this framework for organizing, indexing, and querying ad-hoc process data. We provide abstractions for analyzing cross-cutting aspects in ad-hoc processes and supporting analytics over ad-hoc process data.

2.6 Summary

In this chapter, we have given an overview of existing abstractions and specifications in the areas of business processes, ad-hoc processes, data-spaces, data-services and querying business processes. In business processes, central concepts such as processes management, workflow management systems, Web services, business process analysis, business activity monitoring, and process mining and querying approaches have been presented. Moreover, existing methods and techniques for understanding and analyzing ad-hoc processes have been introduced and challenges in organizing, querying, and analyzing ad-hoc processes data have been addressed.

We discussed the current state-of-the-art in data-space and data services as they can be leveraged in order to organize, index, and query ad-hoc process data. We discussed that the exponential growth in the amount of business data needs revolutionary techniques for data management, analysis and accessibility. Moreover, the current state-of-the-art in business analytics have been presented as organizations today create vast amounts of transactional data and converting process execution data into knowledge, to support the decision making process, is the focus of business analytics. In summary, there are three main limitations with respect to current approaches:

- Understanding of ad-hoc processes and analyzing BP execution data will be difficult as the information about process execution is scattered across several data sources. Moreover, due to the lack of documentation, the process scope and how process events across these systems are correlated into process instances are subjective: depend on the perspective of the process analyst.

- The structure of process graphs can help in understanding, predicting and optimizing the behavior of dynamic processes, however, in many cases process artifacts evolve over time, as they pass through the business's operations. Consequently, identifying the interactions among people and artifacts becomes challenging and requires analyzing the cross-cutting [204] aspects of process artifacts such as versioning and provenance. Analyzing these aspects will expose many hidden interactions among process related entities.

- In modern enterprises, businesses accumulate massive amounts of data from a variety of sources, where business analytics can help in understanding the business data with an eye to predicting and improving business performance in the future. While existing analytics solutions, e.g., OLAP techniques and tools, do a great job in collecting data and providing answers on known questions, key business insights remain hidden in the interactions among objects and data: most objects and data in the process graphs are interconnected, forming complex, heterogeneous but often semi-structured networks.

Enabling above-mentioned analysis requires a model and a query language for representing and querying process entities (e.g., events, artifacts, and actors), relationships among them, and the evolution of business artifacts over time. Moreover, the model should support multi-dimensional/-level views and analytics over ad-hoc processes data. We will address these challenges in the following chapters of this thesis.

Chapter 3

Organizing, Indexing, and Querying Ad-hoc Processes Data

3.1 Introduction

A business process (BP) consists of a set of coordinated tasks and activities employed to achieve a business objective or goal. In modern enterprises, BPs are realized over a mix of workflows, IT systems, Web services and direct collaborations of people. The understanding of business processes and analyzing BP execution data (e.g., logs containing events, interaction messages and other process artifacts) is difficult due to the lack of documentation and especially as the process scope and how process events across these systems are correlated into process instances are subjective: depend on the perspective of the process analyst.

As an example, one may want to understand the delays to the ordering process (the end-to-end from ordering to the delivery) for a specific customer, while another analyst is only considered with the packaging process for any orders in the shipping department. Certainly, one process model would not serve the analysis purpose for both situations. Rather there is a need for a process-aware querying approach that enables analysts to analyze the process events from their perspectives, for the specific goal that they have in mind, and in an explorative manner. In this chapter, we focus on addressing this problem.

The first step of process analysis is gathering and integration of process execution data in a *process event log* from various, potentially heterogeneous, systems and services. We assume that execution data are collected from the source systems and transformed into an event log using existing data integration approaches [280], and we can access the event metadata and the payload content of events in the integrated process log.

The next step is providing techniques to enable users define the relationships between process events. The various ways in which process events may be correlated are characterized in earlier works ([46, 252]. In [252], the authors introduced the notion of a *correlation condition* as a binary predicate defined on the attributes of event payload that allows to identify whether two or more events are potentially related to the same execution instance of a process. We use the concept of correlation condition to formulate the relationships between any pairs of events in the log.

In this chapter, we introduce a data model for process events and their relationships, and a query language to query and explore events, their correlations and possible aggregation of events into process related abstractions. We introduce two concepts of *folders* and *paths*, which help in partitioning events in logs into groups and paths, in order to simplify the discovery of process related relationships (e.g., process instances) and abstractions (e.g., process models). We define a folder node as a placeholder for a group of inter-related events. We use a path node to represent the set of events that are related to each other through transitive relationships. These paths may lead into discovering process instances.

In summary, we present a novel framework for organizing, indexing, and querying ad-hoc processes data. The unique contributions of this chapter are as follows:

- We propose a graph data model that supports typed and untyped entities (e.g., events, artifacts, and actors), and introduce *folder* and *path* nodes as first class abstractions. A folder node contains a collection of related events, and a path node represent the results of a query that consists of one or more paths in the process graph based on a given correlation condition.

- We present a process event query language and graph-based querying process-

ing engine called FPSPARQL, which is a Folder-, Path-enabled extension of SPARQL [276]. We use FPSPARQL to query and analyze events, folder and path nodes in order to analyze business process execution data.

- We describe the implementation of FPSPARQL, the results of the evaluation of the performance of the engine, and the quality of results over large event logs. The evaluation shows that the approach is performing well.
- We provide a front-end tool for the exploration and visualization of results in order to enable users to examine the event relationships and the potential for discovering process instances and process models.

The remainder of this chapter is organized as follows: we present a case study on process event logs in Section 3.2. In Section 3.3 we give an overview of the query language and the data model. We present the FPSPARQL query language in Section 3.4. In Section 3.5 we show how we use the query language for analyzing the case study process log. In Section 3.6 we describe the query engine implementation and in Section 3.7 we discuss the evaluation experiments. Finally, we discuss related work in Section 3.8, before concluding the chapter in Section 3.9.

3.2 Process Log Analysis: Example Scenario

Let us assume a set of web services that are interacting to realize a number of business processes. In this context, two or more services exchange messages to fulfil a certain functionality, e.g., to order goods and deliver them. The events related to messages exchanged during service conversations may be logged using various infrastructures [252]. A generic log model L represented by set of messages $L = \{m_1, m_2, ..., m_n\}$ where each message m is represented by a tuple $m_i \in A_1 \times A_2 \times ... \times A_k$ [250]. Attributes $A_1, ..., A_k$ represent the union of all the attributes contained in all messages. Each single message typically contains only a subset of these attributes and $m_x.A_i$ denotes the value of attribute A_i in message m_x. Each message m_x has a mandatory attribute τ that denotes the timestamp at which the event (related to the exchange of m_x) has been recorded.

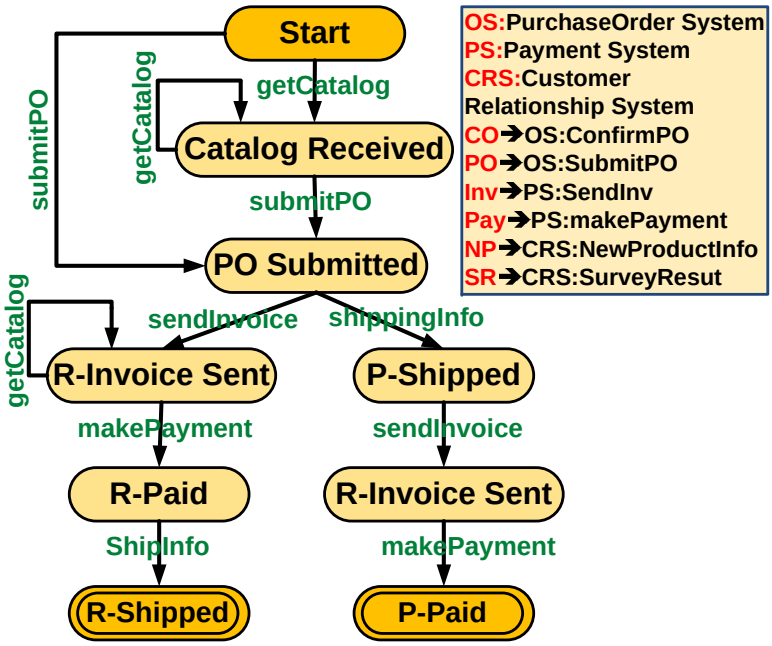


Figure 3.1: A simplified business process in SCM log for retailer service.

In particular, we use the interaction log of a set of services in a supply scenario provided by WS-I (the Web Service Interoperability organization[1]), referred in the following as SCM. Figure 3.1 illustrates a simplified business process in SCM (Supply Chain Management) log for retailer service and Table 3.1 shows an example of the SCM log. The log of the SCM business service contains 4,050 messages, 14 service operations (e.g., CO, PO, and Inv), and 28 attributes (e.g., sequenceid, custid, and shipid).

We will use this log to demonstrate how various users use the querying framework introduced in this chapter for exploring and understanding process event logs. For example, we will show how an analyst can: (i) use correlation conditions to partition SCM log, e.g., see Examples 5 and 6 in Section 3.5; (ii) explore the existence of transitive relationships between messages in constructed partitions to identify process instances, e.g., see Examples 7 and 8 in Section 3.5; and (iii) analyze discovered process instances by discovering a process model to understand the result of the query in terms of process execution visually, e.g., see Example 9 in Section 3.5.

[1]http://www.ws-i.org

Table 3.1: Example of SCM service interaction log.

MessageID	Service	Operation	OrderID	InvoiceID	CustomerID	ShipID	QuoteID	PayID	...
00001	Catalogue	get			21				...
00002	Quoting	RFQuote			21		Q1		...
00003	Ordering	PO	O1				Q1		...
00004	Ordering	RejectOrder	O1						...
00005	Ordering	PO	O2						...
00006	Invoice	Invoice	O2						...
00007	Payment	Pay		I2				P2	...
00008	Shipping	Ship		I2		S2		P2	...
00009	Ordering	OrderFulfil	O2			S2			...
...	...	...	...	...	...	...	...	...	...

3.3 Organizing and Indexing Ad-hoc Process Data

We introduce a graph-based data model for modeling the process entities (e.g., events, artifacts, and actors) in process logs and their relationships, in which the relationship among entities could be expressed using regular expressions. In order to enable the explorative querying of the process logs represented in this model, we propose the design and development of an interactive query language that operates on this graph-based data model. The query language enables the users to find entities of their interests and their relationships.

The data model includes abstractions which act as higher level entities of related entities to browse the results as well as store the result for follow-on queries. The process events in these higher level entities could be used for further process-specific analysis purposes. For instance, inspecting entities for finding process instances, as well as applying process mining algorithms on containers having process instances for discovering process models. Figure 3.2 shows an overview of the steps in analyzing process event logs (i.e. preprocessing, partitioning, and analysis) in our framework, which is described in the following sections.

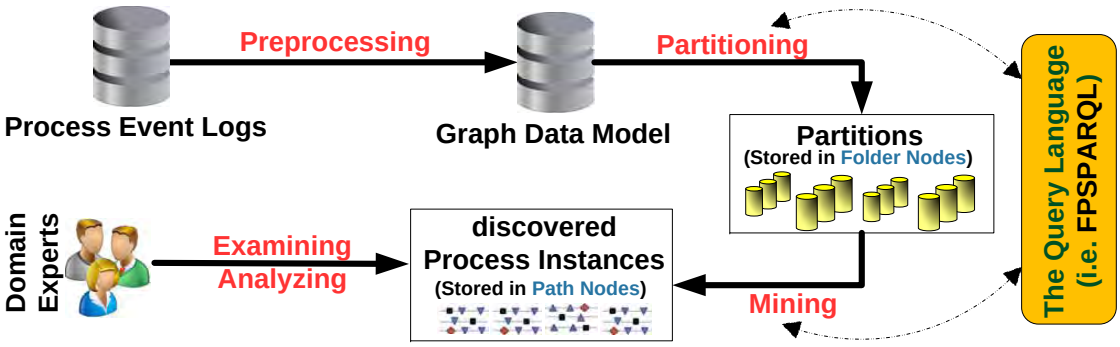


Figure 3.2: Event log analysis scenario.

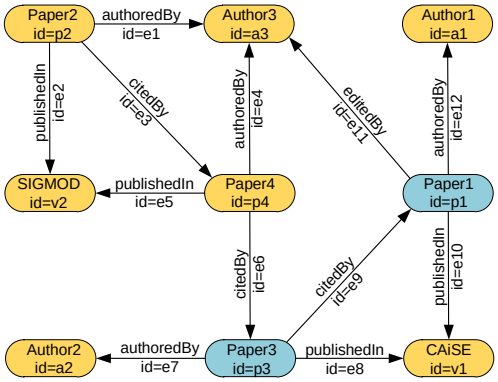
3.3.1 Data Model

We propose to model a process log as a graph of typed nodes and edges. We define a graph data model for organizing a set of entities as graph nodes and entity relationships as edges of the graph. This data model supports: (i) uniform representation of nodes and edges. In order to support querying attributed graphs, both nodes and edges are treated as first class citizens where any node/edge can be described by an arbitrary set of attributes; (ii) entities, which is represented as a data object that exists separately and has a unique identity. Entities can be structured or unstructured; (iii) folder nodes, which contain entity collections. A folder node represents the results of a query that returns a collection of related entities; and (iv) path nodes, which refer to one or more paths in the graph, which are the result of a query, too. A path is the transitive relationship between two entities. Entities and relationships are represented as a directed attributed graph $G = (V, E)$ where V is a set of nodes representing entities, folder or path nodes, and E is a set of directed edges representing relationships among nodes.

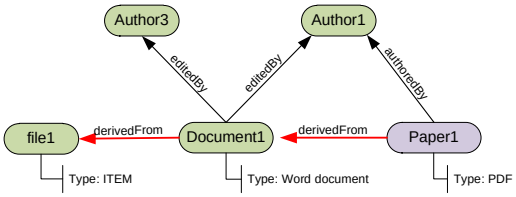
In order to understand entities, relationships, and folder/path abstractions, we present a simple example in bibliographical networks illustrated in Figure 3.3.

Entities

Entities could be structured or unstructured. Structured entities are instances of entity types. An entity type consists of a set of attributes. Unstructured entities, are also described by a set of attributes but may not conform to an entity type.



(A) Sample bibliographical network graph.



(B) Sample historical graph for the ancestry relationships of 'paper1' illustrated in Figure 1-A.

Object Store		
Subject (entity-id)	Predicate (entity-attribute)	Object (attribute-value)
P1	@Type	Paper
P1	@class	entityNode
P1	@Title	Paper1
A1	@Type	Author
A1	@class	entityNode
A1	@Name	Author1
V1	@Type	Venue
V1	@class	entityNode
V1	@Name	CAiSE
E1	@type	directedLink
E1	@Class	Edge
E1	@Label	authoredBy
Folder1	@Class	folderNode
Folder1	@Name	CAiSEPapers
Path1	@Class	pathNode
Path1	@Name	p2p1Path
...	...	...

(C) Sample entity-store for the graph represented in Figure 1-A.

Link Store		
Subject (node-from)	Predicate (edge)	Object (node-to)
P2	E1	A3
P2	E2	V2
P2	E3	P4
P4	E4	A3
P4	E5	V2
P4	E6	P3
P3	E7	A2
P3	E8	V1
P3	E9	P1
P1	E12	A1
P1	E11	A3
P1	E10	V1
...	...	...

(D) Sample triplestore for the graph represented in Figure 1-A.

Folder Store			
Folder-id	Subject (node-from)	Predicate (edge)	Object (node-to)
Folder1	P3	E7	A2
Folder1	P3	E8	V1
Folder1	P3	E9	P1
Folder1	P1	E12	A1
Folder1	P1	E11	A3
Folder1	P1	E10	V1
...	...	...	...

(E) Sample folder-store for the result of Example 2.

Path Store				
Path-id	Paths Include	Subject (node-from)	Predicate (edge)	Object (node-to)
Path1	Path #1	P2	E3	P4
Path1	Path #1	P4	E6	P3
Path1	Path #1	P3	E9	P1
...		...	...	...

(F) Sample path-store for the result of Example 3.

Figure 3.3: Representation of the Graph, Folder, and Path.

This entity model offers flexibility when types are unknown and take advantage of structure when types are known. We assume that all unstructured entities are instances of a generic type called *ITEM*. ITEM is similar to *generic table* in [263]. We store entities in the *object store*.

Example 1 . *Consider the bibliographical network in Figure 3.3-A. In this graph we have entity types such as author, paper and venue. The graph in Figure 3.3-B illustrates the creation (i.e., ancestry relationships) of 'paper1'. 'paper1' and 'document1' are structured entities. 'file1' is an unstructured entity with unknown entity type. The sample object-store in Figure 3.3-C contains all the nodes and edges in this graph, i.e., the model supports uniform representation of nodes and edges in the graph. The link-store in Figure 3.3-D, contains all the directed links between entities. We use triplestores to store graphs: a triplestore is a purpose-built database for the storage and retrieval of triples, a triple being a data entity composed of subject-predicate-object, like "Bob @age 35" or "Bob knows Fred". See Section 3.6.2 for details.*

Relationships

A relationship is a directed link between a pair of entities, which is associated with a predicate (i.e., a regular expression) defined on the attributes of entities that characterizes the relationship. A relationship can be *explicit*, such as 'was triggered by' in '$event_1$ *wasTriggeredBy* $event_2$' in a BPs execution log. Also a relationship can be *implicit*, such as a relationship between an entity and a larger (composite) entity that can be inferred from the nodes.

3.3.2 Representing and Organizing Ad-hoc Process Data

In this section we introduce abstractions and methods that enable the querying and exploration of process entities (e.g., events, artifacts, and actors) and the relationships among them, which facilitate the discovery of potential process models and process instances.

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. The folder concept is akin to that of a database view defined on a graph. However, a folder is part of the graph and creates a higher level node that other queries could be executed on it. Folders can be nested, i.e., a folder can be a member of another folder node, to allow creating and querying folders with relationships at higher levels of abstraction. A folder may have a set of attributes that describes it. A folder node is added to the graph and can be stored in the database to enable reuse of the query results for frequent or recurrent queries.

Example 2 . *[Correlation Condition] A correlation condition ψ proposed as a binary predicate defined on the attributes of entities that allows to identify whether two or more entities (in a given process graph) are potentially related [252]. We call two entities correlated if the predicate is evaluated to true. For example, consider the correlation condition* x.venue='CAiSE' *where x is an instance of type* paper. *This query, groups set of papers published in 'CAiSE' conference. As illustrates in the Figure 3.3-A the result of this query is the set {'paper1','paper3'}. We add a folder node to the original graph, and store the result of this query in the folder store (Figure 3.3-E). For this purpose, we filter all the tuples in the link-store (Figure 3.3-D) whose column 'node-from' is 'paper1' or 'paper3'. Properties of this folder will be stored in the object-store (Figure 3.3-C). In the folder-store, the nodes under the column 'subject' are the members of this folder.*

Path Nodes

A path is a transitive relationship between two entities showing a sequence of edges from the start entity to the end. This relationship can be codified using regular expressions [23, 50] in which alphabets are the nodes and edges from the graph. We define a path node for each query that results in a set of paths. We use existing reachability approaches (See Appendix-A) to verify whether an entity is reachable from another entity in the graph. Some reachability approaches (e.g., all-pairs shortest path [50]) report all possible paths between two entities. We define a path node as a triple of (V_{start}, V_{end}, RE) in which V_{start} is the starting node, V_{end} is the ending

node, and *RE* is a regular expression. We store all paths of a path node in the *path store*.

For example, in a bibliographic graph, one possible query that results in a set of paths in the graph is "find all conferences for papers citing a given paper". Such a query will help to understand which conferences cite papers from a given conference. The details of such a query is a set of paths from the current paper to the publication venue of papers citing the given paper. In cases, where the second entity of a target path query is not given, the query requires a maximum length to limit the search for matching end entities within that maximum length from the start entity.

Example 3 . *Consider the bibliographical network presented in Figure 3.3-A. Assume we are interested in finding occurrences of following pattern: 'paper2' cited-by 'paper1' possibly indirectly. This path can be written as the regular expression "paper (citedBy paper)+", where regular expression elements (nodes and edges from the graph) can be defined by a set of attributes. The plus sign indicates that there is one or more of the preceding element. The result of this example stored in the sample path-store presented in Figure 3.3-F.*

Example 4 . *Consider the historical graph presented in Figure 3.3-B. The ancestry relationships, found in the provenance [93], form a directed graph, i.e., historical graph. When an object A is found to have been derived from some other object B, we say that there is an ancestry path between A and B [181]. Figure 3.3-B illustrates the ancestry path between 'paper1' and 'file1' (follow the red edges in the figure). Ancestry paths through historical graphs form the basis of many provenance queries.*

3.4 Querying Ad-hoc Process Data

As mentioned earlier, we model process logs as a graph. In order to query this graph a query language is needed. Among languages for querying graphs, SPARQL [276] is an official W3C standard and based on a powerful graph matching mechanism. However, SPARQL does not support the construction and retrieval of subgraphs. Also paths are not first class objects in SPARQL [276, 181]. In order to analyze

BPs execution data, we propose a graph processing engine, i.e. FPSPARQL [55] (a Folder-, Path-enabled extension of the SPARQL), to manipulate and query entities, and folder and path nodes. We support two levels of queries: (i) Entity-level Queries: at this level we use SPARQL to query entities in the process logs; and (ii) Aggregation-level Queries: at this level we use FPSPARQL to construct and query folder and path nodes.

3.4.1 Entity-Level Queries

At this level, we support the use of SPARQL to query entities and their attributes in the process logs. SPARQL is a declarative and extendable graph query language, standardized by the World Wide Web Consortium, for semantic Web. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. SPARQL also supports extensible value testing and constraining queries. The results of SPARQL queries can be results sets or RDF graphs. A basic SPARQL query has the following form:

```
select ?variable1 ?variable2 ...
where {
   pattern1.
   pattern2.
   # Other Patterns
}
```

Each pattern consists of *subject*, *predicate* and *object*, and each of these can be either a variable or a literal. The query specifies the known literals and leaves the unknowns as variables. To answer a query we need to find all possible variable bindings that satisfy the given patterns. We use the '@' symbol for representing attribute edges and distinguishing them from the relationship edges between graph nodes. As an example, we may be interested in retrieving a list of messages in SCM log (Section 3.2) that have the same value on 'requestsize' and 'responsesize' attributes and the values for their timestamps falls between t_1 and t_2. Following is the SPARQL query for this example:

```
1  select ?m where {
2    ?m @type message.
3    ?m @requestsize ?x.
4    ?m @responsesize ?y.
5    ?m @timestamp ?t.
6    FILTER (?x=?y && ?t > t1 && ?t < t2).
7  }
```

In this query, variable $?m$ represents messages in the SCM log. Variables $?x$, $?y$, and $?t$ represent the value of the attributes 'requestsize' (line 3), 'responsesize' (line 4), and 'timestamp' (line 5) respectively. Finally, the *filter* statement (line 6) restrict the result to those messages for which the filter expression evaluates to *true*.

3.4.2 Aggregation-level Queries

Standard SPARQL querying mechanism is not enough to support querying needs for analyzing BP execution data, based on the introduced data model in Section 3.3.1. In particular, SPARQL does not support folder and path nodes and querying them natively. In addition, querying the result of a previous query becomes complex and cumbersome, at best. Also path nodes are not first class objects in SPARQL [50, 181]. We extend SPARQL to support aggregation-level queries to satisfy specific querying needs of proposed data model. Aggregation-level queries in FPSPARQL include two special constructs: (a) *construct* queries: used for constructing folder and path nodes, and (b) *apply* queries: used to simplify applying queries to folder and path nodes.

Folder Node Construction

To construct a folder node (e.g., a partition in the process graph), we introduce the $fconstruct$ statement. This statement is used to group a set of related entities or folders. A basic folder node construction query looks like this:

```
fconstruct <Folder_Node Name>
[ select ?var1 ?var2 ... | (Folder_Node1, Folder_Node2, ...) ]
where {
  pattern1.
  pattern2.
  # Other Patterns
}
```

A query can be used to define a new folder node by listing folder node name and entity definitions in the *fconstruct* and *select* statements, respectively. Also a folder node can be defined to group a set of entities, folder nodes, and path nodes, e.g., see Example 6 in Section 3.5. A set of user defined attributes for this folder can be defined in the *where* statement. We instrument folder construction query with the *correlate* statement in order to apply a correlation condition on entity nodes (e.g., messages in service event logs) and correlated the entity nodes for which the condition evaluates to *true*. Here, we consider a correlation condition ψ as a predicate over the attributes of *events* for attesting whether two events belong to the same instance. For example, considering SCM log, one possible correlation condition is $\psi(m_x, m_y) : m_x.custid = m_y.custid$, where $\psi(m_x, m_y)$ is a binary predicate defined over the attribute *custid* of two messages m_x and m_y in the log. This predicate is *true* when m_x and m_y have the same value and *false* otherwise. A basic correlation condition query looks like this:

```
correlate {
   (entity1, entity2, edge1, condition)
   pattern1.
   pattern2.
   # Other Patterns
}
```

As a result, $entity_1$ will be correlated to $entity_2$ through a directed edge $edge_1$ if the *condition* evaluates to *true*. Patterns (e.g., $pattern_1$) can be used for specifying the edge attributes. Example 5 in Section 3.5 illustrates such a query.

Path Node Construction

We introduce the *pconstruct* statement to construct a path node. This statement can be used to: (i) discover transitive relationships between two entities, e.g., by using an existing graph reachability algorithm; or (ii) discover frequent pattern(s) between set of entities, e.g., by using an existing process mining algorithm. In both cases the result will be a set of paths which can be stored under a path node name. In general a basic path node construction query looks like this:

```
pconstruct <Path_Node Name>
( Start-Node,
  End-Node,
  Regular-Expression )
where {
  pattern1.
  pattern2.
  # Other Patterns
}
```

A regular expressions can be used to define a transitive relationship between two entities, i.e., starting node and ending node, or set of frequent patterns to be discovered. Attributes of starting node, ending node, and regular expressions alphabets (i.e., graph nodes and edges) can be defined in the *where* statement. The query applied to a folder, in Example 7 Section 3.5, illustrates such a query.

Folder Node Queries

We introduce the *apply* statement to retrieve information, i.e., by applying queries, from the underlying folder nodes. These queries can apply to one folder node or the composition of several folder nodes. Our model supports the standard set operations (union, intersect, and minus) to apply queries to the composition of several folder nodes. In general, a basic folder node query looks like this:

```
[Folder Node | (Composition of Folder Nodes)]
APPLY (
  [ <Entity-level Queries> |
    <Aggregation-level Queries> |
    <Existing process mining algorithms> ]
)
```

Entity-level queries or aggregation-level queries can be applied to folder nodes by listing folder node or composition of folder nodes before *apply* statement, and placing the query in parenthesis after *apply* statement. We also developed an interface to support applying existing process mining algorithms to folder and path nodes. Examples 7 and 8 in Section 3.5 illustrate such queries.

Path Analysis Queries

This type of query is used to retrieve information, i.e., by applying entity-level queries, from the underlying path node by using *apply* statement. Domain experts may use such queries in order to examine the discovered process instances. In general, a basic path node query looks like this:

```
Path_Node_Name APPLY (
 [ Entity-level Query ]
)
```

An entity-level query can be applied to a path node by listing path node name before *apply* statement, and placing the query in parenthesis after *apply* statement. Example 9 in Section 3.5 illustrates such a query.

3.5 Case Study

In this section we show how we use FPSPARQL query language to analyze process logs. We focus on the case study presented in Section 3.2.

3.5.1 Preprocessing of SCM Log

The aim of preprocessing of the log is to generate a graph by considering the set of messages in the log as nodes of the graph, and correlation between messages as edges between nodes. In order to preprocess the SCM log we performed the following two steps: (i) generating graph nodes: we extracted messages and their attributes from the log and formed a graph node for each message with no relations among nodes; and (ii) generating candidate correlations: we used the correlation condition discovery technique introduced in [252] to generate a set of candidate correlation conditions that could be used for examining the relationship between process events.

3.5.2 Partitioning of SCM Log

We use the candidate conditions, in preprocessing phase, to partition the log. Identifying the interestingness of a certain way of partitioning the logs or grouping the process events is "subjective", i.e., depends on the user perspective and the particular querying goal. To cater for interestingness, we enable users to choose the candidate conditions they are interested in to explore as a basis of relationships among events.

Example 5 . *Adam, an analyst, is interested in partitioning the SCM log into a set of related messages having the same customer ID. To achieve this he can construct a folder node (i.e., 'custID') and apply the correlation condition* $\psi(m_x, m_y) : m_x.custid = m_y.custid$*, where* $\psi(m_x, m_y)$ *is a binary predicate defined over the attribute custid of two messages* m_x *and* m_y *in the log, to the folder. As a result, related messages will be discovered and stored in the folder node 'custID'. Figure 3.8-A in Section 3.7 illustrates how our front-end tool enables users choosing the correlation condition(s) and generating FPSPARQL queries automatically. Following is the FPSPARQL query for this example.*

```
1   fconstruct custID as ?fn
2   select ?m_id, ?n_id
3   where {
```

```
4     ?fn @description 'custid=custid'.
5     ?m @isA entityNode.
6     ?m @type message.
7     ?m @id ?m_id.
8     ?m @custid ?x.
9     ?n @isA entityNode.
10    ?n @type message.
11    ?n @id ?n_id.
12    ?n @custid ?y.
13    correlate{
14      (?m,?n, ?edge, FILTER(?x=?y && ?n_id>?m_id))
15      ?edge @isA edge.
16      ?edge @label "custid".
17    }
18  }
```

In this query, the variable $?fn$ represent the folder node to be constructed, i.e., 'custID' (line 1). Variables $?m$ and $?n$ represent the messages in SCM log and $?m\_id$ and $?n\_id$ represent IDs of these messages respectively (lines 5 to 12). Variables $?x$ (line 8) and $?y$ (line 12) represent the values for 'm.custid' and 'n.custid' attributes. The condition $?x = ?y$ (line 14) applied to the log to group messages having same value for *custid* attribute. The condition $?n_{id} > ?m_{id}$ (line 14) makes sure that only the correlation between each message and the following messages in the log are considered. The *correlate* statement will connect messages, for which the condition $(?x = ?y \;\&\&\; ?n_{id} > ?m_{id})$ evaluates to *true*, with a directed labeled edge, i.e., variable $?edge$. The result will be stored in the folder 'custID' and can be used for further queries.

Example 6 . *Consider two folder nodes 'custID' and 'payID' each representing correlated messages based on correlation conditions '$\psi(m_x, m_y) : m_x.custid = m_y.custid$' and '$\psi(m_x, m_y) : m_x.payid = m_y.payid$' respectively. It is possible to construct a new folder (e.g., 'custID_payID') on top of these two folders in order to group them. The folder*

'custID_payID' will contain events related to customer orders that have been paid. Queries applied to the folder 'custID_payID' will be applied to all its subfolders. Example query is defined as follows.

```
1 fconstruct custID_payID as ?fn (custID,payID)
2 where {
3   ?fn @description 'set of ...'.
4 }
```

In this example the variable $?fn$ represents the folder node to be constructed, i.e., 'custID_payID'. This folder node contains two sub-folders and has a user defined attribute *description*. Sub-folders are hierarchically organized by *part-of* edge, i.e., an implicit relationship.

3.5.3 Discovering Process Models

In this phase a query can be applied to previously constructed partitions to discover process models. As mentioned earlier, a folder node, as a result of a correlation condition, partitions a subset of the events in the log into instances of a process. The process model, which the instances inside a folder represent, can be discovered using one of the many existing algorithms for process mining [8, 7, 251].

It is possible that some folders contain a set of related process events (e.g., the set of orders for a given customer), but not process instances. It is possible for the analyst to apply a regular expression based query on the events in a folder. The regular expression may define a relationship that is not captured by any candidate correlation conditions. Applying such queries to a folder node may result in forming a set of paths which can be then stored in a path node. The constructed path node can be examined by the analysts and may considered as a set of process instances. Example queries are defined as follows.

Example 7 . *Adam is interested in discovering patterns, in 'custID' partition (see Example 5), between specific orders (i.e., messages with IDs '3958' and '4042') which contains*

product confirmation. He can codify his knowledge into a regular expressions that describe paths through sequence of messages in the partition. The path query can be applied on the partition and the result can be stored in a path node (i.e., 'OrderDiscovery'). As a result, one path (i.e., a process model) discovered. Figure 3.8-B in Section 3.7 illustrates the visualized result of this example generated by the front-end tool. Following is the FPSPARQL query for this example.

```
1   (custID)
2   apply(
3    pconstruct OrderDiscovery
4    (?startNode,?endNode,(?e ?n)* e ?msg e (?n ?e)*)
5    where {
6     ?startNode @isA entityNode.
7     ?startNode @type message.
8     ?startNode @id '3958'.
9     ?endNode @isA entityNode.
10    ?endNode @type message.
11    ?endNode @id '4042'.
12    ?n @isA entityNode.
13    ?e @isA edge.
14    ?msg @isA entityNode.
15    ?msg @type message.
16    ?msg @operation 'OrderFulfil'.
17   }
18  )
```

In this example a path construction query, i.e., *pconstruct* query, applied to the folder 'custID'. Variables ?$startNode$ and ?$endNode$ denote messages $m_{id=3958}$ and $m_{id=4042}$ respectively (lines 6 to 11). Variables ?n (line 12) and ?e (line 13) denote any edge or node in the transitive relationship between $m_{id=3958}$ and $m_{id=4042}$. Finally, variable ?msg (lines 14 to 16) denotes a message having the value 'OrderFulfil' for the attribute *operation*. In the regular expression (line 4), parentheses are used to

define the scope and precedence of the operators and the asterisk indicates there are zero or more of the preceding element.

Example 8 . *Adam is interested in discovering frequent patterns between correlated messages in 'custID' partition (see Example 5), to analyze the process of producing a product. He can codify his knowledge into regular expressions that describe paths having the pattern "*start *with a message having the value* produce *for the attribute* operation*, which followed by a message having the value 'confirmProduction' for the attribute* operation*, and* end *with a message having the value* pay *for the attribute* operation*". The path query can be applied on the partition and the result can be stored in a path node (i.e., 'ProductDiscovery'). As the result set of this query, 12 paths discovered. Unlike Example 7, these paths have different starting and ending nodes. Following is the FPSPARQL query for this example.*

```
1  (custID)
2  apply(
3   pconstruct ProductDiscovery
4   (?startNode, ?endNode, e ?msg e)
5   where {
6    ?startNode @isA entityNode.
7    ?startNode @type message.
8    ?startNode @operation 'Produce'.
9    ?endNode @isA entityNode.
10   ?endNode @type message.
11   ?endNode @operation 'Pay'.
12   ?e @isA edge.
13   ?msg @isA entityNode.
14   ?msg @type message.
15   ?msg @operation 'ConfirmProduction'.
16  }
17 )
```

In this example a *pconstruct* query is applied to the folder 'custID', in which

?*startNode* and ?*endNode* denote set of starting and ending nodes (lines 6 to 11). Variable ?*e* (line 12) denotes any edges in the regular expression pattern and variable ?*msg* (lines 13 to 15) denotes a message having the value 'ConfirmProduction' for the attribute *operation*. A frequent sequence mining algorithm, developed based on a process mining method, is used to generate frequent pattern(s) based on the specified regular expression (line 4), i.e., "?startNode → e → ?msg → e → ?endNode".

Example 9 . *Consider the path node 'OrderDiscovery' constructed in Example 7. We are interested to find messages in this path node, that have the keyword 'Retailer' in their binding attributes. Following is the FPSPARQL query for this example.*

```
1  (OrderDiscovery)
2  apply ( select ?m_id
3   where {
4    ?m @isA entityNode. ?m @type message.
5    ?m @binding ?b.
6    Filter regex(?b,"Retailer").
7   }
8  )
```

In this example, variable ?*m_id* (line 3) denotes message *ID*s that fall inside 'OrderDiscovery' path node. The query "retrieve the messages that have the keyword *Retailer* in their *binding* attributes" will apply on this path node.

3.6 Architecture and Implementation: FPSPARQL

3.6.1 FPSPARQL Architecture

We have developed a software prototype for organizing, indexing, and querying ad-hoc process data. As mentioned earlier, we model process logs as a graph. In order to analyze process graphs, we propose a graph processing engine, i.e., FPSPARQL [53, 52, 51] (a Folder-, Path-enabled extension of the SPARQL), to manipulate and query

entities, and folder and path nodes. The query engine is implemented in Java and supports two types of storage back-end:

- Relational Database System: The simplest way to store a set of RDF statements is to use a relational database with a single table that includes columns for subject, property and object. While simple, this schema quickly hits scalability limitations [292]. To avoid this we developed a relational RDF store including its three classification approaches [292]: vertical (triple), property (n-ary), and horizontal (binary). The query engine supports various relational database management systems (e.g., IBM DB2, PostgreSQL, and Microsoft SQL Server) to generate physical storage layer.

- Hadoop File System: We use Hadoop [330], an open source software framework that supports data-intensive distributed applications, data processing platforms to store and retrieve process graphs in Hadoop file system and to support cost-effective and Web-scale processing of process graphs. We use Apache-Pig[2], a high-level procedural language on top of Hadoop for querying large process graphs.

Figure 3.4 illustrates FPSPARQL graph processing architecture which consists of the following components:

- *Data Mapping Layer*: This layer is responsible for creating data element mappings between semantic web technology (i.e., Resource Description Framework) and physical storage layer, i.e., relational database schema and Hadoop File System. In order to generate the physical layer, we developed a workload-independent physical design *loader algorithm*.

- *Loader algorithm*: Input graph can be in the form of RDF, N3 (or Notation3, is a W3C standard and shorthand non-XML serialization of RDF models), or XML. Notice that, in the RDF data model, graph edges can not be described by attributes. In addition, graph edges are used to represent both of the attribute/literal value pairs of the nodes and the structural relationship with

[2]http://pig.apache.org/

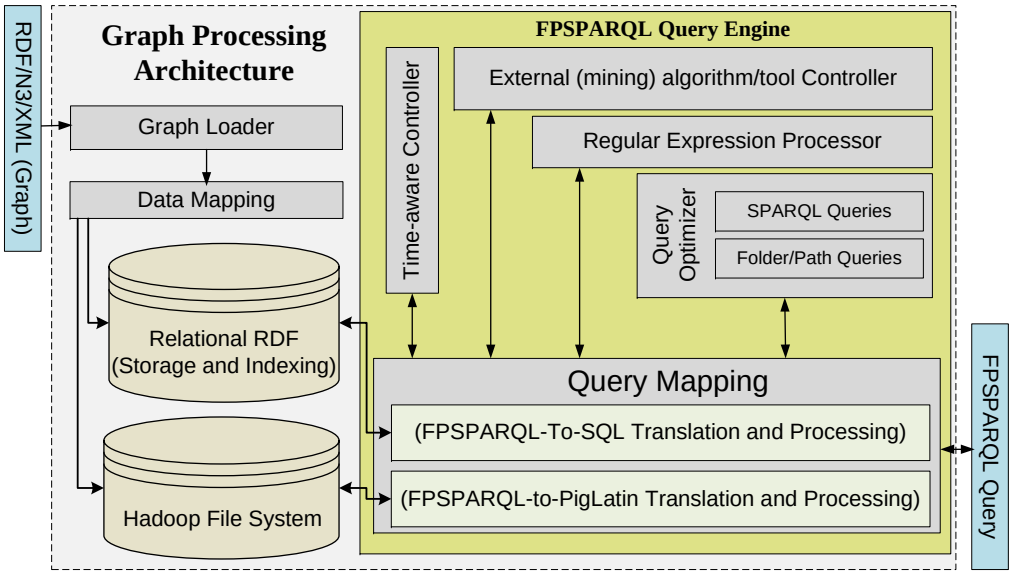


Figure 3.4: FPSPARQL graph processing architecture.

other nodes in the graph with no differentiation. Such uniform treatment for attributes and edges in the graph data will increase the size of graph topology. To avoid this, we prepared an XML schema to support attributed graphs where both nodes and edges of the graph may include rich semantic information. We developed a workload-independent physical design by developing a *loader* algorithm. This algorithm is responsible for: (i) validating the input graph, i.e., RDF, N3, and XML format; (ii) generating the relational/Hadoop representation of triple store, for manipulating and querying entities, folders, and paths; and (iii) generating powerful indexing mechanisms for relational database systems.

- *Time-aware Controller*: RDF databases are not static and changes may apply to graph entities (i.e., nodes, edges, and folder/path nodes) over time. Time-aware controller is responsible for data changes, data manipulation, and incremental graph loading.

- *Query Mapping Layer*: This layer is responsible for FPSPARQL queries translation and processing. We use the relational representation of triple RDF store, to store, manipulate, and query folder nodes and path nodes. To describe constraints on the path nodes, we reused expressions proposed in [23]. This layer contains two components:

- *SPARQL-to-SQL Translation Algorithm*: We implemented a schema-independent SPARQL-to-SQL translation algorithm based on the proposed relational algebra for SPARQL and semantics preserving SPARQL-to-SQL query translation proposed in [103, 91]. This algorithm supports *Aggregate* queries and *Keyword Search* queries. Figure 3.7 shows a SPARQL query, its translation into a relational operator tree, and its equivalent SQL query which is generated by this algorithm.
- *FPSPARQL-to-PigLatin translation algorithm*: In order to translate FPSPARQL queries into Pig-Latin we follow a specific format in which data is read from the Hadoop file system, a number of operations (e.g., LOAD, SPLIT, JOIN, FILTER, GROUP, and STORE) are performed on the data, and then the resulting relation is written back to the file system. We used the techniques proposed in [297], for mapping SPARQL queries to Pig Latin program and consequently to generate MapReduce jobs. Figure 3.5 illustrates the modular translation process for this mapping. In particular, SPARQL graph pattern matching is dominated by join operations, and is unlikely to be efficiently processed. We use existing query optimization techniques [206, 188, 281] to generate the optimal query plan by reinterpreting certain join tree structures as grouping operations, i.e., to enable a greater degree of parallelism in join processing.

- *Regular Expression Processor*: To describe constraints on the path nodes, we reused the specification for regular expressions and filter expressions proposed in [23, 50]. We developed a regular expression processor which supports optional elements (?), loops (+,*), alternation (|), and grouping ((...)).

- *External Algorithm/Tool Controller*: Is responsible to support applying external graph reachability/mining algorithms to the process graph. We developed an interface to support various graph reachability algorithms [19] such as Transitive Closure, GRIPP, Tree Cover, Chain Cover, Path-Tree Cover, and Shortest-Paths [158]. Please see Section 3.6.3 for more details.

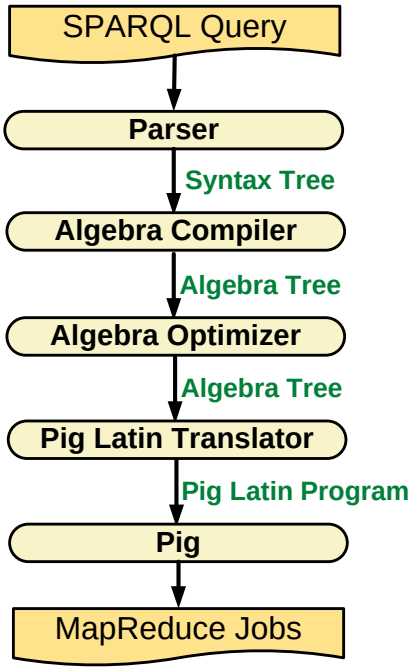


Figure 3.5: Modular translation process for mapping SPARQL to Pig Latin.

- *Query Optimizer*: We leveraged the techniques proposed in [206, 188, 281] to optimize the execution of queries in Hadoop platform. Moreover, to optimize the performance of queries in relational databases, we developed four optimization techniques proposed in [90, 292, 91]: (i) selection of queries with specified varying degrees of structure and spanning keyword queries; (ii) selection of the smallest table to query based on the type information of an instance; (iii) elimination of redundancies in basic graph pattern based on the semantics of the patterns and database schema; and (iv) create separate tables (property tables) for subjects that tend to have common properties to reduce the self-join problem.

3.6.2 Physical Storage Layer

In this section we describe the techniques used to store process graphs in both relational database systems and Hadoop file system.

Relational Database System

The Resource Description Framework (RDF) represents a special kind of attributed graphs: in the RDF data model, graph edges can not be described by attributes. We model graphs based on a RDF data representation, where we support the uniform representation of nodes and edges in the graph. Figure 3.6 illustrates the physical layer for storing the sample graph presented in Figure 3.3, where four types of objects can be stored: nodes, edges, folder-nodes, and path-nodes. Each object has the following mandatory attributes: ID (a unique identifier), class (can be set to entity-node, edge, folder, and path node), type (each node may conform to an entity type, e.g., author and paper), and label. Each object may have other user descriptive attributes (Figure 3.6-B), where these attributes can be defined in objects properties.

As illustrated in Figure 3.6-B, we use triplestores to store each attribute. Attributes may conform to an entity type, e.g., 'author:name' which means the attribute 'name' of type 'author'. We use the '@' symbol for representing attribute edges (e.g., the triple 'a1,@author:name,Boualem' in Figure 3.6-B) and distinguishing them from the relationship edges (e.g., the triple 'a3,e1,p2' in Figure 3.6-C) between graph nodes. To store links between graph entities (i.e., nodes, folder-nodes, and path-nodes), we define triplestores for each edge type. For example, considering Figure 3.6-C, all the links typed as 'author-of' are stored in a separate triplestore. The link itself defined as an object and will be stored separately (Figure 3.6-A).

In folder nodes, entities and relationships among them will be stored in the folder-store (Figure 3.6-A). Folder node, as a graph entity, will have set of attributes. These attributes will be stored in folder-nodes-properties triplestores (see Figure 3.6-B). For example, for the query in Example 2, we add a folder node to the original graph, and store the result of the query in the folder-store. Properties of this folder will be stored in folder-nodes-properties triplestores.

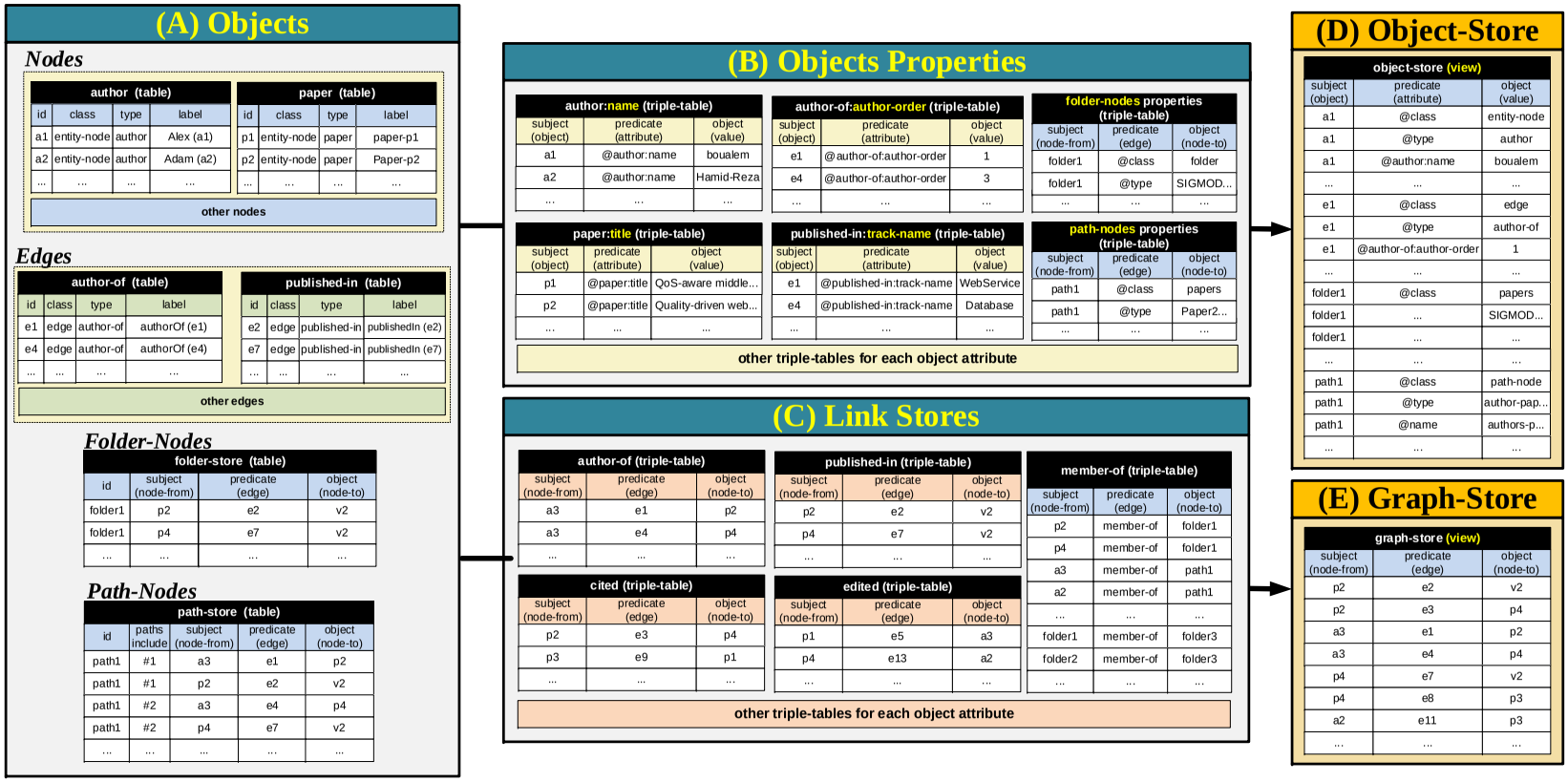


Figure 3.6: Physical layer for storing the sample graph represented in Figure 3.3 including: (A) object stores for storing nodes, edges, folder nodes, and path nodes; (B) object property store for storing objects attributes in triplestore format; (C) link stores for storing relationships among entities; (D) entity store as a view over object stores; and (E) graph store as a view over link stores.

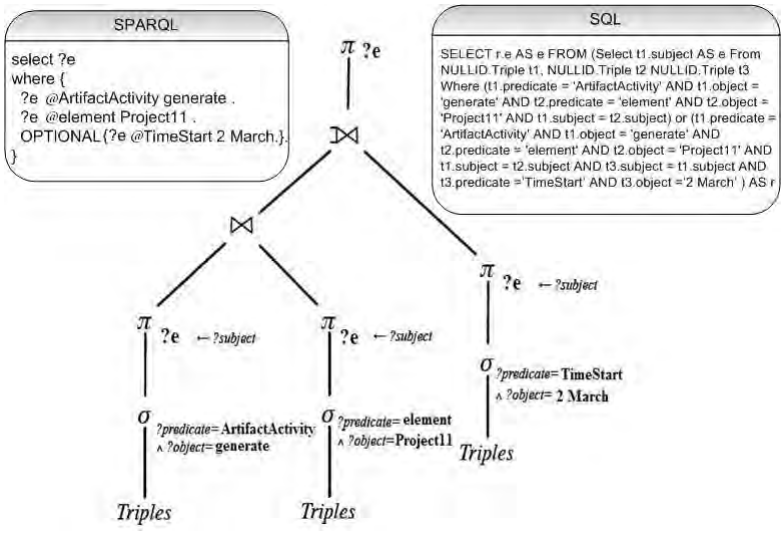


Figure 3.7: A SPARQL query, its translation into a relational operator tree, and its equivalent SQL query generated by our translation algorithm.

In path nodes, set of related paths will be stored in the path-store (Figure 3.6-A). Path node, as a graph entity, will have set of attributes. These attributes will be stored in path-nodes-properties triplestores (see Figure 3.6-B). For example, for the query in Example 3, we add a path node to the original graph, and store the result of the query in the path-store. In the path-store, each path (in a path node) will have a unique identifier, e.g., values '#1' and '#2' in the column 'paths-include' of the table path-store (Figure 3.6-A). Properties of this path node will be stored in path-nodes-properties triplestore.

Finally, we create two views (i.e., object-store and graph-store) over the above explained physical layer. Object-store (Figure 3.6-D) will include all the objects (i.e., nodes, edges, folder-nodes, and path-nodes) in the graph. Graph-store (Figure 3.6-E) will include all the links among graph entities (i.e., nodes, folder-nodes, and path-nodes).

Hadoop File System

We used the techniques proposed in [187] to store and retrieve large number of RDF triples in Hadoop file system. In particular, the graph will be divided into multiple files: (i) predicate split, the graph data will be divided according to the predicates; and (ii) predicate object split, the RDF type file is first divided into as many files

as the number of distinct objects predicates have. For each distinct object values of the predicates, a file will be generated. Notice that, the literals remain in the file named by the predicate: no further processing is required for them. Finally, objects will be moved into their respective file named as predicate type.

3.6.3 FPSPARQL Implementation

To address the above challenges, we have developed a software prototype for organizing, indexing, and querying ad-hoc process data. The query engine is implemented in Java and supports two types of storage back-end: Relational Database System and Hadoop File System. Moreover, a front-end tool prepared to assist users in four steps:

Step1: [**Preprocessing**] We have developed a workload-independent algorithm for: (i) processing and loading a log file into an RDBMS/Hadoop for manipulating and querying entities, folders, and paths; (ii) generating powerful indexing mechanisms for relational databases. We also provided inverted indexes [344] on folder store in order to increase the performance of queries applied on folders. For the Hadoop file system, we used existing query optimization techniques [206, 188, 281] to generate the optimal query plan.

Step2: [**Partitioning**] We provide users with a list of interesting correlation conditions based on the algorithm for discovering interesting conditions in [252]. Users may choose these correlation conditions to partition a log. In order to generate *folder construction queries*, we provide users with an interface (Figure 3.8-A) to choose the correlation condition(s) and generate FPSPARQL queries automatically.

Step3: [**Mining**] We provide users with templates to generate regular expressions and use them in path queries. We developed a regular expression processor. Moreover, an interface has been implemented to support various graph reachability algorithms [19] such as Transitive Closure, GRIPP, Tree Cover, Chain Cover, Path-Tree Cover, and Shortest-Paths [158]. In general, there are two types of graph

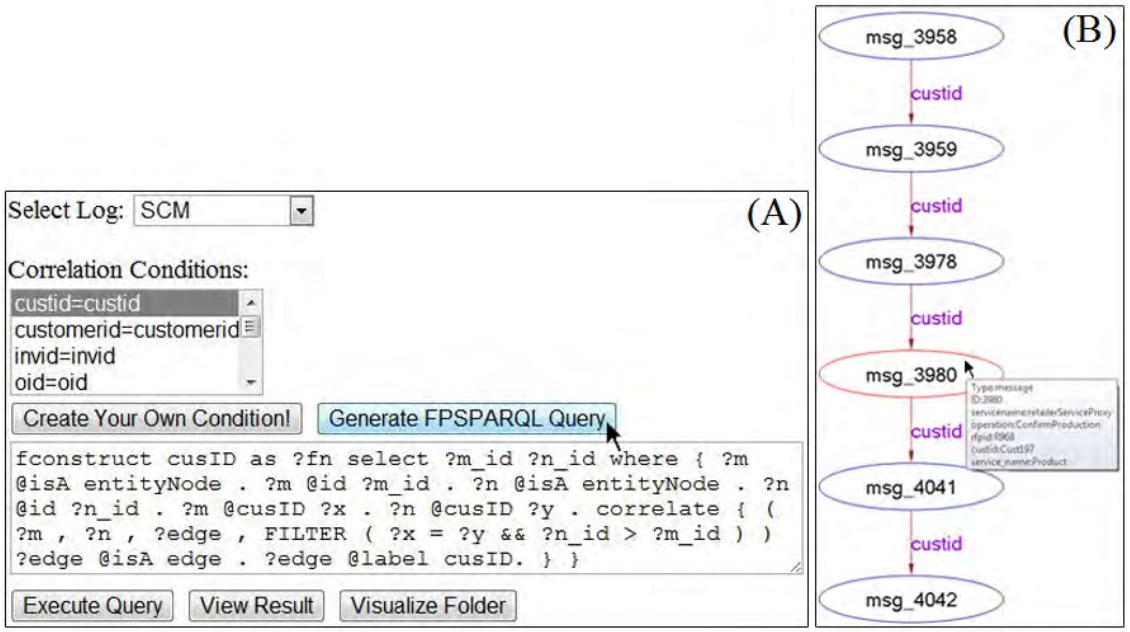


Figure 3.8: Screenshots of FPSPARQL GUI: (A) The query generation interface in FPSPARQL, and (B) The discovered process model for the query result in Example 7.

reachability algorithms [19]: (i) algorithms traversing from starting vertex to ending vertex using breadth-first or depth-first search over the graph, and (ii) algorithms checking whether the connection between two nodes exists in the edge transitive closure of the graph. Considering $G = (V, E)$ as directed graph that has n nodes and m edges, the first approach incurs high cost as $O(n+m)$ time which requires too much time in querying. The second approach results in high storage consumption in $O(n^2)$ which requires too much space. In the experiment, we used the GRIPP [318] algorithm which has the querying time complexity of $O(m-n)$, index construction time complexity of $O(n+m)$, and index size complexity of $O(n+m)$.

Step4: [Visualizing] We provided users with a graph visualization tool for the exploration of results, e.g., see Figure 3.8-B which illustrates the discovered process model for the query result in Example 3. Users are able to view folders, paths, and the result of queries in a list and visualized format. This way, event relationships and candidate process instances can be examined by the analyst.

3.7 Experiments

3.7.1 Datasets

We carried out experiments on three datasets: SCM, Robostrike, and PurchaseNode.

SCM

This dataset has been introduced in the case study, Section 3.2. In this dataset, the interaction log of Web services with clients was collected using a real-world commercial logging system for Web services, i.e., HP SOA Manager[3]. The services in SCM scenario are implemented in Java and use Apache Axis as SOAP implementation engine and Apache Tomcat as Web application server. Table 3.2 shows the characteristics of this dataset.

Robostrike

This log is the interaction log of a multi-player on-line game service called Robostrike[4]. In this game, clients (players) exchange XML messages with the game service performing various operations, e.g., designing new games and playing. Each session of a player may include several game plays or game creations. The log contains 40,000 messages (Table 3.2), which correspond to one day of activities of the game service.

PurchaseNode

This process log was produced by a workflow management system supporting a purchase order management service called PurchaseNode (PN). The PN dataset contains 34,803 tuples corresponding to task executions within workflow instances (Table 3.2). It is private process log of a service in which all messages are correlated using atomic conditions. This dataset was originally organized into two

[3]http://managementsoftware.hp.com/products/soa
[4]http://www.robostrike.com

Table 3.2: Characteristics of the proposed datasets.

Dataset	*SCM*	*Robostrike*	*PurchaseNode*
service operations	14	32	26
messages in log	4,050	40,000	34,803
attributes	28	98	26

tables: one for the workflow definitions and the other for the workflow instances (execution data).

3.7.2 Evaluation

We evaluated the performance and the query results quality using SCM, Robostrike, and PurchaseNode process logs. Also in the Appendix A, we evaluated the performance of the FPSPARQL query engine compared to one of the well-known graph databases, the HyperGraphDB [190], which shows the good performance of the FPSPARQL query engine.

Performance. The performance of FPSPARQL queries assessed using *query execution time* metric. The preprocessing step took 3.8 minutes for the SCM log, 11.2 minutes for the Robostrike log, and 9.7 minutes for the PurchaseNode log. For the partitioning step, we constructed 10 folders for each process log (i.e., SCM, Robostrike, and PurchaseNode). Constructed folders, i.e., partitions, selected according to provided list of interesting correlation conditions by the tool. Figures 3.9-A, -B, and -C show the average execution time for constructing selected folders for each process log.

For the mining step, we applied path node construction queries on each constructed folder. These path queries generated by domain experts who were familiar with the process models of proposed process logs. For each folder we applied one path query. As the result, the set of paths for each query were discovered and stored in path nodes. Figures 3.9-D, -E, and -F show the average execution time for applying constructed path queries on the folders for each log. We ran these experiments for different sizes of process logs.

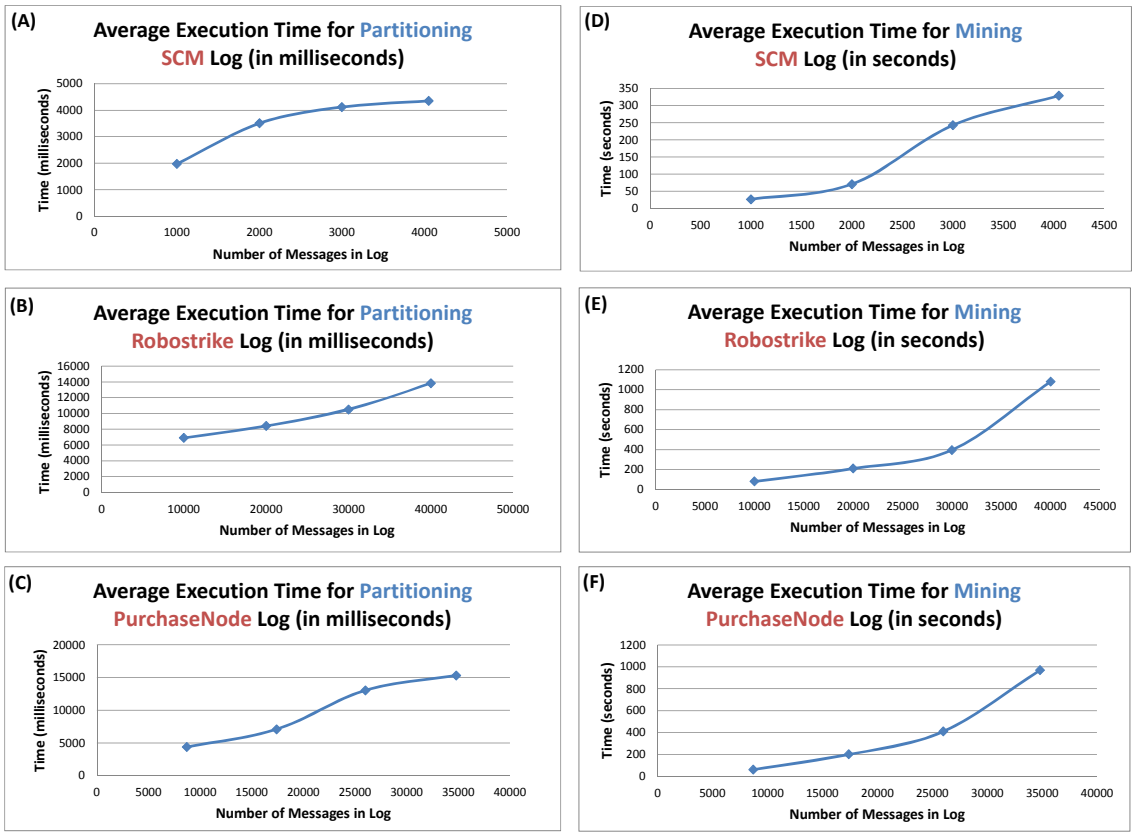


Figure 3.9: The performance evaluation results of the approach on three datasets, illustrating: (i) the average execution time for partitioning: (A) SCM log, (B) Robostrike log, and (C) PurchaseNode log; and (ii) the average execution time for mining: (D) SCM log, (E) Robostrike log, and (F) PurchaseNode log;

Quality. The quality of the results is assessed using classical *precision* metric which defined as the percentage of discovered results that are actually interesting. For evaluating the interestingness of the result, we ask domain experts who have the most accurate knowledge about the dataset and the related process to: (i) codify their knowledge into regular expressions that describe paths through the nodes and edges in the folders; and (ii) analyze discovered paths and identify what they consider relevant and interesting from a business perspective. The quality evaluation applied on SCM log. Five folders constructed and three path queries applied to each folder. As a result 31 paths discovered, examined by domain experts, and 29 paths (precision=93%) considered relevant.

Performance Comparison Between RDBMS and Hadoop Execution Plans. As mentioned earlier, FPSPARQL queries can be run on two types of storage back-end: RDBMS and Hadoop. In this part we compare the performance of query plans on relational triplestores and Hadoop file system. All experiments in this part were conducted on a virtual machine, having 32 cores and 192GB RAM. Figure 3.10 illustrates the performance analysis between RDBMS and Hadoop for queries (average execution time) in Figure 3.9-A (partitioning) and -D (mining) applied to SCM dataset. Figure 3.10 shows a linear scalability between the response time of FPSPARQL queries applied to Hadoop file system and the number of events in the log. The evaluation shows that, Hadoop platform can handle addition of large number of nodes and edges, different size of the datasets, without affecting its performance significantly.

Discussion. We evaluated our approach using different types of process event logs, i.e. PurchaseNode (a single-process log), SCM (a multi-service interaction log), and Robostrike (a complex logic of a real-world). The evaluation shows that the approach is performing well. (also in the Appendix A, we have evaluated the performance of the FPSPARQL query engine compared to HyperGraphDB [190]). As illustrated in Figure 3.9 we divided each log into regular number of messages (we sampled the graphs carefully to guarantee the properties of the samples) and ran the experiment for different sizes of process logs. The evaluation shows a polynomial (nearly linear) increase in the execution time of the queries in respect with the dataset size. Based on the lesson learned, we believe the quality of discovered paths is highly related to the regular expressions generated to find patterns in the log, i.e., generating regular expressions by domain experts will guarantee the quality of discovered patterns.

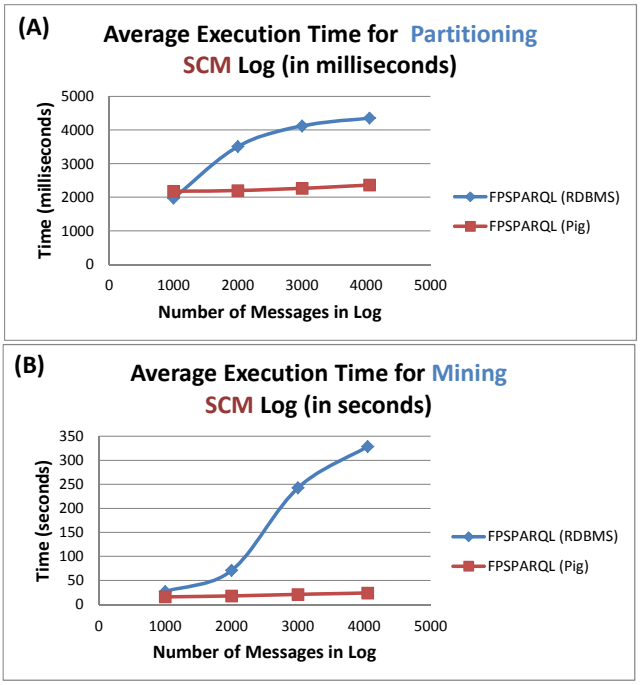


Figure 3.10: The evaluation results, illustrating the performance analysis between RDBMS and Hadoop applied to SCM dataset: (A) the average execution time for partitioning SCM log; and (B) the average execution time for mining SCM log.

3.8 Related Work

3.8.1 NoSQL Databases

Relational database management systems (RDBMSs) have repeatedly shown that they are very efficient and scalable. However, ever-increasing needs for scalability and new advances in Web technology, which requires facilitating the implementation of applications as a distributed and scalable services, have created new challenges for RDBMSs [308, 218, 84, 326]. For example, database requirements for Web, enterprise, and cloud computing applications may vary as storage data consistency is not necessary for all applications. NoSQL (Not only SQL) is a broad class of low-cost and high performance database management systems and proposed to address these challenges.

Specific characteristics of NoSQL systems, such as horizontal scalability (i.e., the ability to distribute both the data and the load of operations over many servers, with no RAM or disk shared among the servers), weak consistency model, distributed

indexes and semi-structured data schema, enables working with a huge quantity of data where availability, scalability, elasticity, load balancing, fault tolerance, and the ability to run in a heteroecious environment are critical system goals [308, 326].

In the literature, the term NoSQL has been used for any database system that is not relational, such as: (i) graph database systems: is a database management system which uses graph structures with nodes, edges, and properties to represent and store data, e.g., Neo4j[5] and OrientDB[6]; (ii) object-oriented database systems: is a database management system in which information is represented in the form of objects as used in object-oriented programming, e.g., Versant[7] and db4objects[8] (by Versant); (iii) distributed object-oriented stores: is a database management system and very similar to object-oriented DBMSs which use distribute objects in-memory and on multiple servers, e.g., GemFire[9]; (iv) document stores: is a document-oriented database in which various techniques (e.g., collections, tags, non-visible metadata, and directory hierarchies) are used for organizing and/or grouping different types of documents, e.g., they all assume documents encapsulate and encode data (or information) in some standard formats or encodings such as XML, YAML, JSON, and BSON, as well as binary forms like PDF and Microsoft Office documents. Examples are Lotus Notes[10] and IBM Lotus Domino[11]; and (v) key-value stores, is similar to file systems which enables applications to store their data in a schema-less way, e.g., data could be stored in a data-type of a programming language or an object. Examples are Apache Cassandra[12] and Dynamo[13].

In particular, the main NoSQL systems are: (i) Google Bigtable [87]: is a scalable and distributed storage system used for Google products such as Google Docs[14],

[5] http://neo4j.org/
[6] http://www.orientdb.org/
[7] http://www.versant.com/
[8] http://www.db4o.com/
[9] http://www.vmware.com/products/application-platform/vfabric-gemfire/overview.html
[10] http://www-01.ibm.com/software/lotus/products/notes/
[11] http://www-01.ibm.com/software/lotus/products/domino/
[12] http://cassandra.apache.org/
[13] http://aws.amazon.com/dynamodb/
[14] https://drive.google.com/

Earth[15], Finance[16], and search engine[17]. Bigtable data model, is a simple data model with dynamic control and semi-structured schema, i.e., supports changing data layout/format without being restricted by data schemas; (ii) Yahoo PNUTS [101]: is a scalable database system, consists of multiple geographically distributed regions, for storing tables (that are horizontally partitioned and scattered across many servers) of attributed records to support Yahoo applications; and (iii) Amazon Dynamo [107]: is a scalable and very reliable database system, consists of thousands of servers geographically distributed all over the world.

Another line of work used MapReduce [106] for processing huge amounts of unstructured data in a massively parallel way. Hadoop [330], the open source implementation of MapReduce, provides the distributed file system (HDFS[18]) and Pig[19], a high level language for data analysis. A new stream of work [199, 98, 228, 187, 297] used Hadoop for large scale graph storage and mining. Many works [206, 188, 281] focused on query optimization techniques for the Pig to store triples and querying graphs. In addition to Pig, there are several high-level language and environments for advanced MapReduce-like systems, including SCOPE [85], Sawzall [273], and Sphere [154].

3.8.2 RDF/SPARQL

Graphs are essential modeling and analytical objects for representing information networks. Several graph querying techniques, e.g., pattern match query [348, 352], reachability query [97, 196], shortest path query [94, 328] and subgraph search [346, 347], have been proposed for querying and analyzing graphs. These methods rely on constructing some indices to prune the search space of each vertex to reduce the whole search space.

A recent book [19] and survey [29] discuss a number of data models and query languages for graph data. Many of these models use RDF [235] (Resource Descrip-

[15]http://www.google.com/earth/
[16]http://www.google.com/finance
[17]http://www.google.com/
[18]http://hadoop.apache.org/
[19]http://pig.apache.org/

tion Framework), an official W3C recommendation for semantic Web data models, to model graphs and use SPARQL, an official W3C recommendation for querying RDF graphs [276]. In particular, SPARQL queries are pattern matching queries on triples that constitute an RDF data graph, where RDF is a data model for schema-free structured information.

Several research efforts have been proposed to address efficient and scalable management of RDF data [292, 14, 232]. Many of existing RDF systems have mapped RDF triples onto relational tables, such as RDFSuite [22], Sesame [73], Jena [332], and Oracle's RDF match implementation [95]. In RDF, the schema may evolve over the time which fits well with the modern notion of data management, dataspaces [166], and its pay-as-you-go philosophy [194].

SPARQL [276] is a declarative query language, an W3C standard, for querying and extracting information from directed labeled RDF graphs. SPARQL supports queries consisting of triple patterns, conjunctions, disjunctions, and other optional patterns. However, there is no support for querying grouped entities. Paths are not first class objects in SPARQL [276, 181]. PSPARQL [31] extends SPARQL with regular expressions patterns allowing path queries. SPARQLeR [209] is an extension of SPARQL designed for finding semantic associations (and path patterns) in RDF bases. In FPSPARQL [53, 52, 51], we support folder and path nodes as first class entities that can be defined at several levels of abstractions and queried. In addition, we provide an efficient implementation of a query engine that support their querying.

Some of existing approaches for querying and modeling graphs [123, 138] focused on defining constraints on nodes and edges simultaneously on the entire object of interest, not in an iterative one-node-at-a-time manner. These approaches do not support querying nodes at higher levels of abstraction. Authors of [138] propose an *Information Fragment* as an abstraction for representing a subgraph. They do not support querying information fragments.

BiQL [123] is an SQL-based graph query language focused on the uniform treatment of nodes and edges and supports queries that return subgraphs. BiQL supports a closure property on the result of its queries meaning that the output of every query can be used for further querying. Compared to BiQL, in our work folders and paths

are first class abstractions (graph nodes) and can be defined in a hierarchical manner, over which queries are supported. HyperGraphDB[20] is a graph database based on hypergraphs (a hypergraph node is connected through an edge to all vertices that are contained in it). There is no query language for HyperGraphDB and querying is performed through special purpose APIs. HyperGraphDB builds on two prior approaches of Hypernode [223] and GROOVY [222] graph representation models which focus on representing objects and object schemas. GROOVY and Hypernode only support typed objects, and have no support for hypernode specific operations.

3.8.3 Querying Process Models and Instances

In recent years, querying techniques for BPs received high interest in the research community. Some of existing approaches for querying BPs [33, 47, 130, 134, 293] focused on querying the definitions of BP models. They assume the existence of enterprise repository of business process models. They provide business analysts with a visual interface to search for certain patterns and analyze and reuse BPs that might have been developed by others. These query languages are based on graph matching techniques. BP-QL [47] and [130] are designed to query business processes expressed in BPEL. BPMN-Q [33, 293] and VQL [134] are oriented to query generic process modeling concepts.

A recent book [2] and a survey [8] discuss the entire process mining spectrum from process discovery to operational support. Moreover, ProM [7] offers a wide range of tools related to process mining and process analysis. The main concern of these approaches is to reverse engineer the definitions of business process models from execution logs of information system components. Moreover, depending on how much details the log gives, they can provide statistics about many aspects of the business processes such as: the average duration of process instances or average resource consumptions. Our process querying approach enables querying and exploring the relationships and correlations among process events, and various ways to form process instances, and therefore it is complementary to process mining tools.

[20] http://www.hypergraphdb.org/

Querying running instances of business processes represents another flavor of querying [48, 246, 274]. It can be considered as a tool in the hand of administrator of a business process enactment engine to monitor the status of running processes and trace the progress of execution. Such queries can be used to discover many problems such as: detecting the occurrence of deadlock situations or recognizing unbalanced load on resources. PQL [246] is an SQL-based query language with a main focus on querying biological pathways. BP-Mon monitoring query [48] is represented as a BPEL process that capture interesting events of monitored processes. For instance, a monitoring query could listen to bid cancelation events of a bidding process. A manager could be interested in tracking users who make cancelations too often. In [274] Pistore et al. presented an approach based on model checking to monitor the composition as well as the execution of web services. Starting with an abstract BPEL specification, they depend on model checking approach to find a composition that reaches the goals as well as synthesizing a monitoring component that checks whether the external services behave as specified in the protocol. The main limitation of these approaches is that they assume an ideal world in which: (i) the business process models are pre-defined and available; and (ii) the execution of the business processes is achieved through a business process management system (e.g., BPEL) where the execution traces should comply with the defined process models.

In our approach, understanding the processes, in the enterprise, and their execution through exploration and querying event logs is a major goal. The focus is also on scenarios, which is often the case in today's environments, where processes are implemented over IT systems, and there is: (i) no up-to-date documentation of process definition and it's execution; and/or (ii) no data on the correlation rules for process events into process instances. Proposed query language, FPSPARQL, provides an explorative approach which enables users to correlated entities and understand which process instances are interesting. Moreover, we support for executing external process mining algorithms on the graph and partitions, folders, to facilitate the analysis of BPs execution.

3.8.4 Enterprise Search

Enterprise search is the practice of making content from multiple enterprise-type sources [173], e.g., electronic forms, external/internal Website, email, databases, and documents. Enterprise search systems, e.g., Google enterprise solutions[21], IBM OmniFind solution for enterprise search[22], Panoptic enterprise search engine[23], and Verity enterprise search solution[24], index (structured and unstructured) data and documents from a variety of sources such as: file systems, intranets, document management systems, and databases in order to describe the software of search information within an enterprise [173, 255].

Enterprise search differs from Internet search in many ways [173, 255, 72, 37, 110]: (i) The result of an enterprise search should be the exact right answer, but on the Internet, users are looking for the most relevant results, i.e., a large number of documents are typically relevant to an Internet search query; (ii) Stakeholders in the Internet are known/unknown authors who are free to publish any content, but in an intranet, stakeholders are known and they follow the entities that the organization serves; and (iii) Internet search engines are fully controlled and managed by one organization as a service, but in enterprises, the search software is licensed to and deployed by a variety of organizations in diverse environments.

In an enterprise search systems, content goes through various phases [143, 173, 255]: (i) Content ingestion, or content collection, is usually either a polling (i.e., periodic data pull at a predefined interval), streaming (i.e., an open call that allows data to be pushed to the caller), or publish-subscribe (i.e., this involves registering to a hub that actively sends data only when new content is available) model; (ii) Content processing and analysis, includes processing and normalizing the incoming documents, which may have many different formats or types; (iii) Indexing, will be applied to the resulting text for quick lookups; and (iv) Query parsing and matching, various querying filters should be provided for intranet search (e.g., to locate company policies, financial information, client histories, and online services),

[21] http://www.google.com/enterprise/
[22] http://www-306.ibm.com/software/data/integration/db2ii/editions_womnifind.html
[23] http://www.panopticsearch.com
[24] http://www.verity.com/products/search/enterprise_web_search/index.html.

internal multi-source search (to integrate various sources related to a specific query, e.g., some data are in official database, some in emails, and some in spreadsheets), and even forensic search (to answer questions of interest to a legal system).

A new stream of work [175, 296, 66, 113] focused on using Linked Data [65], which is a by-product of the Semantic Web and has been promoted to address interoperability and sharing issues for open and online datasets, to enhance the intelligence of Web and enterprise search as well as in supporting information integration and entity extraction. These approaches used Linked Data to provide a publishing paradigm in which not only documents, but also data, can be a first class citizen of the Web and enterprise search.

3.9 Summary

In this chapter, we presented a data model and query language for querying and analyzing business processes execution data. The data model supports structured and unstructured entities, and introduces folder and path nodes as first class abstractions. Folders allow breaking down an event log into smaller clearer groups. Mining folder nodes may result in discovering process related relationships (e.g., process instances) and abstractions (e.g., process models) which can be stored in path nodes for further analysis. The query language, FPSPARQL, defined as an extension of SPARQL to manipulate and query entities, and folder and path nodes.

We have developed an efficient and scalable implementation of FPSPARQL. We provided a front-end tool for the exploration and visualization of results in order to enable users to examine the event relationships and the potential for discovering process instances and process models. To evaluate the viability and efficiency of FPSPARQL, we have conducted experiments over large event logs. The evaluation shows that the approach is performing well.

Chapter 4

Analyzing Cross-cutting Aspects in Ad-hoc Processes

4.1 Introduction

Ad-hoc processes have flexible underlying process definition. 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 becomes challenging and requires analyzing the cross-cutting [204] aspects of process artifacts.

A cross-cutting concern defined as a "universal program behavior, one that is potentially needed in many disparate parts of a program, but is often developed and modeled separately" [127]. In particular, aspect-oriented programming (AOP) [204] paradigm can be applied to the process artifacts in BP execution data, where AOP is used to add support for cross-cutting aspects (e.g., object versioning, event logging, and memory management) to existing code without directly modifying that code [127]. 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), provenance [93] (what manipulations were performed on the artifact to get it to this point), security (who has access to the artifact over time), and privacy (what actions were performed to protect or release artifact information over time). In this chapter we focus on cross-cutting aspects which are related to the *evolution* of business process artifacts over time, i.e., versioning and provenance. Analyzing these aspects will expose many hidden interactions among entities in process graphs.

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. This lead to dynamic and ad-hoc changes of process execution paths in different process instantiations. Activities of knowledge workers in knowledge intensive processes involve directly working on and manipulating artifacts to the extent that these activities can be considered 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 [314], 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. This is challenging, as artifacts can be accessed/modified by different actors over time, various versions of artifacts can be generated in different sysems/departments, and each artifact version can be derived from various sources. To address these challenges, we present a framework, simple abstractions and a language for analyzing cross-cutting aspects in ad-hoc processes. The unique contributions of the paper are as follows:

- 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 ob-

served 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 [53], a graph query language for analyzing processes execution, for explorative querying and understanding of cross-cutting aspects in ad-hoc processes. We introduce simple templates for querying evolution, derivation, and timeseries of artifacts. We present the evaluation results on the performance and the quality of the results using a number of process event logs.
- We provide a front-end tool for assisting users to create queries in an easy way and visualizing proposed graph model and query results.

The remainder of this paper is organized as follows: We fix some preliminaries in Section 4.2. Section 4.3 presents an example scenario in case management applications. In Section 4.4 we introduce a data model for representing cross-cutting aspects in ad-hoc processes. In Section 4.5 we propose a query language for querying the proposed model. In Section 4.7 we describe the query engine implementation and evaluation experiments. Finally, we discuss related work in Section 4.8, before concluding the paper in Section 4.9.

4.2 Preliminaries

Definition 1. ['Artifact'] An artifact is defined as a digital representation of something, i.e., data object, 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'] 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. An artifact can capture its *current state* as a version and can restore its state by loading it. Each version v_i is represented as a data object 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. In order to represent the history of an artifact, it is important to create archives containing all previous states of an artifact. The archive allows us to easily answer certain temporal queries such as retrieval of any specific version from the archive and finding the history of an artifact. Archives can be managed using temporal databases [243].

Definition 3. ['Activity'] An activity is defined as an action performed on or caused by an artifact version. For example, 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, 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 [93]. 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 [248].

Definition 8. ['Cross-cutting Aspects'] Aspect-Oriented Programming [204] (AOP), has been proposed as a technique in computing for improving separation of concerns in software. In AOP, a concern is a particular set of behaviors needed by a computer program, is modeled as an aspect, and can be as general as database interaction or as specific as performing a calculation. In this context, cross-cutting concerns are aspects of a program which affect other concerns, e.g., object versioning, event logging, and memory management. Process artifacts, like code [127], also has cross-cutting aspects such as versioning, provenance, security (who has access to the artifact over time), and privacy (what actions were performed to protect or release artifact information over time). In this chapter we focus on cross-cutting aspects which are related to the *evolution* of business process artifacts over time, i.e., versioning and provenance. Analyzing these aspects will expose many hidden interactions among entities in process graphs.

Definition 9. ['Case Management' and 'Case'] Most important processes for organizations today involve knowledge work. An example of this is the case of government agencies, banks, big legal firms and insurance providers where complex customer and service interactions need to be handled. Case Management, also known as case handling [314], can be defined as a common approach to support knowledge intensive processes. When a customer initiates a request for some services, the set of interactions among people (e.g., customer and other relevant participants) and artifacts from initiation to completion is known as the 'case'. Understanding such

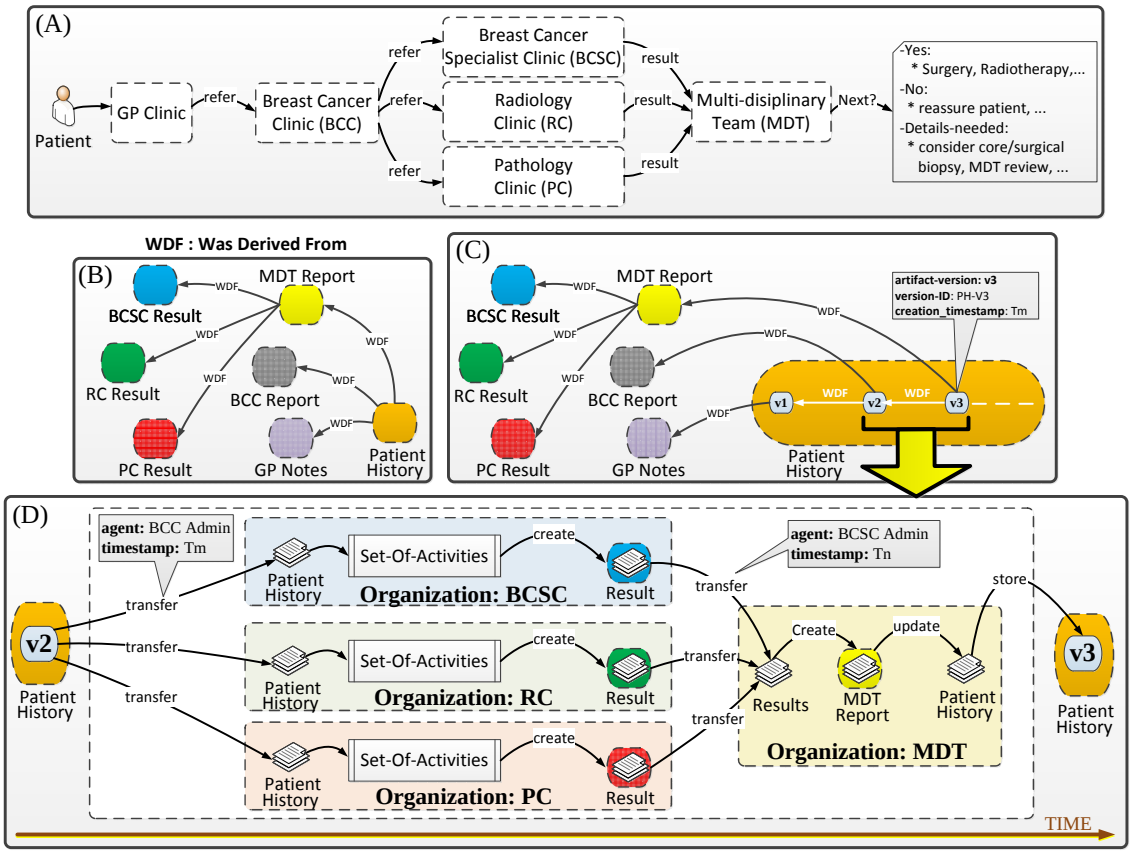


Figure 4.1: Example case scenario for breast cancer treatment including a case instance (A), parent artifacts, i.e. ancestors, for patient history document (B) and its versions (C), and set of activities which shows how version v_2 of patient history document develops and changes gradually over time and evolves into version v_3 (D).

ad-hoc processes entails identifying the interactions, among people and artifacts, where artifacts are developed and changed gradually over a long period of time.

4.3 Example Scenario: Case Management

To understand the problem, we present an example scenario in the domain of case management. This scenario is based on breast cancer treatment cases in Velindre hospital [314]. Figure 4.1-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). BCC checks the patients history and requests assessments such as an examination, imaging, fine needle aspiration, and core biopsy. Therefore, BCC administrator refers patient to Breast

Cancer Specialist Clinic (BCSC), Radiology Clinic (RC), and Pathology Clinic (PC), where these departments apply medical examinations and send the results to Multi-Disciplinary Team (MDT). The results are gathered by the MDT coordinator and discussed at the MDT team meeting involving a surgeon oncologist, radiologist, pathologist, clinical and medical oncologist, and a nurse.

Analyzing the results and the patient history, MDT will decide for next steps, e.g., in case of positive findings, non-surgical (Radiotherapy, Chemotherapy, Endocrine therapy, Biological therapy, or Bisphosphonates) and/or surgical options will be considered. During interaction among different systems, organizations and care team professionals, a set of artifacts will be generated. Figure 4.1-B represents parent artifacts, i.e., ancestors, for patient history document, and Figure 4.1-C represents parent artifacts for its versions. Figure 4.1-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.

4.4 Representing Cross-cutting Aspects

4.4.1 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 [247]. 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: (i) explicitly representing the time information in the provenance graphs, and (ii) providing timed abstractions and efficient mechanisms for time-aware querying of provenance graphs over an ever increasing volume of data.

The existing provenance models, e.g., the open provenance model (OPM) [216,

248], 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, annotations assigned to an artifact (e.g., a file or Web resource) today may no longer be relevant to the future representation of that entity, as entity attributes are very likely to have different states over time and the temporal annotations may or may not apply to these evolving states.

Due to the implicit treatment of time, abovementioned approaches do not enable explicit representation of the evolution of relevant subgraphs (i.e., group of interrelated objects such as versions of artifacts) and paths (i.e., discovering historical paths through provenance graphs forms the basis of many provenance queries [90, 182, 201]) over time. For example, the shortest path from a business artifact to its origin may change over time [285] 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.

4.4.2 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 analyze the evolution of artifacts over periods of time. We use and extend the data model proposed in Chapter 3 to represent AEM graphs. In particular, AEM data model supports: (i) uniform representation of nodes and edges; (ii) structured and unstructured entities; and (iii) *folder* nodes, which enable grouping related entities, and *path* nodes, which help in analyzing the history of artifacts.

In this chapter, 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.

AEM Entities

An entity is an object that exists independently and has a unique identity. AEM consists of two types of entities: artifact versions and timed folder nodes. Folder nodes represent evolution of artifacts over time.

Artifact Version: Artifacts are represented by a set of instances each for a given point in time. For example, artifact Ar is represented by the set of instances $\{Ar_{t_1}, Ar_{t_2}, Ar_{t_3}, ...\}$ where $\{t_1, t_2, t_3, ...\}$ indicates the activity timestamps at distinct points 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/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 node in Chapter 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 folder node by adapting a monitoring code snippet (see Section 4.6). 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.

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-paths can be used for efficient graph analysis and can be modeled using timed path nodes.

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

These attributes, e.g., actors and organizations, can be stored as individual objects and used for annotating activity edges in the graph.

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. Recall from Chapter 3, that the relationship can be codified using

regular expressions in which alphabets are the nodes and edges from the graph. 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.

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. The change-aware query, documents the evolution of path node by adapting a monitoring code snippet (see Section 4.6). 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.

For example, Figure 4.2-A represents a set of activities showing how version v_2 of patient history develops and changes gradually over time and evolves into version v_3. A query which results in a set of paths, can be used to discover all/specific path(s) between v_2 and v_3, and group them under an activity path (Figure 4.2-B). Merging activities using activity-paths will not lose information, as activities that are important to the user will be visible after the merger. Figure 4.2-C illustrates the attributes for the constructed activity path and its storage and representation. As discussed in Chapter 3, we use triple tables to store objects (object-store) and relationships among them (link-store) in graphs.

Notice that the AEM graph model supports the uniform representation of nodes and edges: path nodes have unique identities and considered as first class abstractions. In this chapter we use path nodes to represent activity paths. As illustrated in Figure 4.2-C, activity paths have set of mandatory attributes, e.g., id, type, starting node, and ending node, and also descriptive attributes.

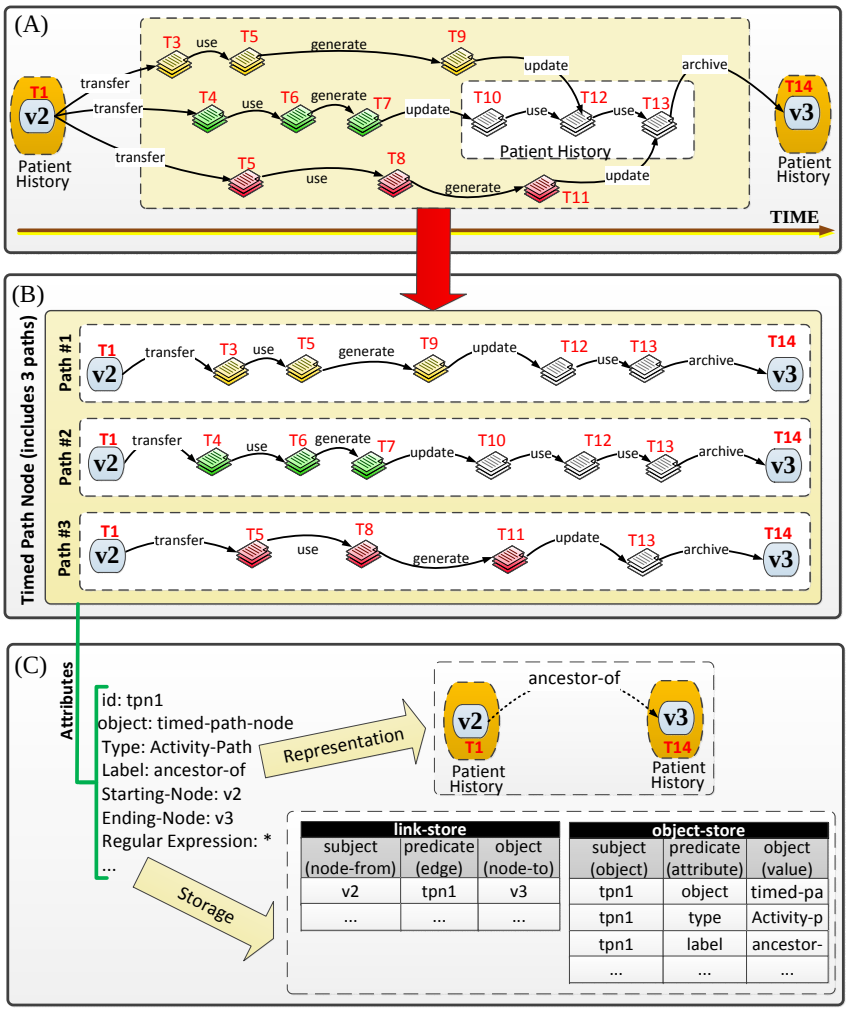


Figure 4.2: Implicit and explicit relationships between versions v_2 and v_3 of patient history document including: (A) activity edges; (B) constructed activity-path stored as a timed (path node) abstraction; and (C) representation and storage of the activity path.

4.5 Querying Cross-cutting Aspects

Querying AEM graphs needs a graph query language that not only supports primitive graph queries but also is capable of: (i) constructing timed folders and group related activities (paths). In general, the output of every query can be stored as folder/path and used for further querying; (ii) applying further queries to constructed folders/paths, e.g., to analyze their evolution or understand the merged activities over time; and (iii) applying external tools and algorithms (e.g., to discover shortest path and frequent patterns) to AEM graphs for further analysis.

Recall from Chapter 3, that FPSPARQL, a Folder-, Path-enabled extension of

SPARQL, is a graph query processing engine which supports primitive graph queries, constructing folders/paths, applying further queries to constructed folder/path nodes, and applying external tools and algorithms to graphs. In this chapter we extend FPSPARQL to support querying cross-cutting aspects in ad-hoc processes. Moreover, we propose simple query templates for discovering derivation, evolution, and timeseries of artifacts over periods of time

4.5.1 Formalizing AEM Queries

Artifact-centric process queries require traversal of AEM graphs. In order to represent AEM graphs and formalize path queries, we model our prototype based on an RDF like data representation. Our representation, supports uniform representation of nodes and edges: edges are treated as first class citizens where any edge can be described by an arbitrary set of attributes. However, that is not the case in the RDF data model where it is not supported that edges can be described by any attribute information. In our model, a *triple* (**S**ubject, **P**redicate, **O**bject) can be defined as an element of $(\upsilon \cup \beta) \times \upsilon \times \tau$, where τ represents RDF terminology, υ represents set of URI references, and β represents set of blanks. Subjects, predicates and objects, can be either a variable or a literal. As discussed in Chapter 3, we use the '@' symbol for representing attribute edges and distinguishing them from the relationship edges between graph nodes. In particular, an *RDF graph* is a finite set of triples [76].

Considering ℓ as set of literals, $\upsilon \cup \ell$ will represent the vocabulary ν. Let ν be the set of names appearing in AEM graph and $\nu_{edge} \subseteq \nu$ be a set of names on the arcs in the graph. The label on each $e \in \nu_{edge}$ defines a relationship between the entities in the graph and also allows us to navigate across the different nodes by a single hop. Consequently, an activity path, in AEM graph, is a sequence of triples, where the object of each triple in the sequence coincides with the subject of its successor triple in the sequence, and the predicate is an activity relationship having the following mandatory attributes: what, how, when, who, where, and which.

4.5.2 Simplifying Path Queries

Discovering activity paths through AEM graphs forms the basis of many AEM queries. In order to discover paths and apply further operations to discovered paths, we use *pconstruct* and *apply* commands proposed in Chapter 3. In FPSPARQL, writing path queries and generating regular expression can be complex and requires being familiar with FPSPARQL/SPARQL syntax. In this chapter, we extend FPSPARQL with *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 [345]. 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 4.1 represents the time-semantics that we support in FPSPARQL queries. The *when* construct, i.e., $when(t1, t2, t3, t4)$ proposed in the above extension, will be automatically translated to *timesemantic* construct in FPSPARQL. Following we will introduce derivation, evolution, and timeseries queries.

4.5.3 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 4.1-D). Following is the sample FPSPARQL query for this example.

```
1 discover.evolutionOf(?artifact1,?artifact2);
2 where{
3  ?artifact1 @id v2. ?artifact2 @id v3.
4  #Path node attributes
5  ?pathAbstraction @id tpn1.
6  ?pathAbstraction @label 'ancestor-of'.
7  ?pathAbstraction @description 'version evolution'.
8 }
```

Table 4.1: FPSPQARL time semantics.

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

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 6), the implicit relationship, with ID 'tpn1', between versions v_2 and v_3 will be added to the graph (see the query translation). Also, if Adam would be interested to see the whole evolution of version v_3, he does not need to specify the first parameter, e.g., in "evolutionOf(,?artifact2)". In the above example, attributes of variables '?artifact1' and '?artifact2' can be defined in the *where* clause. Considering Figure 4.2-B, the result of this query will be a set of paths between versions v_2 and v_3, and can be stored in an activity-path. This query will automatically be translated to the following path construction query proposed in Chapter 3:

```
1  pconstruct tpn1 as ?evolution (?startNode,?endNode, *)
2  where {
3   ?startNode @id v2.
4   ?endNode @id v3.
5   #defining the path node attributes
6   ?evolution @timed true. #timed path node
7   ?evolution @type Activity-Path.
8   ?evolution @Starting-Node v2.
9   ?evolution @Ending-Node v3.
10  ?evolution @type Activity-Path.
11  ?evolution @label 'ancestor-of'.
12  ?evolution @description 'version evolution'.
13  #add the activity-path to the graph
14  ?evolution @addToLinkStore 'triple(v2,tpn1,v3)'.
15 }
```

In Chapter 3, we introduced the $PCONSTRUCT$ command to construct a path node. This command is used to discover transitive relationships between two entities and store it under a path node name. In this chapter we extend this command with

two attributes: (i) '@timed': setting the value of attribute '@timed' to *true* for the path node, will assign a monitoring code snippet to this path node (line 6). The code snippet is responsible for updating the path node content over periods of time: new paths can be added to timed path nodes over time; and (ii) '@addToLinkStore': this attribute is used to add the activity-path to the original graph, using a simple triple format (line 14). In this example the value 'triple(v2,tpn1,v3)' for this attribute, will generate the link between versions v_2 and v_3 represented in Figure 4.2-C. Attribute 'label' (line 11) shows the label of this implicit edge in the graph. The value '*' for the regular expression (line 1) will discover all the paths between the starting node, v_2, and the ending node, v_3. Variable '?evolution' represents the activity-path to be constructed, i.e., 'tpn1' (lines 6 to 12). As mentioned earlier, activity paths have set of mandatory attributes, e.g., id, type, starting node, and ending node, and also descriptive attributes, e.g., description (line 12).

Query Filters. Adam can use the *filter* statement to answer to specific evolution questions: (i) *when queries*: what happens to the artifact during the first three weeks that they are received?; (ii) *where queries*: what happens to the artifact in radiology clinic?; (iii) *who queries*: who (which roles) work on the artifact?; and (iv) *which queries*: what happens to the artifact in the Wiki system? For example, Adam is interested to see the patient history evolution, for the patient having the id 'X14', during November 2012 in radiology clinic. Following is the sample FPSPARQL query for this example.

```
1  discover.evolutionOf( ,?artifact);
2  filter( where(?location),
3          when(?t1,?,?,?t2));
4  where{
5   ?artifact @patient-ID 'X14'.
6   ?location @name 'radiology'.
7   ?t1 @timestamp '11/1/2011 @ 0:0:0'.
8   ?t2 @timestamp '12/1/2011 @ 0:0:0'.
9   #timestamp: M/D/Y @ h:m:s
```

```
10  #Path node descriptive attributes
11  ?pathAbstraction @id 'tpn2'.
12 }
```

In this example, *filter* statement (lines 2 and 3) is used to restrict the result to those activities, happened during November 2011 (lines 7 and 8) in radiology clinic (line 6). As Adam is interested to see the whole evolution of patient history document, he didn't specify the first parameter in the *evolutionOf* construct, i.e., "evolutionOf(,?artifact2)" (line 1). This query will automatically be translated to the following path construction query:

```
1  pconstruct tpn2 as ?patientHisroty
2    (?startNode,?endNode, ?edge (?node ?edge)* )
3  where {
4   #regular expression
5    ?startNode @isA entityNode.
6    ?startNode @id ?stID. ?startNode @patient-ID X14.
7    ?endNode @isA entityNode.
8    ?endNode @id ?stID. ?endNode @patient-ID X14.
9    ?node @isA entityNode. ?node @patient-ID X14.
10   ?edge @isA edge.
11   ?edge @where 'radiology'.
12   ?edge @timestampe ?ts.
13   filter(timesemantic(?ts,[t1,?,?,t2])).
14   ?t1 @timestamp '11/1/2011 @ 0:0:0'.
15   ?t2 @timestamp '12/1/2011 @ 0:0:0'.
16  #defining the path node attributes
17   ?patientHisroty @timed true.
18   ?patientHisroty @type Activity-Path.
19   ?patientHisroty @Starting-Node ?stID.
20   ?patientHisroty @Ending-Node 'X14-med-doc'.
21 }
```

In this example variables '?startNode' and '?endNode' denotes any artifact related to the patient having the ID 'X14' which being used between timestamps '?t1' and '?t2' (lines 14 and 15). Respectively, variables '?edge' and '?node' denotes any edges and nodes in the transitive relationship between starting and ending nodes in the regular expression (lines 9 to 12). To discover the activities applied to artifacts in radiology clinic, the attribute 'where' for the activity relationship '?edge' is set to the value 'radiology' (line 11). The variable '?patientHisroty' is used to define path node attributes (lines 17 to 20). The *when* statement (i.e. $when(t1, ?, ?, t2)$) in the evolution query have been translated to the special construct $timesemantic(?ts, [t1, ?, ?, t2])$ in FPSPARQL (line 13) which is used to represent the activity timestamps '?ts', to be in a specific time interval $[t1, ?, ?, t2]$. Recall from previous example, that this implicit relationship will not be added to the original graph as the attribute 'label' have not been specified in the evolution query.

4.5.4 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, considering the motivating scenario, Adam is interested to query the derivation of version v_3 of the patient history (see Figure 4.1-C). Following is the sample FPSPARQL query for this example.

```
1  discover.derivationOf(?artifact);
2  where{
3   ?artifact @id v3.
4  }
```

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. Considering Figure 4.1-C, the result for this query will be the set

"{MDT-report, BCSC-result, RC-result, PC-result}". This query will automatically be translated to the following path construction query:

```
1  select ?startNode from
2  pconstruct derivation_v3
3    (?startNode, ?endNode, '?edge (?node ?edge)*')
4  where {
5   ?startNode @isA entityNode.
6   ?endNode @isA entityNode.
7   ?endNode @type artifactVersion.
8   ?endNode @id v3.
9   ?node @isA entityNode.
10  ?edge @isA edge.
11 }
```

In Chapter 3, we used *pconstruct* statement to discover paths: i) between two nodes; ii) starting from a specific node and ending to a set of nodes; and iii) starting from a set of nodes and ending to a specific node. In this example, we use *pconstruct* statement to discover paths between set of starting nodes (ancestors) to a specific ending node (version v_3 of patient history). The result of this query will be set of artifacts/versions reachable from version v_3 of patient history document. For the sake of simplicity we enabled applying further operations to the constructed path node using select statement (line 1), e.g., the variable '?startNode' in *select* statement will return the ancestors of version v_3 of patient history. Recall from Chapter 3, that it is possible to use *apply* statement for applying further operations to the constructed path nodes. Moreover, the query in this example can be timed, i.e., using '@timed' attribute.

Query Filters. Adam can use the *filter* statement to answer specific derivation questions. For example, he can find specific artifacts which v_2 was derived from them: (i) in radiology clinic (using *where* statement); (ii) between the time periods τ_1 and τ_2 (using the 'when(τ_1,?,?,τ_2)' statement); or (iii) in a specific system (using

which statement). For example, Adam is interested to find all ancestors of version v_3 of patient history (see Figure 4.1-C) generated in radiology clinic before March 2011. Following is the sample FPSPARQL query for this example.

```
1 discover.derivationOf(?artifact);
2 filter( where(?location),
3         when(?,?,?,?t1) );
4 where{
5  ?artifact @id v3.
6  ?location @name 'radiology'.
7  ?t1 @timestamp '3/1/2011 @ 0:0:0'.
8 }
```

In this example, *filter* statement is used to restrict the result to those activities, happened before March 2011 in radiology clinic.

4.5.5 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_5. Following is the sample FPSPARQL query for this example.

```
1 discover.timeseriesOf(?actor);
2 filter( when("T1",?,?,"T5") );
3 where{
4  ?actor @id Eli-id.
5 }
```

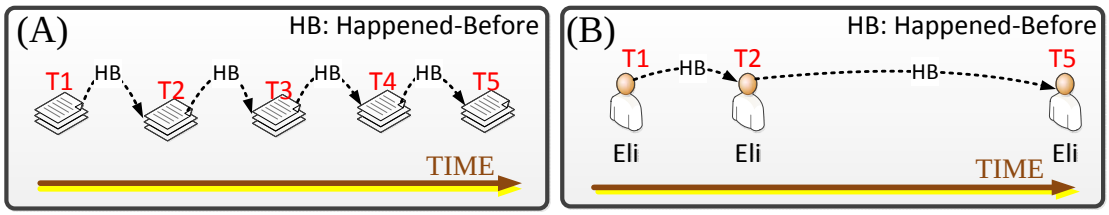


Figure 4.3: Sample timeseries for: (A) patient history document between τ_1 and τ_5; and (B) Eli, an actor, acting on patient history between τ_1 and τ_5.

In this example, *timeseriesOf* statement (line 1) is used to represent the timeseries of Eli, i.e., the variable '?actor'. Attributes of variable ?*actor* can be defined in the *where* clause (line 4). Figure 4.3-B represents the timeseries of Eli for activities she did on top of patient history document. Figure 4.3-A represents time series of patient history document between τ_1 and τ_5. Similar to evolution and derivation queries, *timeseriesOf* statement can be timed and may contain *filter* statement, where *filter* statement can be used to answer specific timeseries questions.

4.5.6 Constructing Timed Folders

In Section 4.5.3 we discussed how we extend path construction queries to support time aware querying. In this section, we discuss how to construct timed folder nodes. In particular, to construct a *timed folder* node, we use FPSPARQL's *fconstruct* statement proposed in Chapter 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 periods of time: new members can be added to timed folders over time. The syntax for a basic construction query of a timed folder node is given as follows:

```
fconstruct <Folder_Node Name> as ?folder
[select ?var1 ?var2 ... | (Folder1, Folder2,...)]
where {
 ?folder @timed true.
 #other patterns
}
```

For example, considering Figure 4.1-C, a timed folder can be constructed to represent a patient history document which may contain various versions of this artifact. New versions (and activities on top of it) can be added to this folder over time. Following is a sample FPSPARQL query for this example.

```
1  fconstruct X14-patient-history as ?med-doc
2  select ?version
3  where {
4   #defining the path node attributes
5    ?med-doc @timed true.
6    ?med-doc @type artifact.
7    ?med-doc @description 'history for patient #X14'.
8   #specifying nodes to be added to the folder
9    ?version @isA entityNode.
10   ?version @patient-ID X14.
11 }
```

In this example, variable '?med-doc' represents the folder node to be constructed, i.e., 'X14-patient-history' (line 1). This folder is of type 'artifact' (line 6). Setting the attribute *timed* to *true* (line 5) will force new artifacts having the patient ID 'X14' (line 10) to be added to this folder over time. The attribute 'description' used to describe the folder (line 7). The variable '?version' is an AEM entity and represents the patient history versions to be collected (line 9). Attribute 'patient-ID' (line 10) indicate that the version is related to the patient history of the patient having the id 'X14'.

As another example, it is possible to construct a timed folder to monitor the artifacts touched (created, transformed, used, deleted, stored, or transferred) by Eli. As the result of this query, all the artifacts touched by Eli will be added to the constructed timed folder. Moreover, new artifacts can be added to this folder over time. Following is a sample FPSPARQL query for this example.

```
1  fconstruct Eli_artifacts as ?Eli_art
2  select ?artifact1
3  where {
4   ?Eli_art @timed true.
5   ?Eli_art @type artifact.
6   ?Eli_art @description 'artifacts generated by Eli'.
7   #
8   ?artifact1 ?activity ?artifact2.
9   ?artifact1 @isA entityNode.
10  ?artifact2 @isA entityNode.
11  ?activity @isA edge.
12  ?activity @who Eli_id.
13 }
```

In this example, variable ?*Eli_art* represents the folder node to be constructed, i.e., 'Eli_artifacts' (line 1). This folder is of type 'artifact' (line 5). Setting the attribute *timed* to *true* (line 4) will force new artifacts touched by Eli to be added to this folder over time. The pattern '?artifact1 ?activity ?artifact2' (line 8) illustrates an activity relationship between two artifacts, '?artifact1' and '?artifact2'. The activity relationship '?activity' (line 11) has a set of mandatory attributes discussed in Section 4.4.2. To discover the artifacts touched by Eli, i.e., '?artifact1', the attribute 'who' for the activity relationship '?activity' is set to Eli's identification (line 12). More activity relationship attributes can be used, e.g., to discover the artifacts generated, deleted, or stored by an actor.

Querying Timed Folder Nodes. In Chapter 3, we introduced the *apply* statement to apply further operations to constructed folder nodes. For example, consider a user who is interested to retrieve information about patient history folder, e.g., folder 'X14-patient-history' constructed in previous examples, between timestamps τ_2 and τ_7. Following is the FPSPARQL query for this example.

```
1 (X14-patient-history)
2 apply (
3  select ?a
4  where {
5    ?a @isA entityNode.
6    ?a @timestamp ?ts.
7    filter( timesemantic(?ts,[t2,?,?,t7]) ).
8  }
9 )
```

In this example the query applied to the constructed timed folder node 'X14-patient-history'. Variable '?a' represents the members (i.e., artifact versions) of the folder node whose (creation) timestamp '?ts' (line 6) falls between time τ_2 and τ_7 (line 7). The 'timesemantic' construct is used to filter the requested timestamps.

4.6 Architecture and Implementation: Temporal Extension

4.6.1 Architecture

In Chapters 3, we introduced FPSPARQL graph processing architecture, where the architecture consists of the following components: Graph Loader, Data Mapping Layer, Time-aware Controller, Query Mapping Layer, Regular Expression Processor, External Algorithm/Tool Controller, and Query Optimizer. In this chapter, we instrument the Time-aware Controller to support the execution of FPSPARQL temporal queries. In particular, Time-aware Controller will be responsible for creating a monitoring code snippet and allocate it to a timed folder/path node in order to monitor its evolution and update its content.

Time-aware Controller enables users to set an AEM query as a: (i) pull query, where a time-tracker will be assigned to this query. Time-tracker will trigger the

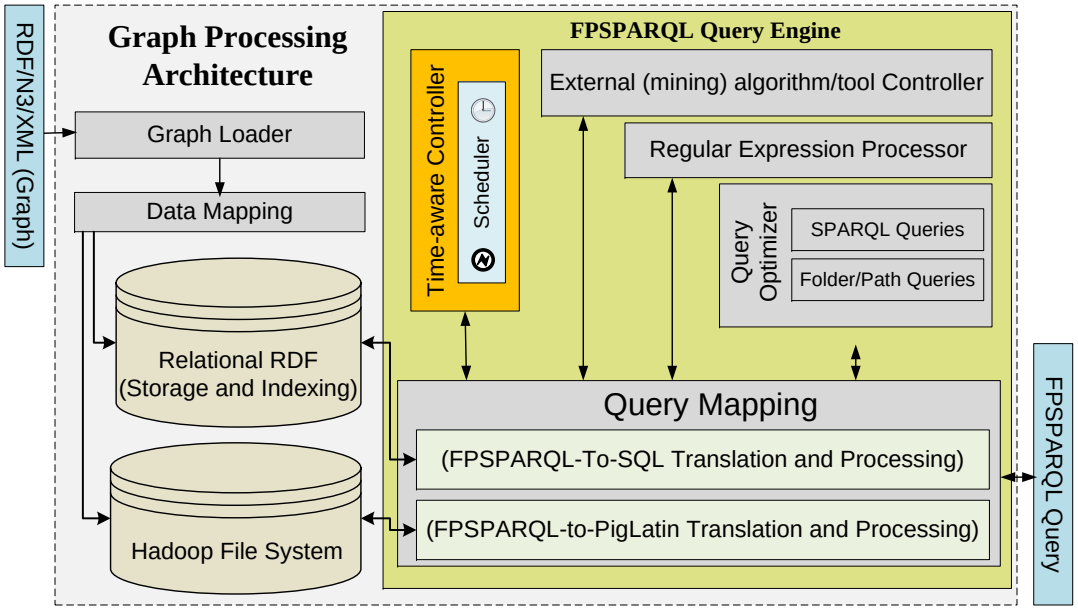


Figure 4.4: FPSPARQL graph processing architecture: analytics extension.

start of the querying process at specific user-defined intervals. The interval attribute can be set manually in the query engine; or (ii) push query, where a database trigger will be assigned to the entities in the query result. Future changes applied to these entities and their relationships will result in re-executing the query. Users can initialize an intelligent agent in order to allocate it to a timed folder/path node and set its time interval or assign it to a database trigger.

Moreover, as discussed in Chapter 3, Time-aware Controller is responsible for data changes and incremental graph loading: RDF databases are not static and changes may apply to graph entities (i.e. nodes, edges, and folder/path nodes) over time. Updates are mostly insertions of new triples into the object store and link store. At the current version of FPSPARQL query engine, updates cannot be performed concurrently with queries: there may be the need for full-fledged ACID transactions. Figure 4.4 illustrates FPSPARQL graph processing architecture.

Figure 4.5 illustrates an overview of Time-aware Controller architecture. The controller uses numerical priorities, priorities take values from the set of real numbers $\mathbb{R}$, to express precedence constraints over the set of executing code snippets. The priority of code snippet c_i at time τ is given by $p_i(\tau)$. For two code snippets c_i and c_j and a time point τ, c_i will execute in preference to c_j if and only if $p_i(\tau) > p_i(\tau)$.

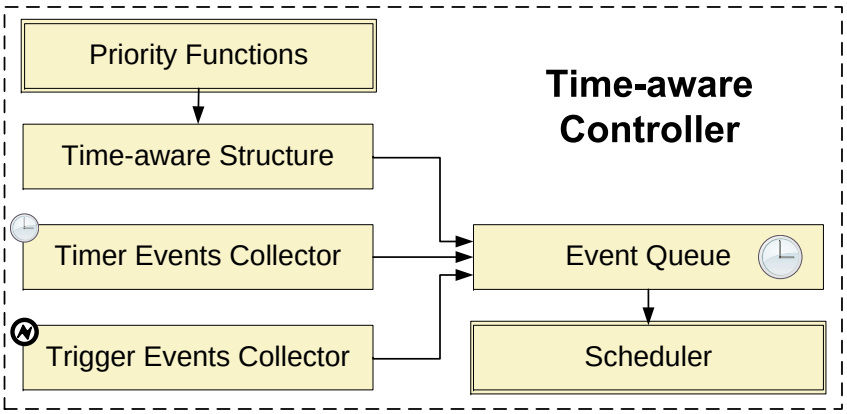


Figure 4.5: Overview of Time-aware Controller architecture.

The Time-aware Controller is composed of the following components:

- Priority Functions: are functions of priority with respect to relative time. They detail how a code snippets priority varies relative to a deadline.
- Time-aware Structure: is a data structure that describes how relative priorities for all agents vary as a function of time. It gives the relative priority ordering of all code snippets possibly executing at time τ.
- Timer Events Collector: is responsible to monitor and manage timer events, i.e., pull queries.
- Trigger Events Collector: is responsible to monitor and manage trigger events, i.e., push queries.
- Event Queue: is a mechanism of dealing with asynchronous events in a synchronous manner. In particular, every time there is a change in the relative precedence of executing code snippets, a timer/trigger event is generated by Time-aware Structure and placed into the Event Queue.
- Scheduler: is responsible for executing timer/trigger events queued in the Event Queue and in the scheduled time.

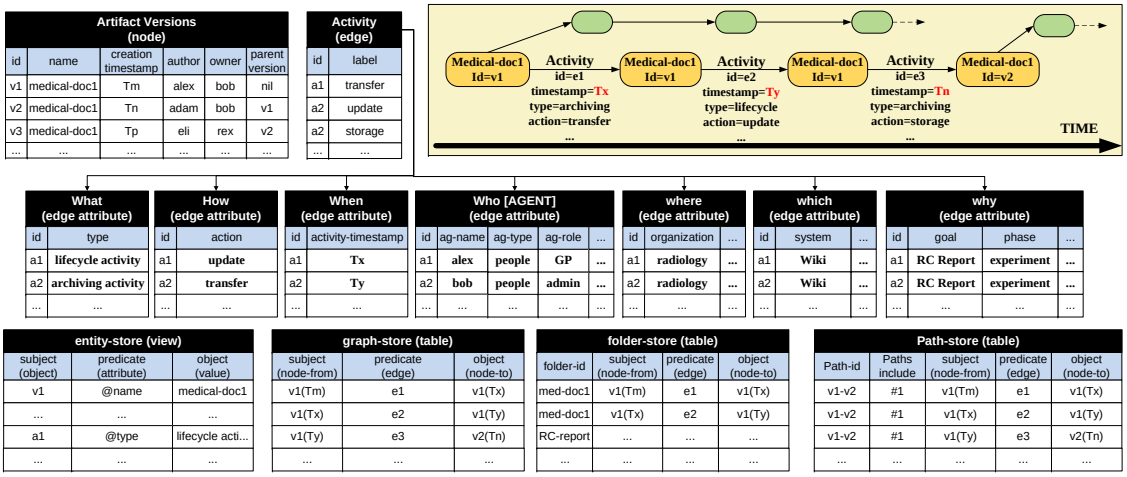


Figure 4.6: Physical layer for storing a sample AEM graph and tables to store AEM entities and relationships.

4.6.2 Implementation

Details about FPSPARQL query engine implementation is presented in Chapter 3. In this Chapter, we instrument the query engine with Time-aware Controller. We model graphs based on a RDF like data representation. Figure 4.6 represents a sample AEM graph and tables to store the graph including: (a) artifact versions, to store AEM entities; (b) activity, to store the relationships among entities. Relationship's attributes can be stored in what, how, when, who, where, which, and why tables; (c) entity store, which is a view on top of graph entities and relationships. This triplestore, stores the node/edge ID in the *subject* column, node/edge attribute in the *predicate* column, and node/edge value in the *object* column; (d) graph store, which contains directed links between graph entities. This triple store, stores the starting node ID in the *subject* column, edge ID in the *predicate* column, and ending node ID in the *object* column; (e) timed folder store, which stores related entities and relationships among them in a triplestore. The 'folder-id' column is added to this triplestore for identifying folders; and (f) timed path store, which stores activity edges between two entities in the graph. The 'path-include' column identifies each path, and the 'path-id' column identifies set of paths considered as an activity-path.

Moreover, we have implemented a front-end tool to assist process analysts in two steps:

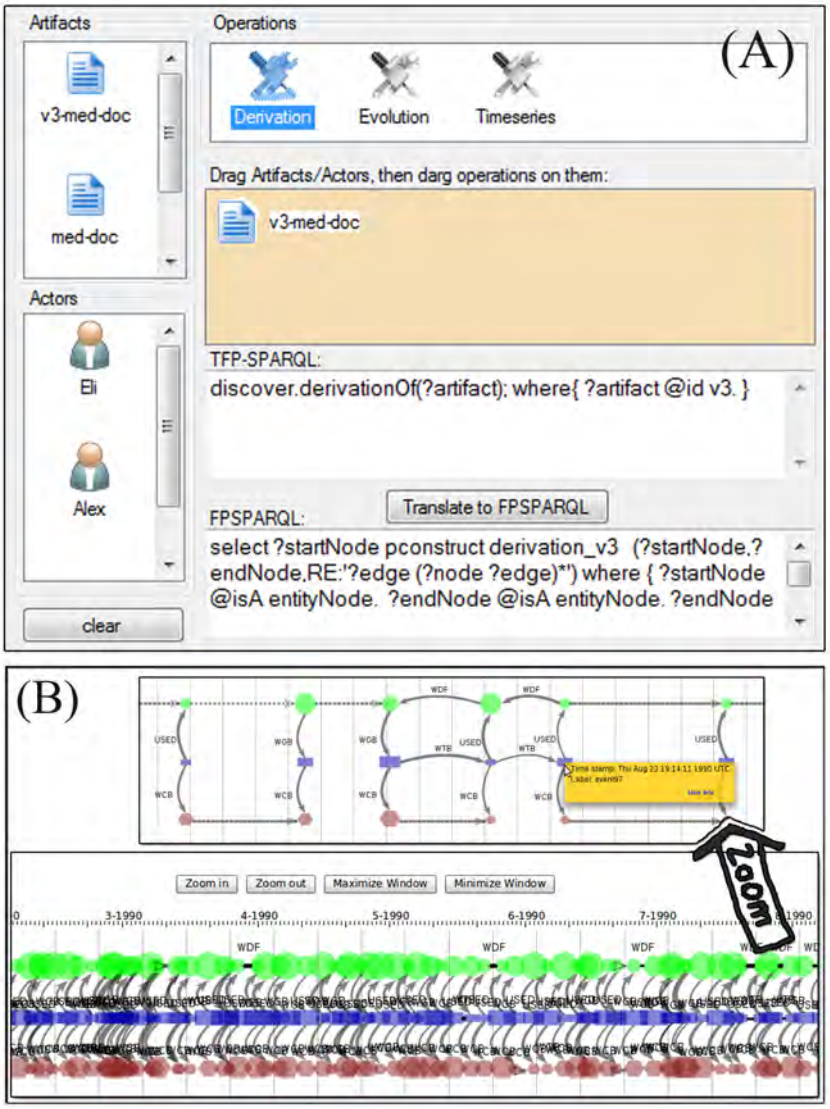


Figure 4.7: Screenshots of front end tool: (A) Query assistant tool; and (B) graph visualization tool: to visualize AEM graphs.

Step1: [Query Assistant] We provided users with a query assistant tool 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. Moreover it is possible to generate the FPSPARQL query by clicking on "Translate to FPSPARQL" button. Also, users can use the tool to generate the regular expressions and other path queries they are interested in. Figure 4.7-A illustrates a screenshot of this tool while generating the derivation query in Section 4.5.2.

Step2: [Visualizing] We provided users with a graph visualization tool for the exploration of graphs and query results. For the AEM graph exploration, we provide users with a timeline like graph visualization with facilities such as zooming in and out. Figure 4.7-B illustrates a screenshot of this tool while generating an evolution query.

4.7 Experiments

4.7.1 Datasets

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, this scenario is built on our experience on managing an online project-based course[2]; and (iii) Supply Chain Management log.

Dutch Academic Hospital

The real-life event log, taken from a Dutch Academic Hospital, contains events related to a heterogeneous mix of patients diagnosed with cancer (at different stages of malignancy) pertaining to the cervix, vulva, uterus and ovary. The event log contains 1143 cases and 150291 events referring to 624 distinct activities. Details about this event log can be found in [69]. Given the heterogeneous nature of these cases, we first applied a preprocessing phase to adapt this dataset to artifact-centric case scenarios. For example, we created more homogeneous subsets of cases, e.g., patients having a particular type of cancer. Then we assigned each case in the event log to a patient document history. Afterward, we generated versions for document histories according to event timestamps: the event log contains rich information stored as attributes both at the event level and at the case level. Finally we generated AEM graph model out of preprocesses log, where the generated graph includes artifacts and activity relationships among them.

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

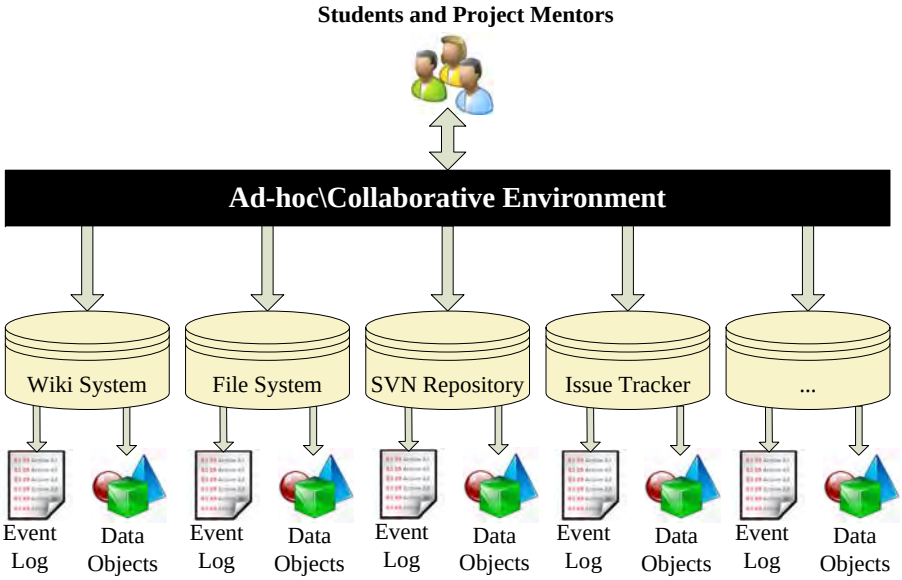


Figure 4.8: e-Enterprise course scenario.

e-Enterprise Course

This scenario, is built on our experience on managing an online project-based course "e-Enterprise Projects". In this scenario, each project can be considered as a case process, where various case workers (e.g. students, mentors and lecturers) are involved. As an example, in the 2^{nd} semester of 2009 we had 66 people (60 students + 5 project mentors + 1 lecturer) involved in course activities. During this semester, fifteen projects (i.e., case instances) were defined, where each case handled by group of four students and one mentor. Each mentor supervised 3 projects. The development process of each project went through a sequence of pre-defined phases: brainstorming, requirements analysis, design phase, prototype implementation, testing, and final product delivery. For each phase various artifacts can be created, e.g., brainstorming documents and records, and each artifact version can be derived from various sources, e.g., IEEE or other templates, and can be accessed/modified by different case workers over periods of time.

In order to document the evolution of artifacts, the activities of each project have been documented through a Web-based project management system which was equipped with many back-end modules such as: (a) Message board: to exchange message and open discussion topics between the project members; (b) Wiki system:

which is used to collaboratively edit documents related to the activities of projects; (c) Blogging system: where each user has their own blog to edit their own posts; (d) File sharing system: where project members can share access to different files and documents; and (e) SVN repository: to synchronize the editing of the projects source codes. This dataset contains 104,050 events. Figure 5.1 depicts an illustration of this scenario.

Supply Chain Management

This dataset is the interaction log of a supply chain service, developed based on the supply chain management scenario provided by WS-I (the Web Service Interoperability organization). SCM dataset contains 4,050 events. We applied a preprocessing phase to adapt this dataset to a case scenario. Details about this dataset can be found in Chapter 3.

4.7.2 Evaluation

We have compared our approach with that of querying open provenance model (OPM) [216, 248]. OPM, a proposal for a standard graph data model and vocabulary for provenance, presents graph nodes as data artifacts, processes, and agents. Five causal relationships are defined in OPM: a process 'used' an artifact, an artifact 'was-Generated-By' a process, a process 'was-Triggered-By' a process, an artifact 'was-Derived-From' an artifact, and a process 'was-Controlled-By' an agent.

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 4.4.2. The OPM graphs generated based on open provenance model specification [216, 248]. Figure 4.9, represents a sample AEM graph (Figure 4.9-A) for the hospital log, a sample OPM graph generated from a part of AEM graph (Figure 4.9-B), and open provenance model entities and relationships (Figure 4.9-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.

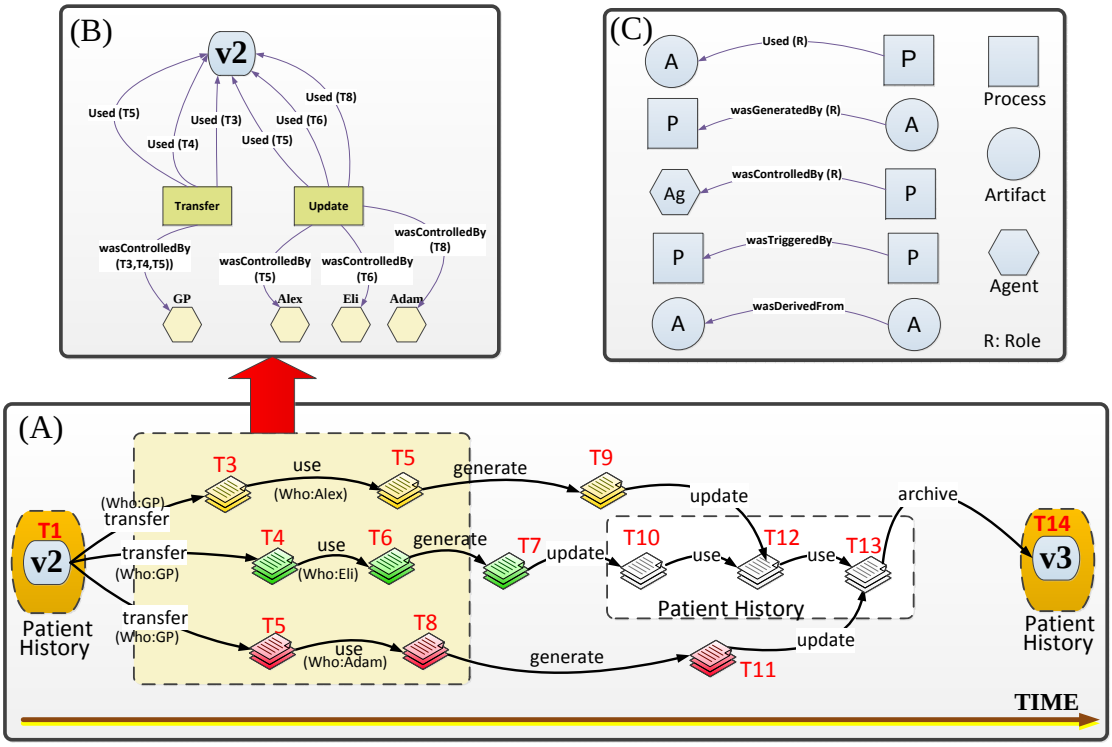


Figure 4.9: A sample AEM graph for the hospital log (A), a sample OPM graph generated from a part of AEM graph (B), and open provenance model entities and relationships (C).

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. 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 4.10 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 4.10 we divided each dataset into regular number of events, then generated AEM and OPM graph for different sizes of datasets, and finally 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.

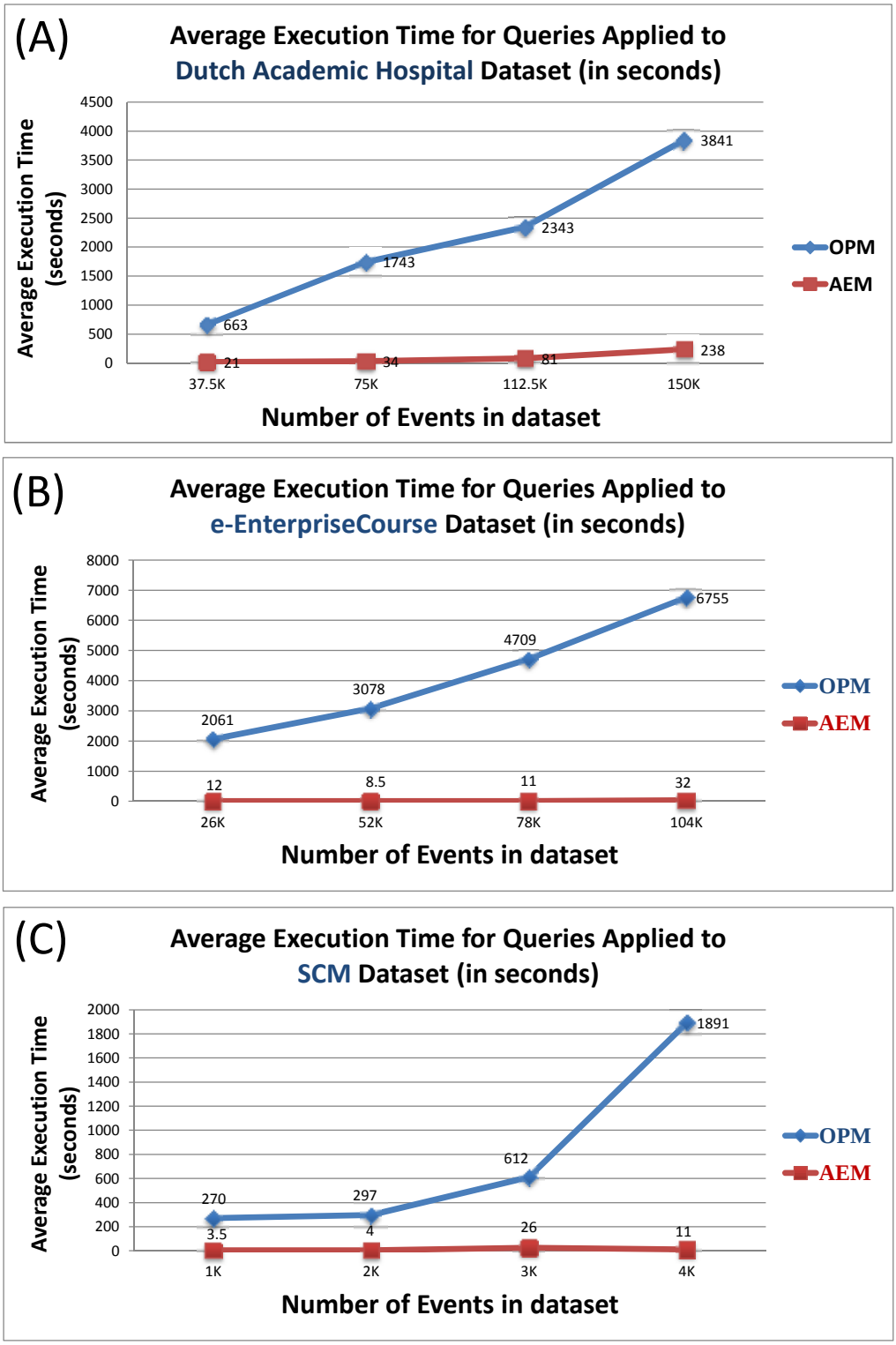


Figure 4.10: The query performance evaluation results, illustrating the average execution time for applying evolution, derivation, and timeseries queries on AEM and OPM graphs generated from: (A) Dutch academic hospital dataset; (B) e-Enterprise course dataset; and (C) SCM dataset.

Quality. The quality of results is assessed using classical *precision* metric which is defined as the percentage of discovered results that are actually interesting. 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.

Performance Comparison Between RDBMS and Hadoop Execution Plans. As mentioned earlier, FPSPARQL queries can be run on two types of storage backend: RDBMS and Hadoop. In this part we compare the performance of query plans on relational triplestores and Hadoop file system. All experiments in this part were conducted on a virtual machine, having 32 cores and 192GB RAM. Figure 4.11 illustrates the performance analysis between RDBMS and Hadoop for queries (average execution time) in Figure 4.10-A applied to Dutch academic hospital dataset. Figure 4.11 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.

Discussion. Evaluation shows that path queries applied to OPM graphs resulted in many irrelevant paths. Moreover, we discovered many cycles in the results of path queries applied to OPM graphs. To eliminate these cycles, we applied the cycle elimination techniques proposed in [19]. To increase the performance of queries, we implemented an interface to support various graph reachability algorithms [19] such as all-pairs shortest path, transitive closure, GRIPP, tree cover, chain cover, path-tree cover, and Sketch. As discussed in Chapter 3, there are two types of graph reachability algorithms: algorithms traversing from starting vertex to ending vertex using breadth-first or depth-first search over the graph, and algorithms checking

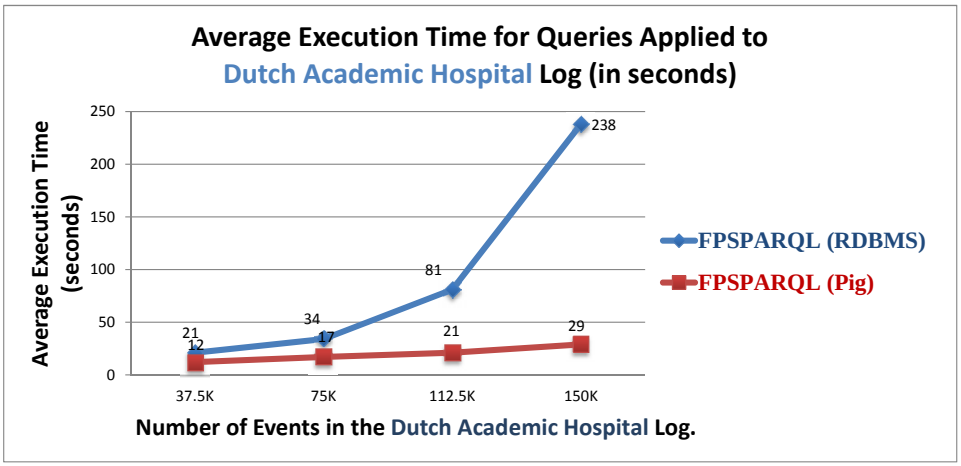


Figure 4.11: The evaluation results, illustrating the performance analysis between RDBMS and Hadoop applied to Dutch academic hospital dataset.

whether the connection between two nodes exists in the edge transitive closure of the graph. In both cases, path queries applied to OPM graphs maximized the consumption of memory and processor and resulted in many irrelevant paths and cycles in the query result.

4.8 Related Work

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

4.8.1 Artifact-centric Processes

Knowledge-intensive processes almost always involves the collection and presentation of a diverse set of artifacts, and capturing human activities around artifacts. This, emphasizes the artifact-centric nature of such processes where *time* becomes an important part of the equation. Many approaches [186, 142, 62, 99, 63] used business artifacts, that combine data and process in a holistic manner, as the basic building block. Some of these works [186, 142, 99] used a variant of finite state machines to specify lifecycles. Some theoretical works [63, 62] 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 [213, 325, 122]. In [213, 325], a document-driven framework, proposed to model business process management systems through monitoring the life-cycle of a document. Dorn et.al. [122], presented a self-learning mechanism for determining document types in people-driven ad-hoc processes through combining process information and document alignment. In these approaches, the document structure is predefined or they presume that the execution of the business processes is achieved through a business process management system (e.g., BPEL) or a workflow process.

Another related line of work is artifact-centric workflows [62] where the process model is defined in terms of the lifecycle of the documents. Some other works [16, 284, 119, 120, 300], focused on modeling and querying techniques for knowledge-intensive tasks. Some of existing approaches [16, 284] for modeling ad-hoc processes focused on supporting ad-hoc workflows through user guidance. Some other approaches [119, 120, 300] 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.

In our model, actors, activities, artifacts, and artifact versions are first class citizens, and the evolution of the activities on artifacts over time is the main focus. The AEM 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.

4.8.2 Provenance

Many provenance models [93, 136, 248, 303] 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) [248], 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 [201, 182, 351, 90, 242]. In ProQL [201], 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 [182] proposed to use a semistructured data model for handling provenance and extended the Lorel query language for traversing and querying provenance graph. NetTrails [351] proposed a declarative platform for interactively querying network provenance in a distributed system in which query execution performs a traversal of the provenance graph. RDFProv [90] is an optimized framework for scientific workflow provenance querying and management. Missie et al. [242] presented a provenance model and query language for collection-oriented workflow systems. They emphasize on querying the provenance of collection of activities. The proposed framework, do not support modeling, querying and analyzing the evolution of group of related entities over time.

4.8.3 Modeling/Querying Temporal Graphs

In recent years, a plethora of work [184, 211, 285] has focused on temporal graphs to model evolving, time-varying, and dynamic networks of data. They capture a snapshot for various states of the graph over time. For example, Ren et al. [285] proposed a historical graph-structure to maintain analytical processing on such evolving graphs. Moreover, authors in [211, 285] 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., [30, 149, 271, 315]) provide temporal ex-

tensions of existing graph models and languages. Tappolet et al. [315] provided temporal semantics for RDF graphs. They proposed τ-SPARQL for querying temporal graphs. Grandi [149] presented another temporal extension for SPARQL, i.e. T-SPARQL, aimed at embedding several features of TSQL2 [243] (temporal extension of SQL). SPARQL-ST [271] and EP-SPARQL [30] 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. Moreover, we extended FPSPARQL [53], our previous work, to support temporal queries and monitor the result of such queries over time.

4.9 Summary

In this chapter, we have presented an artifact-centric activity model (AEM: Artifact Evolution Model) 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. Folders enabled grouping related entities and paths helped in analyzing the history of entities in time. Timed folders and activity-paths show their evolution for the time period they represent. We have extended FPSPARQL, a query language for analyzing business processes execution, 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 where time is considered as annotation. We have conducted experiments over real-world datasets. The results of evaluation showed the viability and efficiency of our approach. A front-end tool has been provided to facilitate the exploration and visualization of AEM graphs and assisting users with generating evolution, derivation, and timeseries queries.

Chapter 5

Analytics over Ad-hoc Process Data

5.1 Introduction

In modern enterprises, businesses accumulate massive amounts of data from a variety of sources. In order to understand businesses one needs to perform considerable analytics over process execution data: data analysis is at the heart of decision making in all business applications. Analytics, i.e., the discovery and communication of meaningful patterns in data, can help in understanding the business data with an eye to predicting and improving business performance in the future. In particular, business process analytics can facilitate the analysis of process graphs in a detailed and intelligent way through describing the applications of analysis, data, and systematic reasoning [241, 35, 210, 74, 171]. Consequently, an analyst can gather more complete insights using techniques such as modeling, summarizing, and filtering.

Applications of business analytics extend to nearly all managerial functions in organizations. For example, considering financial services, applying business analytics on customer dossiers and financial reports can specify the performance of the company over periods of time. As another example, consider the collaborative relationship between researchers, affiliated with various organizations, in the process of writing scientific papers, where it would be interesting to analyze the collaboration

patterns (e.g., frequency of collaboration, degree of collaboration, mutual impact, and degree of contribution) among authors or analyze the reputation of a book, an author, or a publisher in a specific year. Such operations, requires supporting n-dimensional computations on process graphs, providing multiple views at different granularities, and analyzing set of dimensions coming from the entities and the relationship among them in process graphs.

In traditional databases (e.g., relational DBs), data warehouses and OLAP (On-Line Analytical Processing) technologies [15, 88] were conceived to support decision making and multidimensional analysis within organizations. To achieve this, a plethora of OLAP algorithms and tools have been proposed for integrating data, extracting relevant knowledge, and fast analysis of shared business information from a multidimensional point of view. Moreover, several approaches have been presented to support the multidimensional design of a data warehouse. Cubes defined as set of partitions, organized to provide a multi-dimensional and multi-level view, where partitions considered as the unit of granularity. Dimensions defined as perspectives used for looking at the data within constructed partitions. Furthermore, OLAP operations have been presented for describing computations on cells, i.e. data rows.

While existing analytics solutions, e.g., OLAP techniques and tools, do a great job in collecting data and providing answers on known questions, key business insights remain hidden in the interactions among objects and data: most objects and data in the process graphs are interconnected, forming complex, heterogeneous but often semi-structured networks. Traditional OLAP technologies were conceived to support multidimensional analysis, however, they cannot recognize patterns among process graph entities and analyzing multidimensional graph data, from multiple perspectives and granularities, may become complex and cumbersome.

Existing approaches [169, 317, 92, 349, 278, 208, 198, 131], in on-line analytical processing on graphs, took the first step by supporting multi-dimensional and multi-level queries on graphs, however, much work needs to be done to make OLAP heterogeneous networks a reality [168]. The major challenges here are: (i) how to extend decision support on multidimensional networks, e.g., process graphs, considering both data objects and the relationships among them; and (ii) providing

multiple views at different granularities is subjective: depends on the perspective of OLAP analysts how to partition graphs and apply further operations on top of them.

To address these challenges, we provide users with an explorative method to analyze process data from multiple perspectives and granularities. The key contributions of this chapter are:

- Proposing a graph data model, *GOLAP*, for online analytical processing on process graphs. This data model enables extending decision support on multidimensional networks considering both data objects and the relationships among them. We use the notions of folder and path to support multi-dimensional and multi-level views over large process graphs. We redefine OLAP data elements (e.g., dimensions, measures, and cubes) by considering the relationships among graph entities as first class objects.

- Extending SPARQL to support n-dimensional computations on process graphs. The extension supports partitioning graphs (using folder and path nodes) and allows evaluation of OLAP-style operations on graphs independently for each partition, providing a natural parallelization of execution. We propose two types of OLAP operations: *assignments*, to apply operations on entity attributes, and *functions*, to apply operations on network structures among entities. GOLAP operations support UPDATE and UPSERT [333] semantics. We describe optimizations and execution strategies possible with the proposed extensions. We provide a front-end tool for assisting users to create GOLAP queries in an easy way.

The remainder of this chapter is organized as follows: Section 5.2 presents an example scenario. In Section 5.3 we present a graph data model for online analytical processing on graphs and we redefine OLAP data elements for graphs. In Section 5.4 we propose a query language for applying OLAP operations on graphs. In Section 5.5 we describe the query engine implementation and architecture. Section 5.6 reports the evaluation results of the query engine extension. Finally, we discuss related work in Section 5.7, before concluding the chapter in Section 5.8.

5.2 Example Scenario: Collaborative Case Management

To understand the problem, we present an example scenario in the domain of collaborative case management. This scenario is based on the collaborative relationship between researchers, affiliated with various organizations, for the process of writing scientific papers. Leveraging the recent advances of Web 2.0 and social media facilitated the collaboration between researchers: collections of Web technologies such as blogs (e.g., ScienceBlogs[1]), online repositories (e.g., arXiv[2]), share scientific workflows (e.g., MyExperiment[3]), and online file sharing (e.g., Box[4]) can be used to collaboratively generate scientific papers. Projects such as Liquid-Journals [36] can be used to collect interesting links among scientific contributions. Furthermore, libraries and other cultural heritage institutions[5] have been collecting, describing and presenting scientific publications for a long time.

While these approaches do a great job in collecting and managing scientific knowledge, key business insights remain hidden in the interactions among scientific entities, e.g., publications and authors. For example, a user may be interested in understanding collaboration patterns among authors or analyzing the reputation of a book, author, or publisher in a specific year. Figure 5.1 illustrates a simplified graph to represent the network among scientific entities. The graph contains set of nodes (e.g., authors, papers, venues, and affiliations) and the relationships among them (e.g., author-of, published-in, affiliated-with, cited, and published-in), where any node/edge can be described by an arbitrary set of attributes (see Figure 5.1).

The discovery and communication of meaningful patterns in scientific data, i.e., analytics, can help in calculating hidden attributes of scientific entities (e.g., number of publications or citations of an author, reputation of a book or an author, or even calculating the G-index [128] and H-index [180] of an author) or understanding the

[1] http://scienceblogs.com/
[2] http://arxiv.org/
[3] http://www.myexperiment.org/
[4] https://www.box.com/
[5] Organization such as International Federation of Library Associations and Institutions (IFLA)) initiated processes to collect and organize literature and information (http://www.ifla.org/).

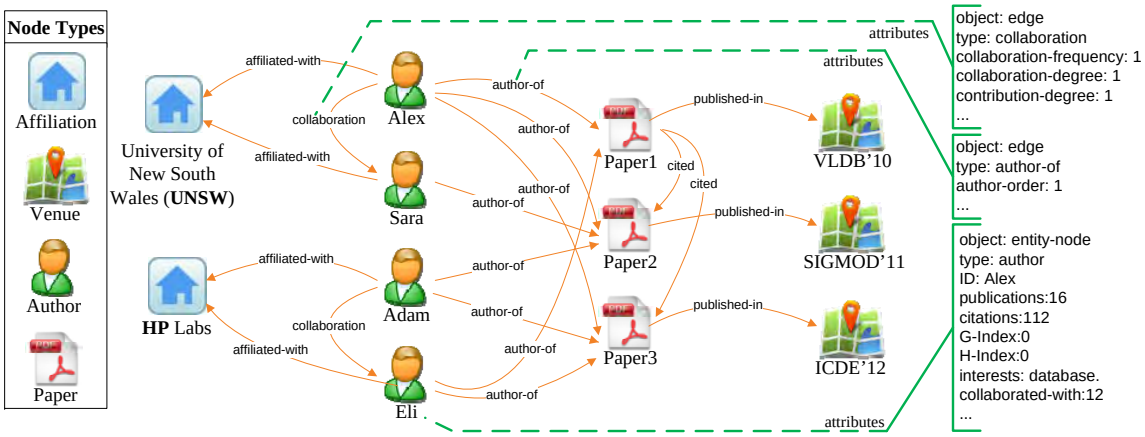


Figure 5.1: Motivating Scenario in on-line analytical processing on process graphs.

hidden relationships among scientific entities, e.g., collaboration patterns, where collaboration can be defined as a directed link between every two authors having at least one co-authored paper, and may have the following attributes:

- *frequency of collaboration*: showing how often the authors have collaborated in a given period of time, e.g., in a year.
- *degree of collaboration*: as a pairwise metric showing how the authors have collaborated in the time, e.g., the number of the papers they have co-authored divide by all the papers every one has.
- *mutual impact*: depicting how the authors have had impact on each other's publications.
- *degree of contribution*: depicting what portion of community contributions, e.g., papers, are generated by this author.

We use this example scenario to demonstrate how various users can use the GOLAP framework, for supporting decision making based on multidimensional analysis of graphs. For example, we will show how an analyst can: (i) partition scientific data into disjoint subsets, i.e., folder/path nodes; (ii) identify dimensions within each partition; and (iii) identify measures, with different grains (e.g., number of papers accepted in all venues or a specific conference) and types (e.g., additive and non/semi-additive measures which can be added across any/non/some dimensions).

5.3 Representing Analytics over Ad-hoc Process Data

5.3.1 GOLAP Data Model

We consider extending decision support on process graphs by introducing a model, i.e., GOLAP, for graph exploration and summarization. We use the data model proposed in Chapter 3 to represent online analytical processing on process graphs. In particular, GOLAP data model supports: (i) uniform representation of nodes and edges; (ii) structured and unstructured entities; and (iii) *folder* and *path* nodes, which can represent a network snapshot, i.e. a subgraph, from multiple perspectives and granularities. The set of (related) entities in a folder/path node is the result of a given query that requires grouping graph entities based on set of dimensions coming from the attributes of graph entities or the network structures, i.e., patterns among graph entities.

5.3.2 GOLAP Data Elements

Basic logical model for OLAP, presents data elements such as cubes, partitions, and dimensions to easily understand and analyze data from a multidimensional point of view. In this section, we redefine OLAP data elements, for graphs, by considering the relationships among graph entities as first class objects.

Cubes

A cube enables effective analysis of the graph data from different perspectives and with multiple granularities. We reuse and extend the definition for graph-cube proposed in [349]. In particular, given a multidimensional network N, the graph cube is obtained by restructuring N in all possible aggregations of set of node/edge attributes A, where for each aggregation A' of A, the measure is an aggregate network G' w.r.t. A'. In order to consider both multidimensional attributes and network structures, patterns among entities, into one integrated framework, we define pos-

sible aggregations upon multidimensional networks using partitions. In particular, $Q = \{q_1, q_2, ..., q_n\}$ is a set of n cubes, where each q_i is a cube, a placeholder for set of partitions, and can be modeled using folder nodes. A partition can be considered as the unit of granularity, supports multi-dimensional and multi-level views over graphs, and allows evaluation of (OLAP) operations providing a natural parallelization of execution. Three types of partitions are recognized: CC-, PC-, and Path-Partitions.

Example 1. **[CC-Partitions]** *Recall from Chapter 3 that a correlation condition ψ is a binary predicate defined on the attributes of data objects that allows to identify whether two or more entities (in a given graph) are potentially related. We use correlation conditions (CC) to partition graphs (i.e., referred in this chapter as CC-Partitions) based on set of dimensions coming from the attributes of node entities. Folder nodes can be used to represent CC-Partitions. For example, Adam, an OLAP analyst, is interested in partitioning the graph in the example scenario into a set of related entities having the same type. The correlation condition $\psi(node_x, node_y)$: $node_x.type = node_y.type$ can be defined over the attribute type of two node entities $node_x$ and $node_y$ in the graph. This predicate is true when $node_x$ and $node_y$ have the same type value and false otherwise. Related node entities will be stored in folders, where each folder can conform to an entity type and described by a set of attributes. Figure 5.2-A represents the result of this example.*

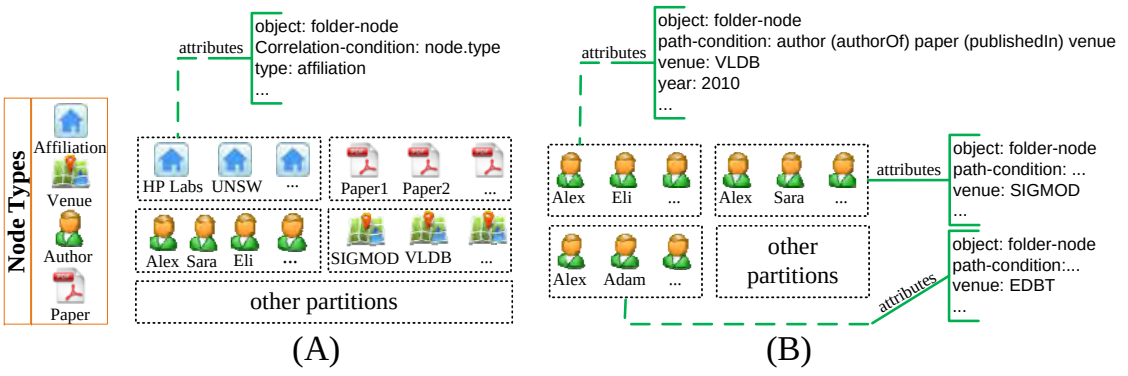


Figure 5.2: Examples of folder partitions: (A) result of Example 1; and (B) result of Example 2.

Example 2. **[PC-Partitions]** *A* path condition *(PC) can be used to group related entities in a process graph based on set of dimensions coming from the attributes of network structures. We define a path condition ϕ as a binary predicate defined on the attributes of a path that allows to identify whether two or more entities (in a given process graph) are potentially related through that path. Recall from Chapter 3 that the (transitive) relationship, in a path, can be codified using regular expressions in which alphabets are the nodes and edges from the process graph. Path conditions can be used to partition process graphs (i.e., referred in this chapter as PC-Partitions) based on set of dimensions coming from the attributes of node and edge entities. For example, Adam is interested in partitioning the graph in the example scenario into a set of related authors having at least one paper published in a specific venue. In other words, Adam is interested in: (i) creating partitions for a set of related authors; and (ii) adding authors to related partitions if: there exists the path 'author $\xrightarrow{(authorOf)}$ paper $\xrightarrow{(publishedIn)}$ venue' between the author and the venue. In this case, correlation conditions cannot be used to partition the graph as we need to apply conditions not only on graph entities but also on the relationships among them. The path condition $\phi(node_{start}, node_{end}, RE)$ can be defined on the existence of the path codified by the regular expression RE:[$author(authorOf)paper(publishedIn)venue$] between starting node, $node_{start}$, and ending node, $node_{end}$. This predicate is* true *if the path exists, and* false *otherwise. Related authors satisfying the path condition, will be added to partitions. Folder nodes can be used to represent PC-Partitions. Figure 5.2-B represents the result of this example.*

Example 3. **[Path-Partitions]** *There are cases where OLAP analysts may be interested in partitioning the graph into set of related paths. For example, Adam is interested in partitioning the graph in the example scenario into a set of related paths having the pattern 'RE: $author(authorOf)paper(publishedIn)venue$', to calculate quality metrics for venues by analyzing related papers and/or authors quality metrics. Set of related paths in a path node can be grouped, for example, by authors (Figure 5.3-A) or by venues (Figure 5.3-B). Regular expressions can be used to discover paths. Path nodes, a placeholder for a set of related paths, can be used to represent Path-Partitions.*

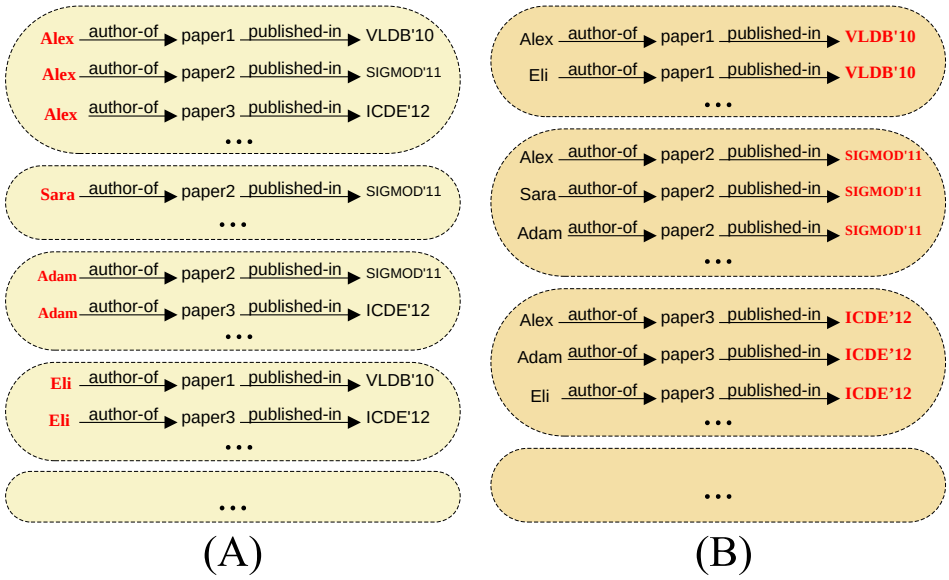


Figure 5.3: Result of Example 3 grouped by: (A) authors; and (B) venues.

Dimensions

Dimensions can be defined as perspectives used for looking at the data. In particular, $D = \{d_1, d_2, ..., d_n\}$ is a set of n dimensions, where each d_i is a dimension name. Each dimension d_i is represented by a set of elements (E) where elements are the nodes and edges of the graph. In particular, $E = \{e_1, e_2, ..., e_m\}$ is a set of m elements, where each e_i is an element name. Each element e_i is represented by a set of attributes (A), where $A = \{a_1, a_2, ..., a_p\}$ is a set of p attributes for element e_i, and each a_i is an attribute name. A dimension d_i can be considered as a given query that require grouping graph entities in a certain way. Correlation conditions and path conditions can be used to define such queries. The result of this query can be stored in folder and path nodes.

Cells

A dimension uniquely identify a subgraph within each partition, which we call a *cell*. In particular, $C = \{c_1, c_2, ..., c_n\}$ is a set of n cells, where each c_i is a cell name. Each cell can be constructed using GOLAP operations (see *operations* definition). For example, considering the motivating scenario, we may be interested in a set of dimensions coming from: (i) the attributes of graph nodes, e.g. *authors* with specific

g-index or h-index; or (ii) the attributes of graph nodes and edges, e.g. *authors* who published in specific venues or *authors* affiliated with universities in Australia during a specific time period. In order to identify cells, dimensions may have *levels* used for drilling down/up, where levels enable visiting the general/detailed view of dimensions. For example, it is important to see if the number of publications for a specific author (or group of related authors) are higher in a particular year, or drill down to see if they were higher in a particular part of the year.

Measures

Dimensions can be used as an index in order to analyze measures. A measure can be considered as numerical and computational attributes of dimensions' elements. In particular, $M = \{m_1, m_2, ..., m_n\}$ is a set of n measures, where each m_i is a measure name. For example, in the motivating scenario, number-of-publications can be considered as a measure for the element 'author' (which is a node entity) in a specific dimension. As another example, collaboration-frequency can be considered as a measure for the element 'collaboration' (which is an edge entity) in a specific dimension. Measures can be calculated by applying operations to multidimensional graph data, where operations can support UPSERT and UPDATE semantics (see *operations* definition). Three types of measures can be recognized:

- *object attributes*, are attributes of nodes and edges in the graph. The attribute for existing graph nodes, i.e., entities/folders/paths, can be considered as measure m_i. The value for existing attributes can be updated or new attributes can be added to nodes as the result of a GOLAP operation. For example, the value for the attribute 'citations' of an author can be updated/inserted during a GOLAP operation. Also, the attribute for existing graph edges can be considered as measure m_j. The value for existing attributes can be updated or new attributes can be added to edges as the result of a GOLAP operation. For example, the value for the attribute 'collaboration-frequency' of a collaboration edge between two authors can be updated/inserted during a GOLAP operation.

- *aggregated nodes*, are subgraphs including set of related nodes and relationships among them which can be considered as measure m_i. For example, considering Figure 5.1, OLAP analysts may be interested in the collaborative relationship between researchers affiliated with HP Labs and the University of New South Wales (UNSW), e.g., see Example 7. Folder and path nodes are examples of aggregated nodes, can be added to graphs during a GOLAP operation.

- *inferred edges*, are new edges which can be added between two nodes in the graph as a result of a GOLAP operation. For example, collaboration between two authors can be calculated and can be added as an attributed edge between two authors in the graph.

Operations

Typical operations on data cubes are: (i) roll-up: to aggregate data by moving up along one or more dimensions; (ii) drill-down: to disaggregate data by moving down dimensions; and (iii) slice-and-dice: to perform selection and projection on snapshots. Such operations are supported to explore different multidimensional views and allow interactive querying and analysis of the underlying data. Consequently, operations can be used for describing a computation on cells and can be ordered based on the dependencies between cells. In particular, $O = \{o_1, o_2, ..., o_n\}$ is a set of n operations, where each o_i is an operation name. Operations support UPSERT and UPDATE semantics. In order to provide a natural parallelization of execution, each operation should be evaluated independently for each partition. In GOLAP, operations can be divided into two categories:

- *assignments*: are defined to apply operations on entity attributes. Assignments support correlation between the left side (which designates target cells) and right side (which contains expressions involving cells or ranges of cells within the partition), i.e., to simulate the effect of multiple joins and UNIONs using a single access structure. For example, it is possible to apply operations on number of publications (for different group) of authors in order to rank each author, e.g., see Example 4.

- *functions*: are defined to apply set of related operations on network structures among entities. A function can be considered as a portion of SPARQL patterns (see Section 5.4) used to apply operations on the constructed partitions. For example, functions can be used to aggregate/disaggregate authors, to calculate measures such as G-index, H-index, and number-of-publication of authors, and to calculate collaboration-frequency between two authors, e.g., see Examples 5 to 8.

5.4 Querying Analytics over Ad-hoc Process Data

In Chapter 3, we discussed the shortcomings of SPARQL and introduced FPSPARQL to address some of them. In this section we extend FPSPARQL to support n-dimensional computations on graphs. In particular, on-line analytical processing on graphs requires dividing objects and relationship among them into partitions, dimensions, and measures. To model that, we extend FPSPARQL by introducing the 'GOLAP' clause. This command is used to identify partitions, dimensions, and measures on graphs. A basic GOLAP query looks like this:

```
Select <measures>
Where{
 # <existing parts of a query block>
 GOLAP{
   ?analytic [@CC|@PC|@Path-Partition] <identify partitions>.
   ?analytic @dimension <identify dimensions>.
   ?analytic @measure <identify measures>.
   <operation>; <operation>; ... <operation>;
 }
}
```

The GOLAP command is evaluated after aggregations but before ORDER BY clause. The variable ?ANALYTIC is defined to specify partitions, dimensions, and measures, where:

- *Partitions*: can be considered as: (i) CC-Partition: folder nodes which are constructed according to a correlation condition; (ii) PC-Partition: folder nodes which are constructed according to a path condition; and (iii) PATH-Partition: path nodes which are constructed according to a regular expression.

- *Dimensions*: can be defined on the attributes of set of: (i) node elements, and can be used in assignments; or (ii) node and edge elements, and can be used in functions.

- *Measures*: can be defined to identify expressions computed by GOLAP clause.

GOLAP operations can be defined using *assignments*, to apply operations on the attributes of node elements, and *functions*, to apply operations on edge attributes, aggregated nodes, and inferred edges measures. By default, the evaluation of operations occurs in the order of their dependencies. However, there are scenarios in which sequential ordering of evaluation is desired. We provide an explicit processing option, i.e. SEQUENTIAL, for that as in: $SEQUENTIAL(... < operation > ...)$. Following we explain the extension for online analytical processing on graphs through examples.

Example 4. **[nodes attributes]** *Adam is interested in partitioning the graph in the example scenario into a set of authors having same interests, where interest is an enumerated type consisting of a set of named values such as database and business process. Adam is interested in applying following operations on constructed partitions:*

1. *Update the value for the 'rank' attribute (i.e., the value for this attribute is a real number in a fixed range [0,M] where M means the highest rank) of the authors who has:*

   - *more than 200 publications to '6';*
   - *more than 200 publications and more than 1000 citations to 50% higher than the authors having same number of publications but less than 1000 citations;*

- *between 100 and 200 publications to '4';*
- *between 50 and 100 to '2';*
- *less than 50 to '1';*

2. *Insert a new attribute, 'contribution-degree', for authors within each partition. For each partition, this attribute will be calculated from division of number of publications for an authors into the sum of publications of all authors.*

In this example, all the measures come from the attributes of nodes in the graph. Following is the FPSPARQL query for this example.

```
1  Select ?ar, ?cd
2  Where
3   {
4    GOLAP{
5     ?analytic  @CC-Partition 'node[type="author"].interest'.
6     ?analytic  @dimension 'publications as ?ap, citations as ?ac'.
7     ?analytic  @measure 'rank as ?ar, contribution-degree as ?cd'.
8     #operation 1
9     update ?ar[?ap >= 200]= 6;
10    update ?ar[?ap >= 200 AND ?ac >= 1000] =
11                          ?ar[?ap > 200 AND ?ac < 1000]*1.5;
12    update ?ar[200 > ?ap >= 100]= 4;
13    update ?ar[100 > ?ap >= 50]= 2;
14    update ?ar[?ap < 50]= 1;
15    #operation 2
16    upsert ?cd[*] = ?ap / ?sum_publications;
17   }
18  AGGREGATE { (?sum_publications, SUM, {?ap} ) }
19 }
```

In this example, the variable ?ANALYTIC (lines 5 to 7) is used to define partitions, dimensions, and measures. The predicate @CC-PARTITION (line 5) parti-

tions the graph into a set of related nodes, i.e., set of authors having same interests where constructed partitions will be stored in a set of folder nodes. The NODE keyword (line 5) helps in partitioning the graph by filtering graph nodes through set of nodes attributes and their values in the bracket, e.g., in "node[type='author']", and selecting an attribute for partitioning the graph, e.g., in "node[type='author'].interest". In particular, line 5, will partition the graph into set of authors having same interest. The predicate @DIMENSION (line 6) used to define dimensions to apply further operations on constructed partitions. The predicate @MEASURE (line 7) used to define measures to be updated/upserted. In order to write operations in a simple way we support aliases (using AS statement) for dimensions and measures, e.g., ?*ap* in 'publications as ?ap'.

The Keyword UPDATE/UPSERT used to update/upsert a measure in an operation. Evaluation of operations will apply independently for each partition providing a natural parallelization of execution. An assignment can designate a single object reference (e.g., '[author-id = 2]' which designates an author whose ID is 2) or set of objects (e.g., '[publications >= 200]' which designates set of authors who has more than 200 publications). Assignments support the correlation between the left side and right side of assignments, e.g., in lines 10 and 11. Notice that to remove unnecessary computations, assignments whose results are not referenced in outer blocks will be removed automatically.

To calculate contribution-degree for each author we use aggregate functions. In order to support aggregation functions in FPSPARQL, we implemented the aggregation extension proposed in C-SPARQL [40]. Aggregation clause will be added at the end of the query, and have the following syntax:

```
AggregateClause -->
   ( 'AGGREGATE { (' var ',' Function ',' Group ')' [Filter] '}' )*
Function -->
   'COUNT' | 'SUM' | 'AVG' | 'MIN' | 'MAX'
Group    -->
   var | '{' var ( ',' var )* '}'
```

Using AGGREGATE statement we calculate sum of publications of all authors in the partition and assign this value to the variable '?sum_publications' (line 18). Notice that all aggregates are computed before evaluation of operations so they are available for the operations. Then we calculate contribution-degree, '?cd', and add it as a new attribute for all author, '?cd[*]' (line 16).

Example 5. **[inferred edges]** *Adam is interested in partitioning the graph in the example scenario into a set of related authors collaborating on (specific) papers. To achieve this, set of dimensions coming from the attributes of authors, papers and the relationship among them should be analyzed. Similar to Example 2, a path-condition can be used to partition authors. After partitioning, Adam is interested to establish a pairwise 'collaboration' edge between authors in each partition. This edge may have some attributes, where attributes can be calculated in an operation. For example, considering Figure 5.1, authors (e.g., Alex and Sara) collaborating on a paper will be correlated with a* collaboration *edge having attributes such as* collaboration-frequency*,* collaboration-degree*, and* contribution-degree*. These attributes are defined in the motivating scenario.*

This example represents, how the query language is able to establish an edge between two members of a partition as a result of a GOLAP operation. As future work, we will show that how this feature can be used for enriching crowd computing graphs [24, 25],e.g., establishing weighted edges between requesters and workers in a crowdsourcing [114] graph. Following is the FPSPARQL query for this example.

```
1  Select ?m, ?edge, ?n
2  Where{
3   # defining path condition
4   ?path-condition @regular-expression '?author (?authorOf) ?paper'.
5   ?path-condition @groupBy ?paper.
6   ?path-condition @partition-item 'distinct ?author'.
7   #
8   # defining @regular-expression variables
9   ?author @isA entityNode. ?author @type author.
```

```
10  ?authorOf @isA edge. ?authorOf @type author-of.
11  ?paper @isA entityNode. ?paper @type paper.
12  #
13  GOLAP{
14   ?analytic  @PC-Partition ?path-condition.
15   ?analytic  @dimension '?m, ?n'.
16    #dimensions(s) are defined in the function!
17   ?analytic  @measure '?edge'.
18    #measure(s) are defined in the function!
19   #
20   function.F1;
21  }
22 }
23
24 functions{
25  F1{
26   ?m @isA entityNode. ?m @type author. ?m @name ?m_name.
27   ?n @isA entityNode. ?n @type author. ?n @name ?n_name.
28   correlate{
29    (?m,?n,?edge,FILTER(?m_name <> ?n_name))
30    ?edge @isA edge. ?edge @type 'collaboration'.
31    ?edge @collaboration-frequency '0'.
32    ?edge @collaboration-degree '0'.
33    ?edge @contribution-degree '0'.
34   }
35  }
36 }
```

In this example, the predicate @PC-PARTITION (line 14) is used to partition the graph into a set of related authors, where each partition contains authors of a specific paper. This condition is generated through the regular expression

'$author(authorOf)paper$' (line 4), where variable $?path - condition$ is used to define the path-condition attributes. Also set of predicates used to define the regular expression (@REGULAR-EXPRESSION in line 4), grouping selected paths (@GROUPBY in line 5), and defining the items to be added to constructed partitions (@PARTITION-ITEM in line 6). Moreover, The DISTINCT keyword can be used to add distinct (different) values to the partition. The second block of codes in this example (lines 9 to 11), defines the elements of regular expression such as $?author$, $?authorOf$, and $?paper$.

To construct an attributed edge between two nodes in partitions we use functions. The FUNCTIONS keyword (line 24) is used to define a block of functions. Each function defined by a name (e.g., function F1) and a block of SPARQL patterns. Defined functions can be called using FUNCTION keyword (line 20) following by a full stop and function name (e.g., 'function.F1'). In this example, function F1 defined to establish a collaboration edge among authors. In Chapter 3, we introduced CORRELATE statement to establish a directed edge between two nodes in a graph. As a reminder, a basic correlation condition query looks like this:

```
correlate {
   (entity1, entity2, edge1, condition)
   pattern1.
   pattern2.
   ...
}
```

As a result, $entity_1$ will be correlated to $entity_2$ through a directed edge $edge_1$ if the condition evaluates to true. Patterns can be used for specifying the edge attributes. In function F1 (lines 25 to 35), variables $?m$ and $?n$ represent authors. The condition in *correlate* statement (i.e., '$?m_{name} <> ?n_{name}$') makes sure that only two different authors will be connected. For simplicity reason, in this example we assign a constant value for collaboration edge attributes. Next example shows how attributes of the collaboration edge can be calculated dynamically.

Example 6. [**edges attributes**] *Adam is interested in calculating the degree of collaboration in Example 5. In section 5.2, degree of collaboration defined as a pairwise metric showing how the authors has collaborated in time, e.g., the number of papers they have coauthored divided by all the papers every one has. Notice that this attribute may have different values over time. Following, we revise function F1 in Example 5, in order to calculate the collaboration degree between authors.*

```
1  F1{
2   ?m @isA entityNode. ?m @type author. ?m @name ?m_name.
3   ?n @isA entityNode. ?n @type author. ?n @name ?n_name.
4   ?m @publications ?pubs.
5   #
6   ?edge @isA edge. ?edge @type 'collaboration'.
7   ?edge @numOfCoauthoredPapers ?numCoPapers.
8   ?edge @collaboration-degree ?clbDegree.
9   #
10  optional{
11   BIND (?numCoPapers+1 AS ?numCoPapers).
12   BIND (?numCoPapers/?pubs AS ?clbDegree)
13  } FILTER EXISTS { ?m ?edge ?n }
14  #
15  optional{
16   correlate{
17    (?m,?n,?edge,FILTER(?m_name <> ?n_name))
18    BIND (1 AS ?numCoPapers). #BIND assign a value to a variable
19    BIND (?numCoPapers/?pubs AS ?clbDegree)
20   }
21  } FILTER NOT EXISTS { ?m ?edge ?n }
22 }
```

In this example, partitions will be constructed similar to Example 5, where each partition contains authors of a specific paper. Considering that the operations are

evaluated for one partition at a time, to calculate the number of coauthored papers between two authors it will be enough to follow two steps: (a) if the collaboration edge exists, increase the value for coauthored-papers attribute by 1; and (b) if the collaboration edge does not exist between authors, construct the edge and set the coauthored-papers attribute to 1. Variables ?m and ?n represent authors in the partitions and defined in lines 2 to 4, and variable ?*pubs* represents the number of publications for author ?m. The collaboration edge, i.e., ?*edge*, is defined in lines 6 to 8, where variable ?$numCoPapers$ represents number of coauthored papers between authors ?m and ?n. The variable ?$clbDegree$ (line 8) represents the collaboration degree between authors ?m and ?n and will be calculated by '?numCoPapers/?pubs', i.e., number of papers they have coauthored divided by all the papers every one has.

There are two optional statements used in this function. The first optional statement (lines 10 to 13) used to updated the number of coauthored papers and recalculating the collaboration degree if the edge exists between authors ?m and ?n. Notice that in SPARQL, the EXISTS and NOT EXISTS statements are used based on testing whether a pattern exists/not-exists in the graph. Moreover, in SPARQL, the BIND statement allows a value to be assigned to a variable in a group graph pattern. The second one (lines 15 to 21) used to construct the collaboration edge ?*edge* between authors ?m and ?n if the edge does not exists between them. In this case, the edge will be constructed (line 17), the number of coauthored papers will be set to 1 (line 18), and the collaboration degree will be calculated (line 19).

Example 7. **[aggregated nodes]** *Adam is interested in the collaborative relationship between researchers affiliated with affiliations, e.g., HP Labs and UNSW. To achieve this, he partitioned the graph in the example scenario into a set of related authors affiliated with specific affiliations by setting the regular expression to 'author (affiliated-with) affiliation'. Then he set the group-by attribute to 'affiliation' and put related authors in partitions. This example shows how partitions can be stored (e.g., as a folder node in this example) and added to the graph as a result of a GOLAP query. Notice that such aggregated nodes can have set of descriptive attributes. This*

way Adam will be enable to construct relationships (e.g., 'collaboration' edge) among aggregated nodes in the graph. Following is the FPSPARQL query for this example.

```
1  Select ?aff_title
2  Where{
3   # defining path condition
4   ?path-condition @regular-expression
5                 '?author (?affiliated-with) ?affiliation'.
6   ?path-condition @groupBy ?affiliation.
7   ?path-condition @partition-item 'distinct ?author'.
8   #
9   # defining @regular-expression variables
10  ?author @isA entityNode. ?author @type author.
11  ?affiliated-with @isA edge.
12  ?affiliated-with @type affiliated-with.
13  ?affiliation @isA entityNode. ?affiliation @type affiliation.
14  ?affiliation @title ?aff_title. #e.g., UNSW or HP Labs.
15  #
16   GOLAP{
17    ?analytic  @PC-Partition ?path-condition.
18    ?analytic  @dimension '?author, ?aff_title'.
19     #dimensions(s) are defined in the function!
20    ?analytic  @measure '?folderNode'.
21     #measure(s) are defined in the function!
22
23    function.F1;
24   }
25 }
26
27 functions{
28  F1{
29   BIND (?folderNode as ?aff_title) #?aff_title defined in Line14
```

```
30   fconstruct ?folderNode
31    select * where {
32    ?folderNode @description 'set of ...'.
33    ?folderNode @numberOfAuthors ?authorsCount.
34    # other patterns.
35   }
36   AGGREGATE { (?authorsCount, COUNT, {?author} ) }
37  }
38 }
```

In this example, the predicate @PC-PARTITION (line 17) is used to partitions the graph into a set of related authors affiliated with specific affiliations. This condition is generated through the regular expression 'author (affiliated-with) affiliation' (lines 4 and 5). The variable *?path-condition* is used to define the path-condition attributes, i.e., @REGULAR-EXPRESSION, @GROUPBY, and @PARTITION-ITEM. The second block of codes (lines 9 to 14), defines the elements of regular expression such as *?author*, *?affiliated-with*, and $?affiliation$. Then the GOLAP statement will partition the graph. Finally the function F1 will apply on all partitions. As a result, each partition will be stored in a folder node and will be added to the graph as an aggregated node.

In Chapter 3 we introduced the FCONSTRUCT command which is used to group a set of related entities or folders. As a reminder, a basic folder node construction query looks like this:

```
fconstruct <Folder_Node Name>
[select ?var1 ?var2 ... | (Folder_Node1 Name, Folder_Node2 Name,...)]
where {
   pattern1.
   pattern2.
   ...
}
```

In this example we use FCONSTRUCT command, in function F1, in order to store each partition as a folder node and add them to the graph. We set the name of the folder node ($?folderNode$) as the affiliation title ($?aff\_title$) by binding its value to the folder to be constructed. For example, if function F1 apply on the partition which contains set of authors affiliated with UNSW, then the variable $?aff\_title$ will contain 'UNSW' (line 14), and consequently the name of the folder to be constructed will be 'UNSW'. Notice that the folder node can have set of descriptive attributes, e.g., description and number of authors (lines 32 to 34). The value for these attributes can be calculated dynamically, e.g., number of authors.

Next, Adam can use a function, similar to Example 5, to construct relationships (e.g., 'collaboration' edge with attributes such as *collaboration-frequency*, *collaboration-degree*, and *contribution-degree*) between aggregated nodes. Recall from Chapter 3 that we introduced the PCONSTRUCT command to construct path nodes. In particular, in scenarios similar to Example 7, FCONSTRUCT can be used to store CC-Partitions and PC-Partitions in folder nodes and PCONSTRUCT can be used to store Path-Partitions in path nodes.

Example 8. **[path partitions]** *Adam is interested in partitioning the graph in the example scenario into a set of related paths having the pattern 'RE: author (authorOf) paper (publishedIn) venue', and group by authors. Figure 5.3-A illustrates such partitions. As next step, Adam is interested in:*

- *Update the number of publications for each author. This query can be done by counting papers in each partition.*
- *Update the number of citations for each author. This can be done by calculating the summation of all papers' citations.*
- *Calculate ERA (Excellence in Research for Australia) rank for each author. For example, consider that in ERA ranking, papers published in venues ranked: (i) 'A*' have 4 points; (ii) 'A' have 1 point; (iii) 'B' have 0 point; and (iv) 'C' have -1 point.*

Following is the FPSPARQL query for this example.

```
1  Select ?pub, ?paperCitation, ?authorCitation, ?ERA
2  Where{
3   # defining path condition
4   ?path-node @regular-expression
5              '?author (?authorOf) ?paper (?publishedIn) ?venue'.
6   ?path-node @groupBy ?author.
7   #
8   # defining variables used in the regular expression
9   ?author @isA entityNode. ?author @type author.
10  ?authorOf @isA edge. ?authorOf @type author-of.
11  ?paper @isA entityNode. ?paper @type paper.
12  ?publishedIn @isA edge. ?publishedIn @type published-in.
13  ?venue @isA entityNode. ?venue @type venue.
14  ?paper @citations ?paperCitation
15  ?author @publication ?pub.
16  ?author @citation ?authorCitation.
17  ?author @ERA_Ranking ?ERA.
18  ?venue @isA entityNode. ?venue @type venue.
19  #
20   GOLAP{
21    ?analytic  @Path-Partition ?path-condition.
22    ?analytic  @dimension '?author, ?paper, ?venue'.
23    #dimensions are defined in lines 9 to 18!
24    ?analytic  @measure '?pub,?paperCitation,?authorCitation,?ERA'.
25    #measures are defined in lines 9 to 18!
26
27    function.F1;
28    function.ERA;
29   }
30 }
31
```

```
32 functions{
33  F1{
34   #1-update number of publications for each author.
35   update ?pub[*] = ?numberOfPapers;
36
37   #2-update number of citations for each author.
38   update ?authorCitation[*] = ?allCitations;
39
40   AGGREGATE { (?numberOfPapers, COUNT, {?paper} ) }
41   AGGREGATE { (?allCitations, SUM, {?paperCitation} ) }
42  }
43
44  ERA{
45   #4-calculating ERA ranking
46   update ?ERA[*] = (?numOfTopA * 4)+(?numOfA * 1)-(?numOfC * 1);
47
48   AGGREGATE { (?numOfTopA, COUNT, {?paper} )
49     FILTER (?paper published-in ?venue. ?venue @Rank = 'A*') }
50   AGGREGATE { (?numOfA, COUNT, {?paper} )
51     FILTER (?paper published-in ?venue. ?venue @Rank = 'A') }
52   AGGREGATE { (?numOfC, COUNT, {?paper} )
53     FILTER (?paper published-in ?venue. ?venue @Rank = 'C') }
54  }
55 }
```

In this example, parameter @PATH-PARTITION (line 21) is used to partition the graph into set of path nodes. The regular expression for path nodes is defined in the parameter @REGULAR-EXPRESSION (line 4 and 5). Predicate @GROUPBY is used to group discovered paths by authors (line 6). The second block of code (lines 8 to 18) defines the variables used in the regular expression. The GOLAP statement (lines 20 to 29) defines dimensions and measures to be used in functions

F1 and ERA. In function F1, aggregates used to calculate the number of papers (line 40) and sum of citations (line 41) for the authors. The assignment in line 35 will update the number of publications, ?*pub*, for all authors, ?*pub*[∗]. The assignment in line 38 will update the number of citations, ?*authorCitation*, for all authors, ?*authorCitation*[∗]. In function ERA, aggregates used to calculated number of papers published in venues ranked 'A*', 'A', and 'C'. Note that, venues ranked as *B* will not be calculated as they have zero points. The assignment in line 46 will update the ERA ranking, ?*ERA*, for all authors, ?*pub*, for all authors, ?*ERA*[∗].

5.5 Architecture and Implementation: Analytics Extension

5.5.1 Architecture

In Chapters 3 and 4, we introduced FPSPARQL graph processing architecture, where the architecture consists of the following components: Graph Loader, Data Mapping Layer, Query Mapping Layer, Regular Expression Processor, External Algorithm/Tool Controller, Time-aware Controller, and Query Optimizer. In this chapter, we instrument the Query Optimizer component, to support the better performance of analytics queries, and add a new component, i.e., GOLAP Controller, to support the execution of FPSPARQL analytics queries. In particular, GOLAP Controller is responsible for partitioning graphs and allows evaluation of OLAP operations on graphs independently for each partition, providing a natural parallelization of execution. Figure 5.4 illustrates FPSPARQL graph processing architecture. In next section we describe the GOLAP Controller and the optimization techniques applied to analytics queries.

5.5.2 Analytics Queries Execution and Optimization

Figure 5.5 illustrates the GOLAP Controller, classifies the evaluation order and dependency analysis of analytics queries, and identifies the execution algorithm. In

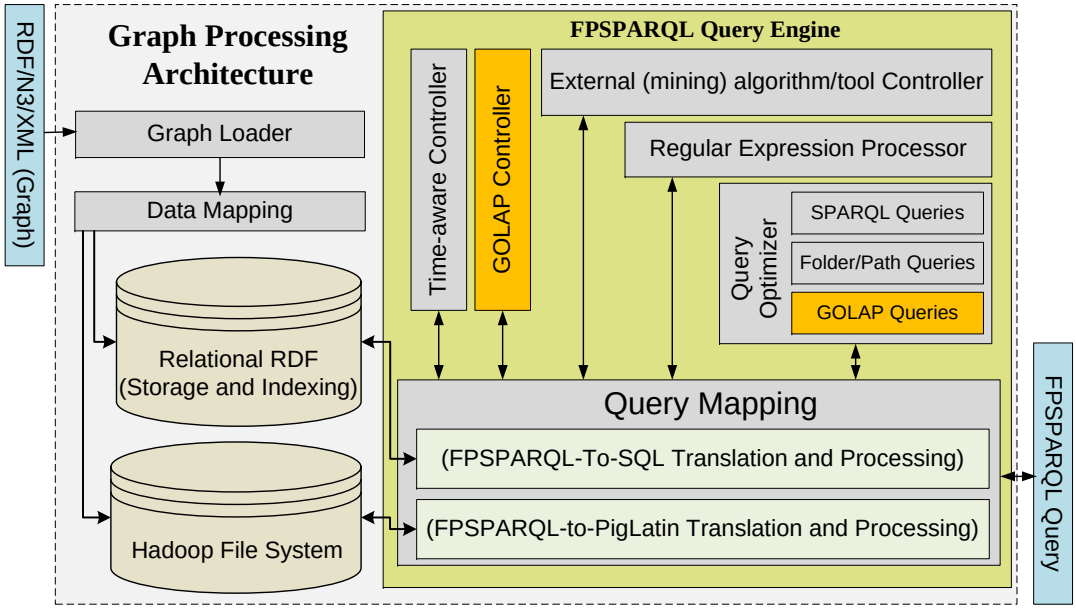


Figure 5.4: FPSPARQL graph processing architecture: analytics extension.

particular, the analytics query will be evaluated using query mapping layer. This layer is consist of a FPSPARQL parser for parsing FPSPARQL queries based upon the syntax of FPSPARQL. Next step is to analyze the partitions. The partitioning of the graph provides an obvious way to parallelize OLAP operations on graph partitions and provide scalability. For CC-Partitions, dimensions will be specified and query filters will be generated to partition the graph based on entity attributes. For PC-Partitions and Path-Partitions, in order to convert path queries, regular expressions will be analyzed and path-based indexing will be applied to the graph. We use an existing path-based indexing approach (i.e., gIndex [337]) for partitioning the graph, based on patterns among graph entities. We use gIndex to build path indices, in order to help processing path conditions and constructing path partitions. Moreover, to eliminate cycles, we applied the cycle elimination technique proposed in [158, 353].

As discussed in Chapter 3, FPSPARQL supports *aggregate* queries and *keyword* search queries. To help users conduct partitioning on graphs and to enhance the capability of the keyword search technique on triple tables, we develop the aggregate keyword search method which finds aggregate groups of entities jointly matching a set of query keywords, i.e., both for folder and path nodes. Moreover, for efficient access to single cells we built a partition level hash access structure. In particular,

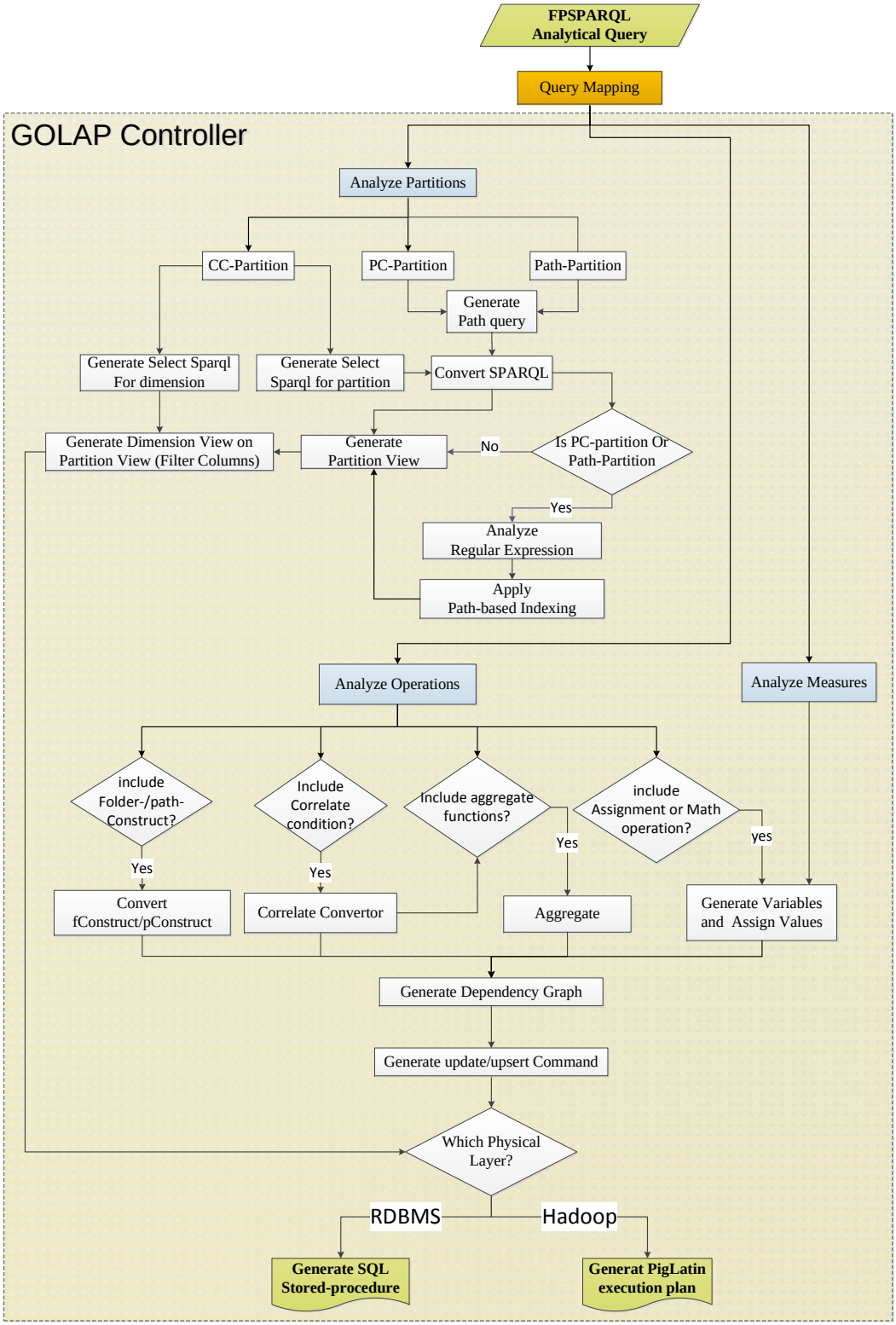


Figure 5.5: Execution plan for FPSPARQL analytics queries.

data is hash partitioned on '@CC-PARTITION', '@PC-PARTITION', and '@PATH-PARTITION' parameters. Moreover, for CC-Partitions and PC-Partitions, a hash table will be built on the dimensions. For Path-Partitions, two hash tables will be built, one for nodes in path partitions and the other for relationship edges. Consequently, we avoid spilling to disk for evaluating the operations: the partitions will be kept in memory and the operations will evaluated for one partition at a time.

After analyzing dimensions and constructing partitions, operations and measures need to be evaluated. For example, for functions, it will be checked if they contain aggregation nodes (folder/path construction), correlate statement, and aggregate functions. All the aggregates at any level needs to be computed before evaluation of operations at that level. This requires a scan of entities in the partition for each level. Moreover, with each operation we store a list of aggregates dependent on the cell being upserted (or updated) by it. Next step is to analyze measures, generate update/upsert commands, and determine the order of evaluation of operations. In order to apply operations, the order of evaluation of operations is determined from their dependency graph. The dependency graph is a weighted, directed graph where each vertex corresponds to one operation and each edge of the graph has a weight e_{ij} representing the strength of the dependency between vertex i and vertex j. Edge weights $\{e_{ij}\}$ are real numbers: the larger the e_{ij} is, the stronger the dependency is between vertex i and vertex j. To construct the dependency graph, a pairwise dependency metric [342] is calculated for each pair of vertexes.

In particular, given a set of operations O and a transitive relation $R = O \times O^{with}(a, b) \in R$ modeling a dependency "a needs b evaluated first", the dependency graph will be the graph $G = (O, T)$ with $T \subseteq R$ and R being the transitive closure of T. For example, given a several assignments like "i-Index = sum(publications)*c-measure; c-measure = avg(citations)", then 'i-Index' depends on 'c-measure' which should be calculated before 'i-Index'. We derive an evaluation order, or the absence of an evaluation order, that respects the given dependencies from the constructed dependency graph. Moreover, we use the dependency graph to identify the operations that can be *pruned*, e.g., the evaluation of an assignment becomes unnecessary when the cells it updates are not used in the evaluation of other assignments.

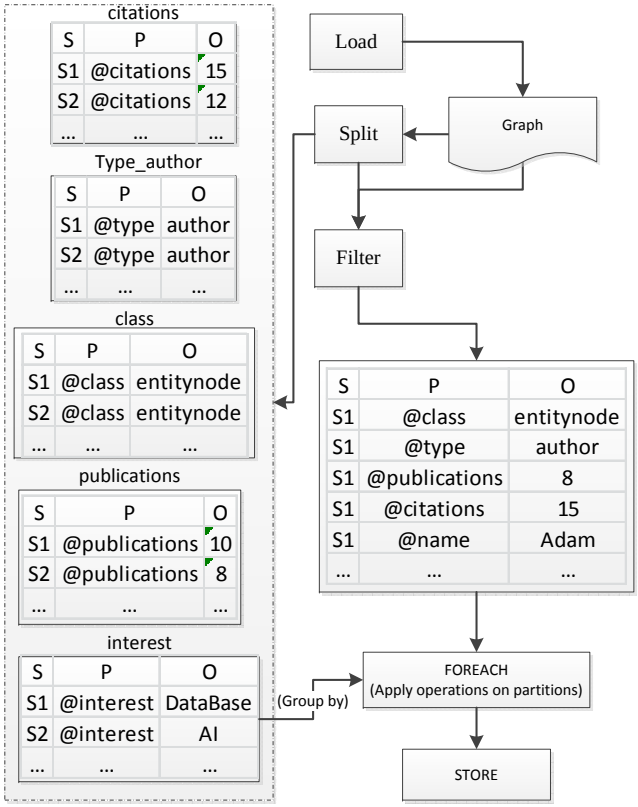


Figure 5.6: Execution plan for the query in Example 4.

After parsing the query, two execution plans will be available: RDBMS and Hadoop. As discussed in Chapter 3, two types of storage back-end are supported in FPSPARQL: Relational Database System and Hadoop File System. In order to support RDBMS execution plan, the output for the execution plan in Figure 5.5 will be a SQL stored procedure. Recall from Chapter 3 that Apache Pig is a language for querying large semi-structured datasets using Hadoop and the MapReduce Platform, and consists of a textual language called Pig-Latin which supports ease of programming, optimization opportunities, and extensibility. For the Hadoop execution plan, the output will be a set of Pig-Latin scripts: a script in Pig often follows a specific format in which data is read from the Hadoop file system, a number of operations (e.g., LOAD, SPLIT, JOIN, FILTER, and STORE) are performed on the data, and then the resulting relation is written back to the file system.

As an example, we discuss the Hadoop query plan for the Example 4. Processing this query using Pig Latins query algebra results in the query plan shown in Figure 5.6. The logical plan can be described as follows: (1) load the input dataset

using the LOAD operator; (2) split the dataset, based on the partitioning condition, and create triple tables for related predicates. Next step is to filter the dataset into related authors, where the 'interest' triplestore will be needed for the partitioning phase and 'publications' and 'citations' triplestores will be needed to apply OLAP style operations on partitions; (3) filter the graph using the result of previous step, i.e., to support the triple syntax and weave the predicates to related partitions. Notice that, in the case of using JOIN operator in this step, the triple syntax will be no longer available; (4) group by the interest table on the object column to remove redundant values, e.g., in cases where two or more authors, different subjects, having same interests; (5) evaluation of OLAP operations on graphs independently for each partition, providing a natural parallelization of execution; and (6) store the final result on Hadoop cluster using the STORE operator.

5.5.3 Implementation

Details about FPSPARQL query engine implementation is presented in Chapter 3. In this Chapter, we instrument the query engine with GOLAP Controller, which is responsible to support n-dimensional computations on graphs. Moreover, we have implemented a front-end tool for assisting users to create GOLAP queries, by providing an easy way to create partitions, select dimensions, and define measures. To create CC-partitions, we provide list of graph objects (i.e., nodes and edges) and their attributes. To create PC/Path-Partitions, we provide an interface to easily create regular expressions to be used in path conditions. Figure 5.7-B illustrates how we created the regular expression and the path condition for the Example 5. We use frequent pattern discovery approaches to provide users with frequent patterns, which can help in identifying groups of related paths. We have created an interface to assist users writing and debugging operations. Figure 5.7-A illustrates how we use the tool to write functions for the Example 8. Finally, we provided users with a graph visualization tool for the exploration of partitions and query results.

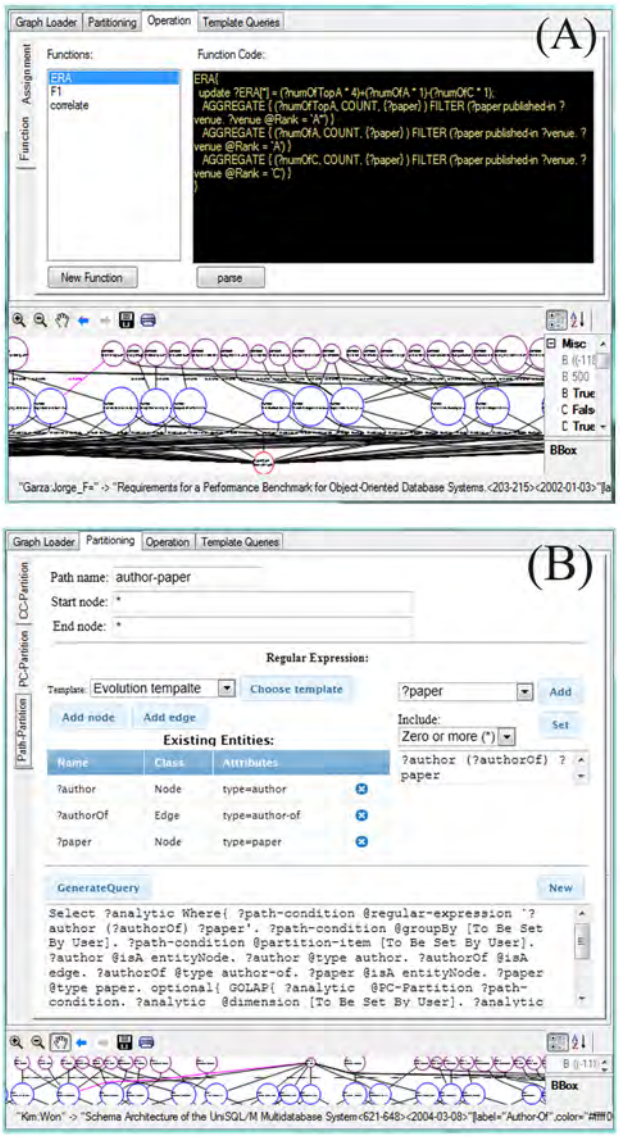


Figure 5.7: Screenshots of front-end tool for: (a) writing functions in Example 8; and (b) creating the regular expression and the path condition in Example 5.

5.6 Experiments

5.6.1 Datasets

We carried out the experiments on two graph datasets: DBLP[6] and Amazon Online Rating System[7].

[6]http://dblp.uni-trier.de/db/
[7]http://snap.stanford.edu/data/amazon-meta.html

DBLP

DBLP (Digital Bibliography and Library Project) dataset, is a computer science bibliography database which listed more than 1.8 million publications (in May 2012) and tracked most of journals and conference proceedings. We use DBLP to generate graph models containing set of nodes (e.g., paper, author, venue, and affiliation) and edges (e.g., author-of, published-in, and affiliated-with). In order to enrich the DBLP graph, and generate the graph introduced in the example scenario, we added attributes to graph nodes and edges. For example, we enriched nodes typed as 'author' with attributes such as type and number of publications and citations. As another example, we enriched edges typed as 'author-of' with attributes such as order of author in the paper and temporal attributes. Moreover, we used FPSPARQL to construct edges among nodes of the DBLP graph. For example, we added edges typed as 'collaboration' between pair of authors. This edge and its attributes (e.g., frequency of collaboration, degree of collaboration, mutual impact, and degree of contribution) have been introduced in the example scenario. The generated graph from DBLP dataset contains over 1,670,000 nodes (i.e. authors, papers, venues, and affiliations) and over 2,810,000 edges (i.e., author-of, published-in, affiliated-with, cited, and published-in).

Amazon Online Rating System

Amazon is one of the most popular online rating systems. One application of crowdsourcing [114] systems is evaluating and rating products on the Web which refereed as Online Rating Systems. In an online rating system: (i) producers or sellers advertise their products; (ii) customers rate them based on their interests and experiences; and (iii) the rating system aggregates scores received from customers to calculate a general rating score for every product. In this experiment, we use the rating log of Amazon online rating system that are collected by Leskovec et al. [220] (i.e., referred in the following as AMZLog) for analyzing dynamics of viral Marketing by applying online analytical processing on this graph dataset. Online rating graphs may contain some types of nodes (i.e., product, reviewer, category) and edges (e.g.,

Dataset statistics	
Products	548,552
Product-Project Edges	1,788,725
Reviews	7,781,990
Product category memberships	2,509,699
Products by product group	
Books	393561
DVDs	19828
Music CDs	103144
Videos	26132

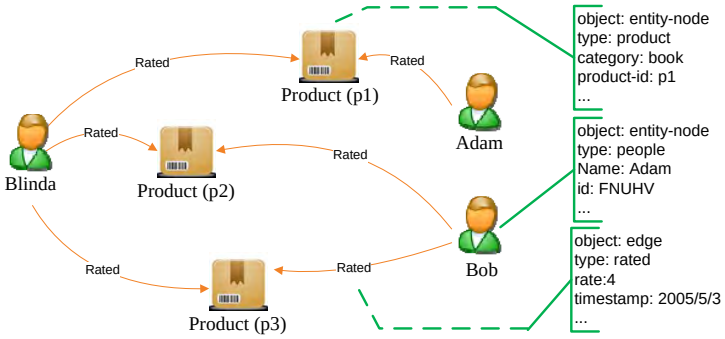


Figure 5.8: A sample of data stored in AMZLog.

hasRated in 'aleks hasRated book1'). Figure 5.8 illustrates a sample of data stored in AMZLog and the dataset statistics. We enriched Amazon graph by Adding attributes to nodes. For example, we enriched nodes typed as 'product' with (i) *rank in category*, to show how popular is the product in a particular category, e.g., a book among all available books; and (ii) *overall rank*, to show the overall rank of a product in the system. We also enrich the 'reviewers' by adding: (i) *degree of interest*, to show in what extent a reviewer is interested in a particular category of products, e.g., books, movies or albums; and (ii) *degree of expertise*, to show how dependable are the ratings that a reviewer has given to a particular product.

5.6.2 Evaluation

We report the evaluation results of the query engine extension, GOLAP, in terms of: (i) *performance*: we report performance in terms of running time of queries in seconds; (ii) *scalability*: we report scalability by characterizing how the performance of each query changes as the size of the graph increases; and (iii) *quality of results*: the quality of the results is assessed using classical *precision* metric which defined as the percentage of discovered results that are actually interesting. Moreover, we compare the performance of queries with and without query optimization techniques discussed in Section 5.5.2. Notice that, the performance and effectiveness of FPSPARQL query engine have been represented in Chapter 3. To the best of our knowledge, we couldn't find any open source system that support similar functionalities of FPSPARQL, analytics extension, to compare with.

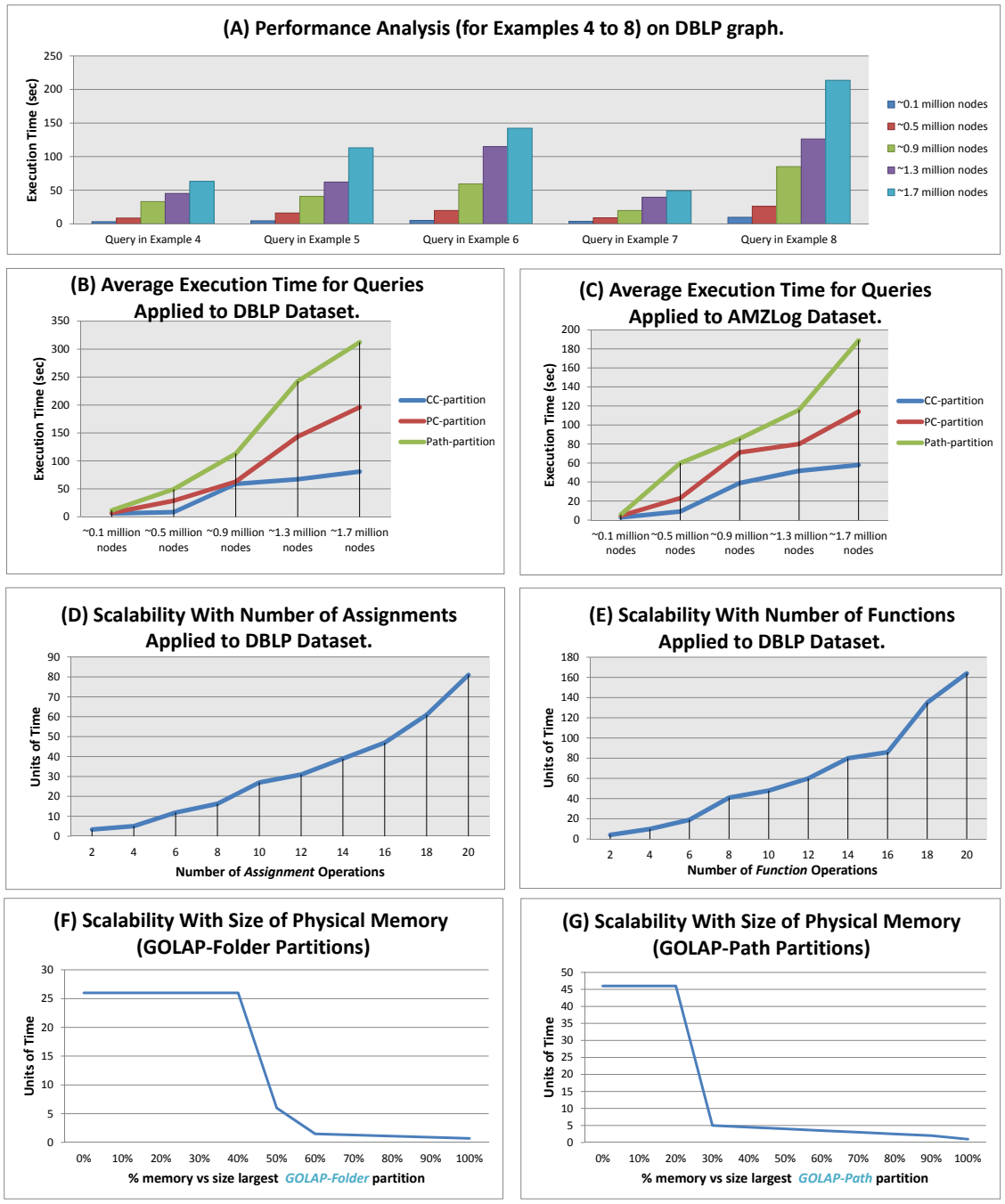


Figure 5.9: The evaluation results, illustrating: (A) performance analysis (for queries in Examples 4 to 8) applied to the DBLP graph; (B) average execution time for 10 CC-Partition (blue line), 10 PC-Partition (red line), and 10 Path-Partition (green line) queries applied to different sizes of DBLP graph dataset; (C) average execution time for 10 CC-Partition (blue line), 10 PC-Partition (red line), and 10 Path-Partition (green line) queries applied to different sizes of AMZlog graph dataset; (D) scalability with number of *assignment* operations for 10 queries applied to DBLP dataset; (E) scalability with number of *function* operations for 10 queries applied to DBLP dataset; (F) scalability with size of physical memory for CC-Partitions and PC-Partitions; and (G) scalability with size of physical memory for Path-Partitions.

Performance. We report performance in terms of running time in seconds. We applied optimization techniques (see Section 5.5.2) on DBLP and AMZLog datasets. The optimization took 62 minutes for DBLP graph dataset and 41 minutes for AMZLog graph dataset. In the first step, we executed queries in Examples 4 to 8 on DBLP graph dataset. Figure 5.9-A illustrates the execution times for these queries. As illustrated in this figure, we divided each dataset into regular number of graph nodes (e.g., 0.1, 0.5, 0.9, 1.3, and 1.7 million nodes) and ran the experiment for different sizes of DBLP graph dataset. We sampled DBLP graph according to venues and AMZlog graph according to products, to guarantee the properties of the sampled graphs. The evaluation shows a polynomial (nearly linear) increase in the execution time of the queries in respect with the dataset size. Recall from Section 5.5.2 that the partitions will be kept in memory and the operations will evaluated for one partition at a time.

In the second step, we provided 10 CC-Partition, 10 PC-Partition, and 10 Path-Partition queries (each having one operation) for DBLP dataset. We provided similar queries for AMZLog dataset. These queries were generated by our colleagues who are expert in these domains and familiar with the proposed datasets. Figures 5.9-B and -C show the average execution time for applying the queries to DBLP and AMZLog datasets respectively. As illustrated in this figure, we divided each log into regular number of graph nodes (same sizes for both DBLP and AMZlog graph) and ran the experiment for different sizes of graph datasets. In particular, the performance for applying queries on AMZlog is better, as SPARQL graph pattern matching is dominated by several join operations, and is unlikely to be efficiently processed when the type of nodes and edges in the graph increased, i.e., AMZlog dataset contains two types of nodes and one type of relationships and DBLP dataset contains four types of nodes and five types of relationships. Finally, in Figures 5.10 we compare the performance of queries applied to DBLP dataset (i.e., CC/PC/Path-Partition queries in Figure 5.9-A) with and without query optimization.

Scalability. Another key measure of effectiveness of a query is its scalability. In our case, the key question to ask is how does each of the queries perform when the size

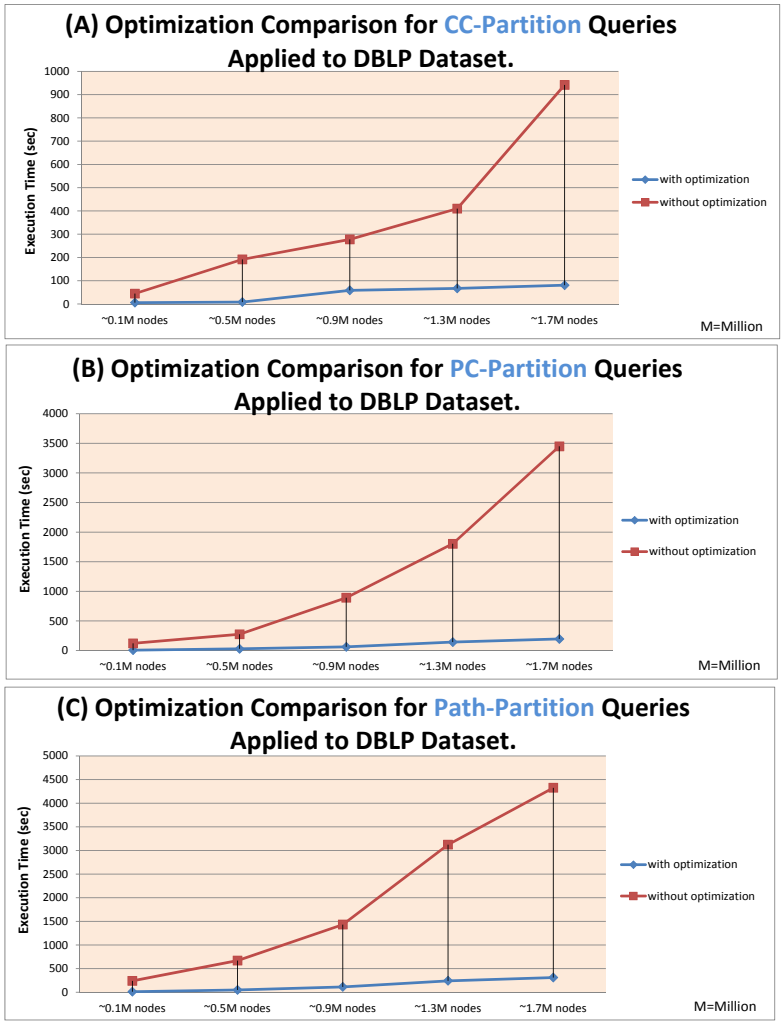


Figure 5.10: The query optimization results, illustrating optimization comparison for CC-Partition (A), PC-Partition (B), and Path-Partition (C) queries applied to DBLP dataset.

of the graph increases? As mentioned earlier, we divided each dataset into regular number of graph nodes and sampled the graphs carefully to guarantee the properties of the samples. Figures 5.9-A, -B, and -C shows an almost linear scalability between the response time of queries and the number of nodes in the graph. As another scalability metric we increased the number of *assignment* operations for 10 queries applied to DBLP dataset. Figure 5.9-D shows an almost linear scalability between the average response time of queries and the number of assignment operations. Moreover, we increased the number of *function* operations for 10 queries applied to DBLP dataset. Figure 5.9-E shows an almost linear scalability between the average response time of queries and the number of function operations.

Figures 5.9-F and -G show the performance of our access structure as a function of available memory for folder-node partitions (CC-Partitions and PC-Partitions) and path-node partitions (Path-Partitions) respectively. The memory size is expressed as a percentage of the size required to fit the largest partition of data in the hash access structure in physical memory. Recall from Section 5.5.2 that for efficient access to single cells we built a partition level hash access structure where the partitions will be kept in memory and the operations will evaluated for one partition at a time. In the experiment for folder-node partitions, we execute a single assignment, "$update?ar[?ap >= 200 AND ?ac >= 1000] = ?ar[?ap > 200 AND ?ac < 1000] * 1.5;$", from the query in Example 4. In the experiment for path-node partitions, we execute a single function, ERA, from the query in Example 8. In particular, if a partition does not fit in memory we incur an I/O if a referenced cell is not cached. In the case of folder-node partition (Figures 5.9-F), this occurs when the available memory is less than 40% of the largest partition, and for the path-node partition (Figures 5.9-G) this occurs when the available memory is less than 20% of the largest partition.

Quality. The quality of the results is assessed using classical *precision* metric which defined as the percentage of discovered results that are actually interesting. For evaluating the interestingness of the result, we asked domain experts who have the most accurate knowledge about the dataset to construct GOLAP queries on graphs. For each query they codified their knowledge to use correlation/path conditions, construct regular expressions that describe paths through the nodes and edges in the graph, and describe dimensions and measures. They used the front-end tool to construct GOLAP queries, visualize the content of constructed partitions, analyze discovered paths (using path conditions), and identify the query results to see what they consider relevant from an OLAP perspective. The quality evaluation applied to both DBLP and AMZLog datasets, where 18 queries constructed. For each dataset 9 queries constructed: three CC-Partition, three PC-Partition, and three Path-Partition queries, in which three queries applied to entity attributes, three queries applied to aggregated nodes, and three queries applied to inferred edges measures. As a result, all partitions and query results (e.g., updated/upserted measures) examined by domain experts and all considered relevant.

Performance Comparison Between RDBMS and Hadoop Execution Plans. As mentioned earlier, FPSPARQL queries can be run on two types of storage backend: RDBMS and Hadoop. In this part we compare the performance of query plans on relational triplestores and Hadoop file system. All experiments in this part were conducted on a virtual machine, having 32 cores and 192GB RAM. Figure 5.11-A illustrates the performance analysis between RDBMS and Hadoop for queries in Examples 4 to 8 applied to the DBLP graph. Figure 5.11-B illustrates the performance analysis, for the average execution time, between RDBMS and Hadoop for the 10 CC-Partition, 10 PC-Partition, and 10 Path-Partition queries (these queries are discussed earlier in this section) applied to the DBLP graph. Figure 5.11-C illustrates the performance analysis, for the average execution time, between RDBMS and Hadoop for the 10 CC-Partition, 10 PC-Partition, and 10 Path-Partition queries (these queries are discussed earlier in this section) applied to the AMZlog graph.

Discussion. As illustrated in Figure 5.9 we divide each dataset into regular number of nodes and ran the experiment for different sizes of datasets. The evaluation shows a polynomial (nearly linear) increase in the execution time of the queries in respect with the dataset size. Based on the lesson learned, we believe the quality of discovered paths is highly related to the regular expressions generated to find patterns in the log, i.e., generating regular expressions by domain experts will guarantee the quality of discovered patterns. Moreover, the performance of partitioning the graph using path-conditions (e.g., in PC-Partitions and Path-Partitions) is highly related to the reachability algorithms used for discovering paths among entities.

As illustrated in Figure 5.11, the performance for CC-Partitions is much better in Hadoop execution plans. But comparing Figures 5.11-B and -C, it will be noticed that, although the performance for PC- and Path-Partitions are better in Hadoop execution plans, but this performance is much better in AMZLog dataset comparing to DBLP dataset. In particular, SPARQL graph pattern matching is dominated by several join operations, and is unlikely to be efficiently processed when the type of nodes and edges in the graph increased. Currently, Hadoop supports only partition parallelism in which a single operator executes on different partitions of data across

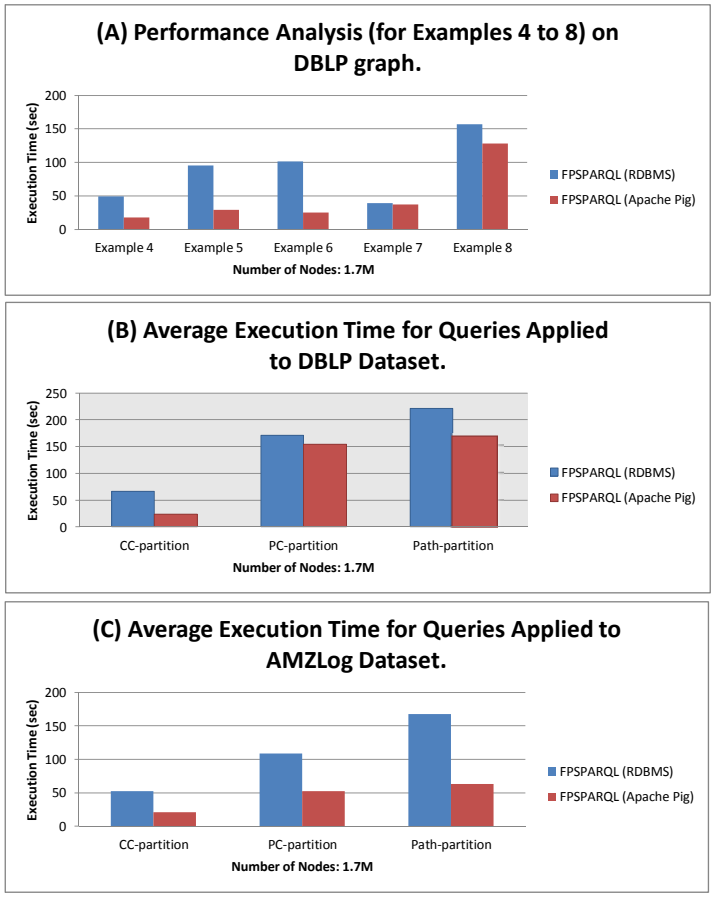


Figure 5.11: The evaluation results, illustrating the performance analysis between RDBMS and Hadoop for: (A) queries in Examples 4 to 8 applied to the DBLP graph; (B) the average execution time, between RDBMS and Hadoop for the CC-, PC-, and Path-Partition queries applied to the DBLP graph; (C) the average execution time, between RDBMS and Hadoop for the CC-, PC-, and Path-Partition queries applied to the AMZlog graph.

the nodes, and the multi-join query plans will be translated into a linear execution plan. This significantly increases the overall communication and I/O overhead involved in RDF graph processing on MapReduce platforms. Consequently, this cost is prohibitive for DBLP graph as it contains four types of nodes and five types of relationships, comparing to AMZlog graph where there are two types of nodes and one type of relationships.

5.7 Related Work

5.7.1 OLAP (On-Line Analytical Processing)

OLAP (On-Line Analytical Processing) [15, 88] is part of the broader category of business intelligence and were conceived to support information analysis using data warehouses in order to extract relevant knowledge of organizations. OLAP applications typically access large (traditional) databases using heavy-weight read-intensive queries. OLAP encompasses *data decision support* (focusing on interactively analyzing multidimensional data from multiple perspectives) and *data mining* (focusing on computational complexity problems). There have been a lot of works, discussed in a recent survey [288] and a book [316], dealing with multidimensional modeling methodologies for OLAP systems. Multidimensional conceptual views allow OLAP analysts to easily understand and analyze data in terms of facts (the subjects of analysis) and dimensions showing the different points of view where a subject can be analyzed from. These line of works, propose OLAP data elements such as partitions, dimensions, and measures and their classification, e.g., classifying OLAP measures into distributive, algebraic and holistic. They discuss that one fact and several dimensions to analyze it give rise to what is known as the data cube.

There are many works, e.g., [167, 343, 336, 147], dealing with the efficient computation of OLAP data cubes. Han et al. [167], studied efficient methods for computing iceberg-cubes[8] with some popularly used measures, such as average, and developed a methodology that adopts a condition for testing and pruning search space. Yuan et al. [343], proposed two algorithms to efficiently compute multiple related skyline[9] results, by sharing as much computation as possible. They proposed to adapt the data cube concept into the skyline computation problem and propose the concept of the Skycube. Xin et al. [336], proposed an aggregation-based approach, named C-Cubing, to compute closed iceberg-cubes more efficiently. Gmez et al. [147] proposed an algebra that operates over data cubes, independently of the underlying

[8] An Iceberg-Cube contains only those cells of the data cube that meet an aggregate condition. It is called an Iceberg-Cube because it contains only some of the cells of the full cube [60].

[9] Skyline [67] has been proposed as an important operator for multi-criteria decision making, data mining and visualization, and user preference queries.

data types and physical data representation. They used the proposed framework to analyze discrete and continuous spatiotemporal data and OLAP cubes together.

Many other works, e.g., [21, 227, 139], deal with clustering and partitioning of large databases, as OLAP queries are typically heavy-weight and ad-hoc thus requiring high processing power. Akal et al. [21], provided a classification of OLAP queries, which is used to decide, whether and how a query should be parallelized. They investigated the number of cluster nodes which should be used to evaluate an OLAP query in parallel. Lima et al. [227], proposed an efficient solution, called adaptive virtual partitioning (AVP), for parallel query processing in a database cluster. The proposed adaptive algorithm, dynamically adjusts the partition sizes during query execution. Moreover, the authors proposed a solution in [139], to combine the physical and virtual partitioning to define table subsets in order to provide flexibility in intra-query parallelism.

Another line of works, e.g., [333, 38], focused on querying multidimensional models, to analyze independent data tuples that mathematically form a set, i.e. conventional spreadsheet data. Witkowski et al. [333], extended SQL with a computational clause to facilitate treating a relation as a multi-dimensional array and specify a set of formulas over it. The formulas replace multiple joins and UNION operations which must be performed for equivalent computation with current ANSI SQL. Balmin et al. [38], proposed a system, Sesame, which leverages both spreadsheets and ad-hoc OLAP tools to assess the effects of hypothetical scenarios. The proposed system, models a hypothetical scenario as a list of hypothetical modifications on the warehouse views and fact data.

5.7.2 On-Line Analytical Processing on Graphs

In recent years, a new stream of work [92, 278, 317, 349, 198, 208, 131, 168, 169] has focused on online analytical processing of information networks[10]. Chen and Qu et al. [92, 278] proposed a conceptual framework for data cubes on graphs and classify

[10]An information-network [168] is a network where each node represents an entity (which may have attributes, labels, and weights) and each link (which may have rich semantic information) represents a relationship between two entities.

their framework into informational (dimensions coming from node attributes) and topological (dimensions coming from node and edge attributes) OLAP. They categorized aggregated graphs based on the difficulty to compute them in an OLAP context, and suggest two properties: localization and attenuation. They proposed techniques in a constraint-pushing framework, to achieve efficient query processing and cube materialization. Tian et al. [317] proposed operations to produce a summary graph (by grouping nodes) and controlling the resolutions of summaries. They proposed operations to group nodes based on user-selected node attributes and relationships. These operations enable users to control the resolutions of summaries and provides the OLAP-style drill-down and roll-up to navigate through summaries with different resolutions.

Zhao et al. [349] introduced a new data warehousing model, Graph-Cube, that supports OLAP queries on graphs. They considered both attribute aggregation and structure summarization of the networks. Besides traditional cuboid queries, they introduced a new class of OLAP queries, crossboid, that is uniquely useful in multi-dimensional networks. We use and extend proposed Graph-Cube to provide multiple views at different granularities. Kämpgen et al. [198] presented a mapping from statistical Linked Data that conforms to the RDF Data Cube vocabulary. They used an extract-transform-load (ETL) pipeline to convert statistical Linked Data into a format suitable for loading into an OLAP system. Moreover, the authors presented an approach in [208] to interact with statistical Linked Data using common OLAP operations. Etcheverry et al. [131] introduced Open Cubes, an RDFS vocabulary for the specification and publication of multidimensional cubes on the Semantic Web. They also presented a general algorithm for creating the SPARQL queries that implement the Roll-up operation. Although these line of works took the first step to put graphs in a rigid multi-dimensional and multi-level framework, much work needs to be done to make OLAP heterogeneous networks a reality [168].

Another line of related work [310, 19, 277, 311, 312, 195] focused on clustering and classification of networks by studying systematically the methods for mining information networks. In particular, graph clustering algorithms, e.g. [310, 19, 277], are of two types: node clustering, in which algorithms determine dense regions of

the graph based on edge behavior, and structural clustering, in which algorithms attempt to cluster different graphs based on overall structural behavior. There are some works in clustering, e.g. [311, 312], developed a ranking-based clustering approach that generates interesting results for both clustering and ranking techniques [168]. Classification techniques [19, 195] classify graphs into a certain number of categories by similarity. These line of works classify networks based on the fact that nodes that are close to similar objects via similar links are likely to be similar. All these works provide some kind of (network) summaries incorporates OLAP-style functionalities.

Other works [278, 123, 138, 276, 31, 209, 53] focused on mining and querying information networks. Qu et al. [278], proposed techniques for query processing and cube materialization on informational networks. Their method compute measures for the newly generated networks and handle user queries with varied constraints. Some of existing approach for querying and modeling graphs [123, 138] focused on defining constraints on nodes and edges simultaneously on the entire object of interest, not in an iterative one-node-at-a-time manner. Therefore, they do not support querying nodes at higher levels of abstraction. SPARQL [276] query language and some of its extensions, e.g., PSPARQL [31] and SPARQLeR [209], have been discussed in Chapter 3. FPSPARQL [53], Folder-, Path-enabled extension of SPARQL, supports folder and path nodes as first class entities that can be defined at several levels of abstractions and queried. In this work, we extend FPSPARQL to provide users with an explorative method to analyze multidimensional graph data from multiple perspectives and granularities.

5.7.3 Analytics over Process Data

Organizations today create vast amounts of transactional data. Converting process execution data into knowledge to support the decision making process is the focus of business analytics [241, 35, 210]. To achieve this, a family of methods and tools can be used for developing new insights and understanding of business performance based on collection, organization, analysis, interpretation, and presen-

tation of process data. In order to collect and organize process data, businesses are using data warehousing, data integration, and Extract-Transform-Load (ETL) solutions [280, 322].

Various methods and techniques have been proposed for analysis and interpretation of process data. The focus of these techniques is on the behavior of completed processes, evaluate currently running process instances, and predicting the behavior of process instances in the future. Some of these techniques [2, 8, 48, 246, 274] are purely syntax oriented, focusing on filtering, translating, interpreting, and modifying event logs given a particular question. Other methods [239, 70, 81, 176] focused on the semantics of process data and tried to propose techniques to understand the hidden relationships among process artifacts. In particular, existing works on business analytics focused more on exploration of new knowledge and investigative analysis using broad range of analysis capabilities, including: trend analysis, what-if analysis, and advanced analysis.

The focus in trend analysis is to explore data and track business developments with capabilities for tracking patterns. Business processes leave trails in a variety of data sources. Process mining [2, 8, 251] techniques and tools (e.g., ProM[11] [7]) are able to extract knowledge from such traces. Aalst et al. [5] focused on applying process mining techniques to Web services, i.e., services leave trails in so-called event logs and recent breakthroughs in process mining research make it possible to discover, analyze, and improve business processes based on such logs. Another line of work [48, 246, 274], in this category, use techniques to monitor the status of running processes and trace the progress of execution. Some other techniques [105, 323, 261] focused on process optimization. These approaches depends on manual analytics and the ability of business analytics to spot the right designs and areas of improvement. Also, some techniques [239, 70, 81, 176] focused on enabling semantic process mining in order to track business patterns. They introduced an intelligent process data warehouse, in which taxonomies are used to add semantics to process execution data and, therefore, support more business-like analysis for the provided reports. Some of these techniques, e.g., [239] and [70], implemented as plugins in the ProM

[11]ProM is the world-leading process mining toolkit. It is an extensible framework that supports a wide variety of process mining techniques in the form of plug-ins.

framework tool. Some other related works [74, 240] focused on information needs for software development analytics. They presented data and analysis needs of professional software engineers, and discussed project management issues to analyze the work of designers, developers and testers.

In what-if analysis, scenarios with capabilities for reorganizing, reshaping and recalculating data is of high interest. In this category, business process data can be used to forecast the future behavior of the organization through techniques such as scenario planning and simulation [89]. Koutsoukis et al. [212] explored the relationships between what-if analysis and multidimensional modeling. They illustrated the natural coupling, which exists between data modeling, symbolic modeling and what-if analysis phases of a decision support systems. They discussed how OLAP-style operations can translate into aggregation and disaggregation of the underlying decision models. Golfarelli et al. [146] described what-if analysis as a data intensive simulation whose goal is to inspect the behavior of a complex system under some given hypotheses. Wynn et al. [335] discussed the possibilities opened by operational process simulation, in terms of being able to perform what-if analysis, by investigating the requirements for a process simulation environment. Papastefanatos et al. [267, 266] focused on what-if analysis for changes that occur in the schema/structure of the data warehouse sources. They modeled queries and relations as a graph that is annotated with policies for the management of evolution events. In particular, their approach detects the parts of the graph that are affected by a given change and highlights the way they are tuned to respond to it.

Advanced analysis techniques, provide techniques to uncover patterns in businesses and discover relationships among important elements in an organization's environment. Linking between entities across repositories has been the focus of a large number of works. For example, the idea of Linked-Data[12] has recently attracted a lot of attention in information systems. Hassanzadeh et al. [171] presented a framework for discovery of semantic links from data based on declarative specification of linkage requirements by a user. Moreover, in [172], the authors proposed light-weight data linking techniques that could link semantically related records

[12]Linked Data is a method of publishing data on the web based on principles that significantly enhance the adaptability and usability of data, either by humans or machines [65].

across internal and external data sources using the power of external knowledge bases available on the Web. Motahari-Nezhad et al. [252] investigated the problem of event correlation for business processes. They identified various ways in which process-related events could be correlated. Rozsnyai et al. [290] proposed techniques to automatically detect correlation identifiers from arbitrary data sources in order to determine relationships between business data. Kurniawan et al. [214] presented a rigorous approach to discover inter-process relationships in a process repository. Moreover, a new stream of work [236, 71, 148, 244, 313] has focused on weaving social technologies to business process management. They aim to consolidate the opportunities for integrating social technologies into the different stages of the business process lifecycle, in order to discover the hidden relationships among process artifacts.

In our approach, we focus on providing users with an explorative method to analyze process data from multiple perspectives and granularities. We use FPSPARQL, a query language for analyzing business processes execution [53], to provide network summaries and to support multidimensional and multi-level views over graphs. We extend FPSPARQL to query and analyze online analytical processing on process graphs in an explorative manner.

5.8 Summary

In this chapter, we have presented concepts, a model, and a language, for online analytical processing on process graphs. Proposed graph data model, GOLAP, enables extending decision support on process data considering both data objects and the relationships among them. We redefined OLAP data elements by considering the relationships among graph entities as first class objects. GOLAP data mode uses folder and path nodes to support multi-dimensional and multi-level views over process graphs.

We extended FPSPARQL to support n-dimensional computations on process graphs. The extension supports partitioning graphs and allows evaluation of OLAP-style operations on graphs, where operations support UPSERT and UPDATE se-

mantics. We described optimizations and execution strategies possible with the proposed extensions. To evaluate the viability and efficiency of FPSPARQL extension, we have conducted experiments over real-world datasets. The evaluation showed that the approach is performing well.

Chapter 6

Conclusions and Future Work

In this chapter, we summarize the contributions of this dissertation and discuss some future research directions to build on this work.

6.1 Concluding Remarks

The continuous demand for the business process improvement and excellence has prompted the need for business process analysis in the enterprise. Recently, business world is getting increasingly dynamic as various technologies such as Internet and email have made dynamic processes more prevalent. Following this, the problem of understanding ad-hoc business process execution has become a priority in medium and large companies. In particular, querying ad-hoc business processes execution is a crucial requirement for many end-users in order to monitor, analyze, understand and improve their business.

In this dissertation, we focused on the problem of explorative querying and understanding of ad-hoc business processes execution. The first contribution of this dissertation is the characterization of this problem in terms of facilitating the analysis of process execution data. Our study shows that only part of interactions related to the process executions are covered by process-aware systems as BPs are realized over a mix of workflows, IT systems, Web services and direct collaborations of people. In order to fulfill the requirements, we have proposed a framework for orga-

nizing, indexing, and querying ad-hoc process data. In this framework, we proposed novel abstractions, folder and path, and a language, FPSPARQL, for the explorative querying and understanding of BP execution from various user perspectives. Below, we summarize the most significant contributions of this dissertation:

- A framework for analyzing ad-hoc process data (Chapter 3). We proposed a graph data model that supports typed and untyped events, and introduced *folder* and *path* nodes as first class abstractions. A folder node contains a collection of related events, and a path node represent the results of a query that consists of one or more paths in the process graph based on a given correlation condition. We presented a process event query language and graph-based querying processing engine called FPSPARQL [53, 52, 51], which is a Folder-, Path-enabled extension of SPARQL [276]. We used FPSPARQL to query and analyze events, folder and path nodes in order to analyze business process execution data. We provided a front-end tool for the exploration and visualization of the results in order to enable users to examine the event relationships and the potential for discovering process instances and process models.

- A framework for analyzing cross-cutting aspects in ad-hoc processes (Chapter 4). We proposed 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. Moreover, we introduced two concepts of *timed-folders* to represent evolution of artifacts over time, and *activity-paths* to represent the process which led to artifacts. We extended FPSPARQL query language for explorative querying and understanding of cross-cutting aspects in ad-hoc processes. A front-end tool has been provided for assisting users to create historical queries in an easy way.

- A framework for applying analytics to ad-hoc process data (Chapter 5). We proposed a graph data model, *GOLAP*, for online analytical processing on process graphs. This data model enabled extending decision support on multidimensional networks considering both data objects and the relationships among them. We used the notions of folder and path nodes to support multidimensional and multi-level views over large process graphs. We redefined OLAP data elements (e.g., dimensions, measures, and cubes) by considering the relationships among graph entities as first class objects. Moreover, we extended FPSPARQL to support n-dimensional computations on process graphs. The proposed extension supports partitioning graphs (using folder and path nodes) and allows evaluation of OLAP-style operations on graphs independently for each partition, providing a natural parallelization of execution. We proposed two types of OLAP operations: *assignments*, to apply operations on entity attributes, and *functions*, to apply operations on network structures among entities. We provided a front-end tool for assisting users to create GOLAP queries in an easy way.

The lessons learned from the experiments and case studies include: (i) the quality of discovered paths is highly related to the regular expressions generated to find patterns in the log, i.e., generating regular expressions by domain experts will guarantee the quality of discovered patterns; (ii) various graph reachability algorithms can be used in process graph analysis (see Appendix A). In general, there are two types of graph reachability algorithms [19]: algorithms traversing from starting vertex to ending vertex using breadth-first or depth-first search over the graph, and algorithms checking whether the connection between two nodes exists in the edge transitive closure of the graph. Considering G as a directed graph that has n nodes and m edges, the first approach incurs high cost as $O(n+m)$ time which requires too much time in querying. The second approach results in high storage consumption in $O(n^2)$ which requires too much space; (iii) the coupling of graph-based querying and native graph-based databases produces interesting possibilities from the point of view of expressiveness and implementation techniques; (iv) adopting techniques such as substructure search and feature based graph indexing methods are inevitable

for fast graph retrieval; and (v) parallel processing, e.g., using MapReduce framework, can be used to handle large graph analysis, however, many graph algorithms are very difficult to be parallelized.

6.2 Future Directions

In this dissertation, we have investigated the problem of explorative querying and understanding of ad-hoc business processes execution.. We believe that this is an important research area, which will attract a lot of attention in the research community. In the following, we summarize significant research directions in this area.

Organizing and Analyzing Large Process Graphs. The scale of business processes execution data poses challenges to their efficient management and processing. The proposed framework may need extension before being applied to generic process execution data from heterogeneous data sources such as emails, Word documents, text documents, etc. This consists of identifying a methodology, automated approach and tools for pre-processing enterprise data. For efficient processing of process graphs, the heterogeneous business data can be organized using MapReduce, a framework which is introduced by Google. In general, graph algorithms (e.g., Page Rank, Pattern Matching, Clustering) can be written as a series of chained MapReduce invocations that requires passing the entire state of the graph from one stage to the next. However, this approach is ill-suited for graph processing and can lead to substantial suboptimal performance due to the much more communication and associated serialization overhead in addition to the need of coordinating the steps of a chained MapReduce. A possible research direction is to use recent computational model, e.g., Pregel [234] (recently introduced by Google), for efficient, scalable and fault-tolerant implementation and analysis of graphs on the cloud.

Social Media and Web 2.0 technologies. In modern enterprises, collaboration and communication among business users fall outside of the BPM suite container. For example, email communication about a process, instant messaging to get a response to a process related question, allowing business users to generate

processes, and allowing front-line workers to update process knowledge (using new technologies such as process Wikis) emphasis the social media's impact on business process management. Possible research directions discussed in [313] to extend: (i) process discovery and design to include interactive real-time involvement of business stakeholders, e.g., users, customers, and partners; (ii) process development methodology and tools to support collaboration between business and IT roles; and (iii) process design to provide real-time guidance for completing a particular activity based on real-time business analytics and social network analysis using crowdsourcing [114] techniques.

Visualization and Storytelling. Current state-of-the-art in supporting users with respect to query formulation have focused on graphical or visual techniques, i.e., to depict the domain of interest and express related queries. A possible research direction is to design a visual query interface to support users in expressing their queries over the conceptual representation of the process graph in an easy way. Moreover, using interactive graph exploration and visualization techniques (e.g., storytelling systems [295]) can help users to quickly identify the interesting parts of process graphs or understand the result of business analytics.

Graph Database as a Service. Many users increasingly require iterative and ad-hoc analysis over graph structures but cannot individually invest in the computational and intellectual infrastructure required for state-of-the-art solutions. In this context, modern Web applications necessitate a flexible and easy-to-use platform to expose their functionality and make it available through standard Web technologies. Moreover, using the underlying concept of Software as a Service (SaaS) enables an economic advantage, where application and system environments only need to be provided once but can be used by thousands of users.

In this context, a possible future work is to extend FPSPARQL to propose a graph query processing engine as a service. We plan to expose the functionality of FPSPARQL query engine in the form of set of REST Web services, i.e., the REST API uses HTTP and JSON, so that it can be used from many languages and platforms. In particular we will provide three categories of services to: create (e.g., to start the server and load the graph), manipulate (e.g., to create

a node/relationship or adding properties to nodes/relationships), and query (e.g., primitive graph queries, constructing folder/path nodes, applying further queries to constructed folder/path nodes, and applying on-line analytical processing) graphs.

Bibliography

[1] W.M.P.V.D. Aalst. Process-aware information systems: Design, enactment, and analysis. In *Wiley Encyclopedia of Computer Science and Engineering*. 2008.

[2] W.M.P.V.D. Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.

[3] W.M.P.V.D. Aalst. Process mining. *Commun. ACM*, 55(8):76–83, 2012.

[4] W.M.P.V.D. Aalst. Process mining: Overview and opportunities. *ACM Trans. Management Inf. Syst.*, 3(2):7, 2012.

[5] W.M.P.V.D. Aalst. Service mining: Using process mining to discover, check, and improve service behavior. *IEEE Transactions on Services Computing*, 99(PrePrints):1, 2012.

[6] W.M.P.V.D. Aalst, A. Adriansyah, A.K.A. Medeiros, F. Arcieri, T. Baier, T. Blickle, R.P.J.C. Bose, P.V.D. Brand, R. Brandtjen, J.C.A.M. Buijs, A. Burattin, J. Carmona, M. Castellanos, J. Claes, J. Cook, N. Costantini, F. Curbera, E. Damiani, M.D. Leoni, P. Delias, B.F.V. Dongen, M. Dumas, S. Dustdar, D. Fahland, D.R. Ferreira, W. Gaaloul, F.V. Geffen, S. Goel, C.W. Günther, A. Guzzo, P. Harmon, A.H.M. Hofstede, J. Hoogland, J. Espen Ingvaldsen, K. Kato, R. Kuhn, A. Kumar, M. Rosa, F. Maria Maggi, D. Malerba, R.S. Mans, A. Manuel, M. McCreesh, P. Mello, J. Mendling, M. Montali, H.R. Motahari-Nezhad, M.Z. Muehlen, J. Muñoz-Gama, L. Pontieri, J. Ribeiro, A. Rozinat, H.S. Pérez, R.S. Pérez, M. Sepúlveda, J. Sinur, P. Soffer, M. Song, A. Sperduti, G. Stilo, C. Stoel, K.D. Swenson, M. Talamo,

W. Tan, C. Turner, J. Vanthienen, G. Varvaressos, E. Verbeek, M. Verdonk, R. Vigo, J. Wang, B. Weber, M. Weidlich, T. Weijters, L. Wen, M. Westergaard, and M.T. Wynn. Process mining manifesto. In *Business Process Management Workshops (1)*, pages 169–194, 2011.

[7] W.M.P.V.D. Aalst, B.F.V. Dongen, C.W. Günther, A. Rozinat, E. Verbeek, and T. Weijters. ProM: The process mining toolkit. In *BPM*, 2009.

[8] W.M.P.V.D. Aalst, B.F.V. Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow mining: a survey of issues and approaches. *Data Knowl. Eng.*, 47:237–267, November 2003.

[9] W.M.P.V.D. Aalst and K.M.V. Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, 2002.

[10] W.M.P.V.D. Aalst, A.H.M.T. Hofstede, and M. Weske. Business process management: A survey. In *Business Process Management*, pages 1–12, 2003.

[11] W.M.P.V.D. Aalst and A.J.M.M. Weijters. Process mining: A research agenda. *Comput. Ind.*, 53(3):231–244, 2004.

[12] W.M.P.V.D. Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.*, 16(9):1128–1142, 2004.

[13] W.M.P.V.D. Aalst, M. Weske, and D. Grünbauer. Case handling: a new paradigm for business process support. *Data Knowl. Eng.*, 53(2):129–162, 2005.

[14] D.J. Abadi, A. Marcus, S. Madden, and K. Hollenbach. SW-Store: a vertically partitioned DBMS for semantic web data management. *VLDB J.*, 18(2):385–406, 2009.

[15] A. Abelló and O. Romero. On-line analytical processing. In *Encyclopedia of Database Systems*, pages 1949–1954. Springer US, 2009.

[16] M. Adams, A.H.M.T. Hofstede, D. Edmond, and W.M.P.V.D. Aalst. Facilitating flexibility and dynamic exception handling in workflows through worklets. In *CAiSE*, 2005.

[17] A. Adriansyah, B.F.V. Dongen, and W.M.P.V.D. Aalst. Conformance checking using cost-based fitness analysis. In *EDOC*, pages 55–64, 2011.

[18] C.C. Aggarwal and H. Wang. Graph data management and mining: A survey of algorithms and applications. In *Managing and Mining Graph Data*, pages 13–68. 2010.

[19] C.C. Aggarwal and H. Wang. *Managing and Mining Graph Data*. Springer Publishing Company, Incorporated, 2010.

[20] R. Agrawal, D. Gunopulos, and F. Leymann. Mining process models from workflow logs. In *Proceedings of the 6th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '98, pages 469–483, London, UK, UK, 1998. Springer-Verlag.

[21] F. Akal, K. Bhm, and H.J. Schek. OLAP query evaluation in a database cluster: A performance study on intra-query parallelism. In *ADBIS*, pages 218–231, 2002.

[22] S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, and K. Tolle. The ICS-FORTH RDFSuite: Managing voluminous RDF description bases. In *SemWeb*, 2001.

[23] F. Alkhateeb, J.F. Baget, and J. Euzenat. Extending SPARQL with regular expression patterns (for querying RDF). *J. Web Sem.*, 7(2):57–73, 2009.

[24] M. Allahbakhsh, A. Ignjatovic, B. Benatallah, S.M.R. Beheshti, E. Bertino, and N. Foo. Reputation management in crowdsourcing systems. In *CollaborateCom*, pages 664–671, 2012.

[25] M. Allahbakhsh, A. Ignjatovic, B. Benatallah, S.M.R. Beheshti, E. Bertino, and N. Foo. Collusion detection in online rating systems. In *Proceedings of The 15th Asia-Pacific Web Conference*, APWeb 2013, pages 196–207, 2013.

[26] M. Allahbakhsh, A. Ignjatovic, B. Benatallah, S.M.R. Beheshti, N. Foo, and E. Bertino. An analytic approach to people evaluation in crowdsourcing systems. *arXiv preprint arXiv:1211.3200*, 2012.

[27] M. Allahbakhsh, A. Ignjatovic, B. Benatallah, S.M.R. Beheshti, N. Foo, and E. Bertino. Detecting, representing and querying collusion in online rating systems. *arXiv preprint arXiv:1211.0963*, 2012.

[28] G. Alonso, F. Casati, H.A. Kuno, and V. Machiraju. *Web Services - Concepts, Architectures and Applications*. Data-Centric Systems and Applications. Springer, 2004.

[29] R. Angles and C. Gutierrez. Survey of graph database models. *ACM Comput. Surv.*, 40:1:1–1:39, February 2008.

[30] D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic. EP-SPARQL: a unified language for event processing and stream reasoning. In *WWW*, 2011.

[31] K. Anyanwu, A. Maduko, and A. Sheth. SPARQ2L: towards support for subgraph extraction queries in RDF databases. WWW'07, pages 797–806, New York, NY, USA, 2007. ACM.

[32] L. Ardissono, R. Furnari, A. Goy, G. Petrone, and M. Segnan. Fault tolerant web service orchestration by means of diagnosis. In *EWSA*, pages 2–16, 2006.

[33] A. Awad. BPMN-Q: A language to query business processes. In *EMISA*, 2007.

[34] A. Awad, A. Polyvyanyy, and M. Weske. Semantic querying of business process models. In *EDOC*, pages 85–94, 2008.

[35] B. Azvine, D. Nauck, and C. Ho. Intelligent business analytics - a tool to build decision-support systems for ebusinesses. *BT Technology Journal*, 21(4):65–71, October 2003.

[36] M. Báez, A. Mussi, F. Casati, A. Birukou, and M. Marchese. Liquid journals: scientific journals in the Web 2.0 era. In *JCDL*, pages 395–396, 2010.

[37] P. Bailey, N. Craswell, I. Soboroff, and A.P.D. Vries. The CSIRO enterprise search test collection. *SIGIR Forum*, 41(2):42–45, 2007.

[38] A. Balmin, T. Papadimitriou, and Y. Papakonstantinou. Hypothetical queries in an OLAP environment. In *VLDB*, pages 220–231, 2000.

[39] R. Barber, P. Bendel, M. Czech, O. Draese, F. Ho, N. Hrle, S. Idreos, M. Kim, O. Koeth, J. Lee, T.T. Li, G.M. Lohman, K. Morfonios, R. Müller, K. Murthy, I. Pandis, L. Qiao, V. Raman, R. Sidle, K. Stolze, and S. Szabo. Business analytics in (a) blink. *IEEE Data Eng. Bull.*, 35(1):9–14, 2012.

[40] D.F. Barbieri, D. Braga, S. Ceri, E.D. Valle, and M. Grossniklaus. C-SPARQL: SPARQL for continuous querying. In *WWW*, pages 1061–1062, 2009.

[41] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti. Run-Time monitoring of the execution of plans for Web service composition. In *ICAPS*, pages 346–349, 2006.

[42] L. Baresi, C. Ghezzi, and S. Guinea. Smart monitors for composed services. In *ICSOC*, pages 193–202, 2004.

[43] L. Baresi and S. Guinea. Towards dynamic monitoring of WS-BPEL processes. In *ICSOC*, pages 269–282, 2005.

[44] L. Baresi, S. Guinea, R. Kazhamiakin, and M. Pistore. An integrated approach for the run-time monitoring of BPEL orchestrations. In *ServiceWave*, pages 1–12, 2008.

[45] A. Barker and J.I.V. Hemert. Scientific workflow: A survey and research directions. In *PPAM*, pages 746–753, 2007.

[46] A.P. Barros, G. Decker, M. Dumas, and F. Weber. Correlation patterns in service-oriented architectures. volume 4422, pages 245–259, 2007.

[47] C. Beeri, A. Eyal, S. Kamenkovich, and T. Milo. Querying business processes with BP-QL. *Inf. Syst.*, 33(6), 2008.

[48] C. Beeri, A. Eyal, T. Milo, and A. Pilberg. Monitoring business processes with queries. In *VLDB*, 2007.

[49] C. Beeri, A. Eyal, T. Milo, and A. Pilberg. BP-Mon: query-based monitoring of BPEL business processes. *SIGMOD Record*, 37(1):21–24, 2008.

[50] A. Begel, Y. Phang Khoo, and T. Zimmermann. Codebook: discovering and exploiting relationships in software repositories. ICSE'10, pages 125–134, 2010.

[51] S.M.R. Beheshti, B. Benatallah, and H.R. Motahari-Nezhad. Enabling the analysis of cross-cutting aspects in ad-hoc processes. In *25th International Conference on Advanced Information Systems Engineering (CAiSE'13), Valencia, Spain*, 2013.

[52] S.M.R. Beheshti, B. Benatallah, H.R. Motahari-Nezhad, and M. Allahbakhsh. A framework and a language for on-line analytical processing on graphs. In *Web Information System Engineering (WISE), 13th International Conference, Paphos, Cyprus*, 2012.

[53] S.M.R. Beheshti, B. Benatallah, H.R. Motahari-Nezhad, and S. Sakr. A query language for analyzing business processes execution. In *Business Process Management (BPM), 9th International Conference, Clermont-Ferrand, France*, pages 281–297, 2011.

[54] S.M.R. Beheshti, H.R. Motahari-Nezhad, and B. Benatallah. Temporal provenance model (tpm): Model and query language. *arXiv preprint arXiv:1211.5009*, 2012.

[55] S.M.R. Beheshti, S. Sakr, B. Benatallah, and H.R. Motahari-Nezhad. Extending SPARQL to support entity grouping and path queries. *arXiv preprint arXiv:1211.5817*, 2012.

[56] V. Bellotti, N. Ducheneaut, M. Howard, and I. Smith. Taking email to task: the design and evaluation of a task management centered email tool. In *CHI*, pages 345–352, 2003.

[57] B. Benatallah, F. Casati, D. Grigori, H.R. Motahari Nezhad, and F. Toumani. Developing adapters for web services integration. In *CAiSE*, pages 415–429, 2005.

[58] A. Bernstein. How can cooperative work tools support dynamic group process? bridging the specificity frontier. In *CSCW*, pages 279–288, 2000.

[59] A. Bernstein and M. Klein. Towards high-precision service retrieval. In *International Semantic Web Conference*, pages 84–101, 2002.

[60] K.S. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg CUBEs. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA*, pages 359–370. ACM Press, 1999.

[61] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*, pages 431–440, 2002.

[62] K. Bhattacharya, C. Evren Gerede, R. Hull, R. Liu, and J. Su. Towards formal analysis of artifact-centric business process models. In *BPM*, pages 288–304, 2007.

[63] K. Bhattacharya, R. Hull, and J. Su. A data-centric design methodology for business processes. In *Handbook of Research on Business Process Modeling, chapter 23*, pages 503–531, 2009.

[64] D. Bianculli and C. Ghezzi. Monitoring conversational web services. In *IW-SOSWE*, pages 15–21, 2007.

[65] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[66] C. Bizer, P.N. Mendes, and A. Jentzsch. Topology of the web of data semantic search over the Web. In *Semantic Search over the Web*, Data-Centric Systems and Applications, chapter 1, pages 3–29. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[67] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[68] R. Bose and W.M.P.V.D. Aalst. Context aware trace clustering: Towards improving process mining results. In *SDM*, 2009.

[69] R.P. Jagadeesh Chandra Bose and W.M.P.V.D. Aalst. Analysis of Patient Treatment Procedures: The BPI Challenge Case Study. Technical Report BPM-11-18, BPMCenter.org, 2011.

[70] R.P. Jagadeesh Chandra Bose, H.M.W. Verbeek, and W.M.P.V.D. Aalst. Discovering hierarchical process models using ProM. In *CAiSE Forum*, pages 33–40, 2011.

[71] M. Brambilla, P. Fraternali, and C. Vaca. BPMN and design patterns for engineering social BPM solutions. In *Business Process Management Workshops*, volume 99 of *Lecture Notes in Business Information Processing*, pages 219–230. Springer Berlin Heidelberg, 2012.

[72] A.Z. Broder and A.C. Ciccolo. Towards the next generation of enterprise search technology. *IBM Systems Journal*, 43(3):451–454, 2004.

[73] J. Broekstra, A. Kampman, and F.V. Harmelen. Sesame: An architecture for storin gand querying RDF data and schema information. In *Spinning the Semantic Web*, pages 197–222, 2003.

[74] R.P.L. Buse and T. Zimmermann. Information needs for software development analytics. In *ICSE*, pages 987–996, 2012.

[75] C. Bussler. *B2B Integration: Concepts and Architecture*. Springer, 2003.

[76] T. Cadenhead, V. Khadilkar, M. Kantarcioglu, and B.M. Thuraisingham. A language for provenance access control. In *CODASPY*, pages 133–144, 2011.

[77] Y. Cai, X.L. Dong, A.Y. Halevy, J.M. Liu, and J. Madhavan. Personal information management with SEMEX. In *SIGMOD Conference*, pages 921–923, 2005.

[78] T. Calders, C.W. Günther, M. Pechenizkiy, and A. Rozinat. Using minimum description length for process mining. In *SAC*, pages 1451–1455, 2009.

[79] M.J. Carey. SOA what? *IEEE Computer*, 41(3):92–94, 2008.

[80] M.J. Carey, N. Onose, and M. Petropoulos. Data services. *Commun. ACM*, 55(6):86–97, 2012.

[81] F. Casati and M.C. Shan. Semantic analysis of business process executions. In *EDBT*, pages 287–296, 2002.

[82] F. Casati and A. Wombacher. Introduction. *Service Oriented Computing and Applications*, 1(3):155, 2007.

[83] M. Castellanos, F. Casati, U. Dayal, and M.C. Shan. A comprehensive and automated approach to intelligent business processes execution analysis. *Distributed and Parallel Databases*, 16(3):239–273, 2004.

[84] R. Cattell. Scalable SQL and NoSQL data stores. *SIGMOD Record*, 39(4):12–27, 2010.

[85] R. Chaiken, B. Jenkins, P. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. SCOPE: easy and efficient parallel processing of massive data sets. *PVLDB*, 1(2):1265–1276, 2008.

[86] C.Y. Chan, M.N. Garofalakis, and R. Rastogi. RE-tree: an efficient index structure for regular expressions. *VLDB J.*, 12(2):102–119, 2003.

[87] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R.E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2), 2008.

[88] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Record*, 26(1):65–74, 1997.

[89] S. Chaudhuri, U. Dayal, and V. Narasayya. An overview of business intelligence technology. *Commun. ACM*, 54(8):88–98, August 2011.

[90] A. Chebotko, S. Lu, X. Fei, and F. Fotouhi. RDFProv: A relational RDF store for querying and managing scientific workflow provenance. *Data Knowl. Eng.*, 69(8):836–865, 2010.

[91] A. Chebotko, S. Lu, and F. Fotouhi. Semantics preserving SPARQL-to-SQL translation. *Data Knowl. Eng.*, 68(10):973–1000, 2009.

[92] C. Chen, X. Yan, F. Zhu, J. Han, and P.S. Yu. Graph OLAP: Towards online analytical processing on graphs. In *ICDM*, pages 103–112, 2008.

[93] J. Cheney, L. Chiticariu, and W.C. Tan. Provenance in databases: Why, how, and where. *Found. Trends databases*, 1:379–474, April 2009.

[94] J. Cheng and J.X. Yu. On-line exact shortest distance query processing. In *EDBT*, pages 481–492, 2009.

[95] E.I. Chong, S. Das, G. Eadon, and J. Srinivasan. An efficient SQL-based RDF querying scheme. In *VLDB*, pages 1216–1227, 2005.

[96] W. M. Coalition. *Terminology and Glossary*. Document Number WFMCTC-1011, www.wfmc.org/standards/docs/TC-1011 term glossary v3.pdf, Feb. 1999.

[97] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and distance queries via 2-Hop labels. *SIAM J. Comput.*, 32(5):1338–1355, 2003.

[98] J. Cohen. Graph twiddling in a MapReduce world. *Computing in Science and Engineering*, 11(4):29–41, 2009.

[99] D. Cohn and R. Hull. Business artifacts: A data-centric approach to modeling business operations and processes. *IEEE Data Eng. Bull.*, 32(3):3–9, 2009.

[100] J.E. Cook and A.L. Wolf. Discovering models of software processes from event-based data. *ACM Trans. Softw. Eng. Methodol.*, 7(3):215–249, July 1998.

[101] B.F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.A. Jacobsen, N. Puz, D Weaver, and R. Yerneni. PNUTS: Yahoo!'s hosted data serving platform. *PVLDB*, 1(2):1277–1288, 2008.

[102] J. Carlos Corrales, D. Grigori, and M. Bouzeghoub. BPEL processes matchmaking for service discovery. In *OTM Conferences (1)*, pages 237–254, 2006.

[103] R. Cyganiak. A relational algebra for SPARQL. Technical report, hpl-2005-170, HP-Labs, 2005.

[104] A. Datta. Automating the discovery of AS-IS business process models: Probabilistic and algorithmic approaches. *Information Systems Research*, 9(3):275–301, 1998.

[105] U. Dayal. Business process optimization. In *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, volume 3290 of *Lecture Notes in Computer Science*, pages 2–2. Springer Berlin / Heidelberg, 2004.

[106] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[107] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *SOSP*, pages 205–220, 2007.

[108] R. Delbru, S. Campinas, and G. Tummarello. Searching web data: An entity retrieval and high-performance indexing model. *J. Web Sem.*, 10:33–58, 2012.

[109] R. Delbru, N. Toupikov, M. Catasta, and G. Tummarello. A node indexing scheme for Web entity retrieval. In *ESWC (2)*, pages 240–256, 2010.

[110] G. Demartini. Leveraging semantic technologies for enterprise search. In *PIKM*, pages 25–32, 2007.

[111] A.J. Demers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, and W.M. White. Cayuga: A general purpose event monitoring system. In *CIDR*, pages 412–422, 2007.

[112] R.M. Dijkman, M. Dumas, and L. García-Bañuelos. Graph matching algorithms for business process model similarity search. In *BPM*, pages 48–63, 2009.

[113] L. Ding, V. Peristeras, and M. Hausenblas. Linked open government data [guest editors' introduction]. *IEEE Intelligent Systems*, 27(3):11–15, 2012.

[114] A. Doan, R. Ramakrishnan, and A.Y. Halevy. Crowdsourcing systems on the World-Wide Web. *Commun. ACM*, 54(4):86–96, 2011.

[115] X. Dong and A.Y. Halevy. A platform for personal information management and integration. In *CIDR*, pages 119–130, 2005.

[116] X. Dong and A.Y. Halevy. Indexing dataspaces. In *SIGMOD Conference*, pages 43–54, 2007.

[117] B.F.V. Dongen, R.M. Dijkman, and J. Mendling. Measuring similarity between business process models. In *CAiSE*, pages 450–464, 2008.

[118] B.F.V. Dongen, J. Mendling, and W.M.P.V.D. Aalst. Structural patterns for soundness of business process models. In *EDOC*, pages 116–128, 2006.

[119] C. Dorn, T. Burkhart, D. Werth, and S. Dustdar. Self-adjusting recommendations for people-driven ad-hoc processes. In *BPM*, pages 327–342, 2010.

[120] C. Dorn and S. Dustdar. Supporting dynamic, people-driven processes through self-learning of message flows. In *CAiSE*, pages 657–671, 2011.

[121] C. Dorn, C.A. Marín, N. Mehandjiev, and S. Dustdar. Self-learning predictor aggregation for the evolution of people-driven ad-hoc processes. In *BPM*, pages 215–230, 2011.

[122] C. Dorn, C.A. Marn, N. Mehandjiev, and S. Dustdar. Self-learning predictor aggregation for the evolution of people-driven ad-hoc processes. In *BPM*, pages 215–230, 2011.

[123] A. Dries, S. Nijssen, and L. De Raedt. A query language for analyzing networks. CIKM'09, pages 485–494, NY, USA, 2009. ACM.

[124] S. Dumais, E. Cutrell, J. Cadiz, G. Jancke, R. Sarin, and D.C. Robbins. Stuff i've seen: a system for personal information retrieval and re-use. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and*

development in informaion retrieval, SIGIR '03, pages 72–79, New York, NY, USA, 2003. ACM.

[125] M. Dumas, W.M.P.V.D. Aalst, and A.H.M.T. Hofstede. *Process-Aware Information Systems: Bridging People and Software Through Process Technology.* Wiley, 2005.

[126] S. Dustdar, T. Hoffmann, and W.M.P.V.D. Aalst. Mining of ad-hoc business processes with teamlog. *Data Knowl. Eng.*, 55(2):129–158, 2005.

[127] C.E. Dyreson. Aspect-oriented relational algebra. In *EDBT*, pages 377–388, 2011.

[128] L. Egghe. Theory and practise of the *g*-index. *Scientometrics*, 69(1):131–152, 2006.

[129] M. Ehrig, A. Koschmider, and A. Oberweis. Measuring similarity between semantic business process models. In *APCCM*, pages 71–80, 2007.

[130] R. Eshuis and P. Grefen. Structural matching of BPEL processes. In *ECOWS*, pages 171–180, 2007.

[131] L. Etcheverry and A.A. Vaisman. Enhancing OLAP analysis with Web cubes. In *ESWC*, pages 469–483, 2012.

[132] O. Ezenwoye and S.M. Sadjadi. TRAP/BPEL: A framework for dynamic adaptation of composite services. In *In Proceedings of the International Conference on Web Information Systems and Technologies (WEBIST 2007*, 2007.

[133] A. Fernández, Z. Cong, and A. Baltá. Bridging the gap between service description models in service matchmaking. *Multiagent and Grid Systems*, 8(1):83–103, 2012.

[134] C. Francescomarino and P. Tonella. Crosscutting concern documentation by visual query of business processes. In *BPM Workshops*, 2008.

[135] M.J. Franklin, A.Y. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *SIGMOD Record*, 34(4):27–33, 2005.

[136] J. Freire, D. Koop, E. Santos, and C.T. Silva. Provenance for computational tasks: A survey. *Computing in Science and Engg.*, 10:11–21, May 2008.

[137] J.P. Friedenstab, C. Janiesch, M. Matzner, and O. Müller. Extending BPMN for business activity monitoring. In *HICSS*, pages 4158–4167, 2012.

[138] T. Fritz and G.C. Murphy. Using information fragments to answer the questions developers ask. ICSE'10, pages 175–184, NY, USA, 2010. ACM.

[139] C. Furtado, A.A.B. Lima, E. Pacitti, P. Valduriez, and M. Mattoso. Physical and virtual partitioning in OLAP database clusters. In *SBAC-PAD*, pages 143–150, 2005.

[140] Gartner. *Business Activity Monitoring: Calm Before the Storm*. ID Number: LE-15-9727, http://www.gartner.com/resources/105500/105562/105562.pdf, Apr. 2002.

[141] D. Georgakopoulos, M.F. Hornick, and A.P. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.

[142] C.E. Gerede and J. Su. Specification and verification of artifact behaviors in business process models. In *ICSOC*, pages 181–192, 2007.

[143] D. Gibbon and Z. Liu. Large scale content analysis engine. In *Proceedings of the First ACM workshop on Large-scale multimedia retrieval and mining*, LS-MMRM '09, pages 97–104, New York, NY, USA, 2009. ACM.

[144] R. Giugno and D. Shasha. GraphGrep: A fast and universal method for querying graphs. In *ICPR (2)*, pages 112–115, 2002.

[145] A. Goderis, P. Li, and C.A. Goble. Workflow discovery: the problem, a case study from e-science and a graph-based solution. In *ICWS*, pages 312–319, 2006.

[146] M. Golfarelli, S. Rizzi, and A. Proli. Designing what-if analysis: towards a methodology. In *DOLAP*, pages 51–58, 2006.

[147] L.I. Gómez, S.A. Gómez, and A.A. Vaisman. A generic data model and query language for spatiotemporal OLAP cube analysis. In *EDBT*, pages 300–311, 2012.

[148] R. Gottanka and N. Meyer. ModelAsYouGo: (re-) design of S-BPM process models during execution time. In *S-BPM ONE Scientific Research*, volume 104 of *Lecture Notes in Business Information Processing*, pages 91–105. Springer Berlin Heidelberg, 2012.

[149] F. Grandi. T-SPARQL: a TSQL2-like temporal query language for RDF. In *International Workshop on Querying Graph Structured Data*, pages 21–30, 2010.

[150] J. Gray, D.T. Liu, M.A. Nieto-Santisteban, A.S. Szalay, D.J. DeWitt, and G. Heber. Scientific data management in the coming decade. *SIGMOD Record*, 34(4):34–41, 2005.

[151] O. Grebner, E. Ong, U.V. Riss, M. Brunzel, A. Bernardi, and T. Roth-Berghofer. Task management model - nepomuk deliverable D3.1. Technical report, 2006.

[152] G. Greco, A. Guzzo, L. Pontieri, and D. Saccà. Discovering expressive process models by clustering log traces. *IEEE Trans. Knowl. Data Eng.*, 18(8):1010–1027, 2006.

[153] D. Grigori, J. Carlos Corrales, and M. Bouzeghoub. Behavioral matchmaking for service retrieval: Application to conversation protocols. *Inf. Syst.*, 33(7-8):681–698, 2008.

[154] R.L. Grossman and Y. Gu. Data mining using high performance data clouds: experimental studies using sector and sphere. In *KDD*, pages 920–927, 2008.

[155] Object Management Group. *Business Process Modeling Notation Specification*. OMG Final Adopted Specification, February 2006.

[156] D. Gruen, S.L. Rohall, S.O. Minassian, B. Kerr, P. Moody, B. Stachel, M. Wattenberg, and E. Wilcox. Lessons from the reMail prototypes. In *CSCW*, pages 152–161, 2004.

[157] M. Grund, P. Cudré-Mauroux, and S. Madden. A demonstration of HYRISE - a main memory hybrid storage engine. *PVLDB*, 4(12):1434–1437, 2011.

[158] A. Gubichev, S.J. Bedathur, and S. Seufert. Fast and accurate estimation of shortest paths in large graphs. CIKM'10, pages 499–508, 2010.

[159] P.K. Gunda, L. Ravindranath, C.A. Thekkath, Y. Yu, and L. Zhuang. Nectar: automatic management of data and computation in datacenters. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

[160] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked keyword search over XML documents. In *SIGMOD Conference*, pages 16–27, 2003.

[161] R. Hartmut Güting. GraphDB: Modeling and querying graphs in databases. In *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 297–308. Morgan Kaufmann, 1994.

[162] A. Guttman. R-Trees: A dynamic index structure for spatial searching. In B. Yormark, editor, *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, pages 47–57. ACM Press, 1984.

[163] M. Gyssens, J. Paredaens, J.V.D. Bussche, and D.V. Gucht. A graph-oriented object database model. *IEEE Trans. Knowl. Data Eng.*, 6(4):572–586, 1994.

[164] H. Hacigümüs, S. Mehrotra, and B.R. Iyer. Providing database as a service. In *ICDE*, pages 29–38, 2002.

[165] A.Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.

[166] A.Y. Halevy, M.J. Franklin, and D. Maier. Principles of dataspace systems. In *PODS*, pages 1–9, 2006.

[167] J. Han, J. Pei, G. Dong, and K. Wang. Efficient computation of iceberg cubes with complex measures. In *SIGMOD Conference*, pages 1–12, 2001.

[168] J. Han, Y. Sun, X. Yan, and P.S. Yu. Mining knowledge from data: An information network analysis approach. In *ICDE*, 2012.

[169] J. Han, X. Yan, and P.S. Yu. Scalable OLAP and mining of information networks. In *EDBT*, 2009.

[170] P. Harmon. *Complex, Dynamic Processes.* Bptrends, Volume 4, Number 1, January 2011.

[171] O. Hassanzadeh, S. Duan, A. Fokoue, A. Kementsietsidis, K. Srinivas, and M.J. Ward. Helix: online enterprise data analytics. In *WWW (Companion Volume)*, pages 225–228, 2011.

[172] O. Hassanzadeh, A. Kementsietsidis, L. Lim, R.J. Miller, and M. Wang. A framework for semantic link discovery over relational data. In *CIKM*, pages 1027–1036, 2009.

[173] D. Hawking. Challenges in enterprise search. In *ADC*, pages 15–24, 2004.

[174] H. He. *Querying and Mining Graph Databases.* PhD thesis, UCSB, 2007.

[175] T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space.* Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers, 2011.

[176] M. Hepp, F. Leymann, J. Domingue, A. Wahler, and D. Fensel. Semantic business process management: A vision towards using semantic web services for business process management. In *ICEBE*, pages 535–540, 2005.

[177] J. Herbst. A machine learning approach to workflow management. In *ECML*, pages 183–194, 2000.

[178] M.A. Hernández and S.J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Min. Knowl. Discov.*, 2(1):9–37, 1998.

[179] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F.B. Cetin, and S. Babu. Starfish: A self-tuning system for big data analytics. In *CIDR*, pages 261–272, 2011.

[180] J.E. Hirsch. An index to quantify an individual's scientific research output that takes into account the effect of multiple coauthorship. *Scientometrics*, 85(3):741–754, 2010.

[181] D.A. Holl, U. Braun, D. Maclean, and K. Muniswamy-reddy. Choosing a data model and query language for provenance. IPAW'08, 2008.

[182] D.A. Holland, U. Braun, D. Maclean, K.K. Muniswamy-Reddy, , and M. Seltzer. Choosing a data model and query language for provenance. In *IPAW*, 2008.

[183] D. Hollingsworthm. *The Workflow Reference Model*. Workflow Management Coalition, http://www.wfmc.org/standards/docs/tc003v11.pdf, Jan. 1995.

[184] P. Holme and J. Saramki:. Temporal networks. *CoRR*, abs/1108.1780, 2011.

[185] H. Holz, O. Rostanin, A. Dengel, T. Suzuki, K. Maeda, and K. Kanasaki. Task-based process know-how reuse and proactive information delivery in TaskNavigator. In *CIKM*, pages 522–531, 2006.

[186] R. Hull. Artifact-centric business process models: Brief survey of research results and challenges. In *OTM Conferences (2)*, pages 1152–1163, 2008.

[187] M.F. Husain, P. Doshi, L. Khan, and B.M. Thuraisingham. Storage and retrieval of large RDF graph using Hadoop and MapReduce. In *CloudCom*, pages 680–686, 2009.

[188] M.F. Husain, L. Khan, M. Kantarcioglu, and B.M. Thuraisingham. Data intensive query processing for large RDF graphs using cloud computing tools. In *IEEE CLOUD*, pages 1–10, 2010.

[189] M. Indulska, J. Recker, M. Rosemann, and P.F. Green. Business process modeling: Current issues and future challenges. In *CAiSE*, pages 501–514, 2009.

[190] B. Iordanov. HyperGraphDB: A generalized graph database. WAIM'10, pages 25–36, 2010.

[191] Z.G. Ives, N. Khandelwal, A. Kapur, and M. Cakir. ORCHESTRA: Rapid, collaborative sharing of dynamic data. In *CIDR*, pages 107–118, 2005.

[192] E. Olding D. Plummer B. Rosser J. Hill, B. Lheureux and J. Sinur. *Predicts 2010: Business Process Management Will Expand Beyond Traditional Boundaries*. Gartner Reports, 2009.

[193] C. Janiesch, M. Matzner, and O. Müller. A blueprint for event-driven business activity management. In *BPM*, pages 17–28, 2011.

[194] S.R. Jeffery, M.J. Franklin, and A.Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In *SIGMOD Conference*, pages 847–860, 2008.

[195] M. Ji, Y. Sun, M. Danilevsky, J. Han, and J. Gao. Graph regularized transductive classification on heterogeneous information networks. In *ECML/PKDD (1)*, pages 570–586, 2010.

[196] R. Jin, Y. Xiang, N. Ruan, and D. Fuhry. 3-HOP: a high-compression indexing scheme for reachability query. In *SIGMOD Conference*, pages 813–826, 2009.

[197] H.D. Jorgensen. *Interactive Process Models*. PhD thesis, Norwegian University of Science and Technology, Trondheim, Norway, 2004.

[198] B. Kämpgen and A. Harth. Transforming statistical linked data for use in OLAP systems. In *I-SEMANTICS*, pages 33–40, 2011.

[199] U. Kang, C.E. Tsourakakis, and C. Faloutsos. PEGASUS: mining peta-scale graphs. *Knowl. Inf. Syst.*, 27(2):303–325, 2011.

[200] D.R. Karger, K. Bakshi, D. Huynh, D. Quan, and V. Sinha. Haystack: A general-purpose information management tool for end users based on semistructured data. In *CIDR*, pages 13–26, 2005.

[201] G. Karvounarakis, Z.G. Ives, and V. Tannen. Querying data provenance. In *SIGMOD*. ACM, 2010.

[202] A. Kemper and T. Neumann. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In *ICDE*, pages 195–206, 2011.

[203] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.

[204] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C.V. Lopes, J.M. Loingtier, and J. Irwin. Aspect-oriented programming. In *ECOOP*, pages 220–242, 1997.

[205] C. Kiefer, A. Bernstein, H.J. Lee, M. Klein, and M. Stocker. Semantic process retrieval with iSPARQL. In *ESWC*, pages 609–623, 2007.

[206] H. Kim, P. Ravindra, and K. Anyanwu. From SPARQL to MapReduce: The journey using a nested triplegroup algebra. *PVLDB*, 4(12):1426–1429, 2011.

[207] M. Klusch, B. Fries, and K.P. Sycara. Automated semantic web service discovery with OWLS-MX. In *AAMAS*, pages 915–922, 2006.

[208] B. Kmpgen, S. O'Riain, and A. Harth. Interacting with statistical linked data via OLAP operations. In *ILD-ESWC*, 2012.

[209] K.J. Kochut and M. Janik. SPARQLeR: Extended SPARQL for semantic association discovery. ESWC'07, pages 145–159, Berlin, Heidelberg, 2007. Springer.

[210] R. Kohavi, N.J. Rothleder, and E. Simoudis. Emerging trends in business analytics. *Commun. ACM*, 45(8):45–48, August 2002.

[211] V. Kostakos. Temporal graph. *Physica A: Statistical Mechanics and its Applications*, 388(6):1007–1023, 2009.

[212] N.S. Koutsoukis, G. Mitra, and C. Lucas. Adapting on-line analytical processing for decision modelling: the interaction of information and decision technologies. *Decis. Support Syst.*, 26(1):1–30, July 1999.

[213] J. Kuo. A document-driven agent-based approach for business processes management. *Information and Software Technology*, 46(6):373–382, 2004.

[214] T.A. Kurniawan, A.K. Ghose, L.S. L, and H.K. Dam. On formalizing interprocess relationships. In *Business Process Management Workshops*, volume 100 of *Lecture Notes in Business Information Processing*, pages 75–86. Springer Berlin Heidelberg, 2012.

[215] J.M. Küster, C. Gerth, A. Förster, and G. Engels. Detecting and resolving process model differences in the absence of a change log. In *BPM*, pages 244–260, 2008.

[216] N. Kwasnikowska, L. Moreau, and J. Van Den Bussche. A formal account of the open provenance model. *submitted*, pages 1–49, 2010.

[217] N. Leavitt. Are web services finally ready to deliver? *IEEE Computer*, 37(11):14–18, 2004.

[218] N. Leavitt. Will NoSQL databases live up to their promise? *IEEE Computer*, 43(2):12–14, 2010.

[219] J. Lee, M. Podlaseck, E. Schonberg, and R. Hoch. Visualization and analysis of clickstream data of online stores for understanding Web merchandising. *Data Min. Knowl. Discov.*, 5(1-2):59–84, January 2001.

[220] J. Leskovec, L.A. Adamic, and B.A. Huberman. The dynamics of viral marketing. *TWEB*, 1(1), 2007.

[221] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, KDD '05, pages 177–187, New York, NY, USA, 2005. ACM.

[222] M. Levene and A. Poulovanssilis. An object-oriented data model formalised through hypergraphs. *Data Knowl. Eng.*, 6:205–224, May 1991.

[223] M. Levene and A. Poulovassilis. The hypernode model and its associated query language. JCIT, pages 520–530, CA, USA, 1990. IEEE Computer Society Press.

[224] A.Y. Levy, A. Rajaraman, and J.J. Ordille. Querying heterogeneous information sources using source descriptions. In *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 251–262. Morgan Kaufmann, 1996.

[225] F. Leymann and D. Roller. *Production workflow - concepts and techniques*. Prentice Hall, 2000.

[226] L. Lim, H. Wang, and M. Wang. Semantic queries in databases: problems and challenges. In *CIKM*, pages 1505–1508, 2009.

[227] A.A.B. Lima, M. Mattoso, and P. Valduriez. Adaptive virtual partitioning for OLAP query processing in a database cluster. *JIDM*, 1(1):75–88, 2010.

[228] J. Lin and M. Schatz. Design patterns for efficient graph algorithms in MapReduce. In *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*, MLG '10, pages 78–85, New York, NY, USA, 2010. ACM.

[229] D.R. Liu and M. Shen. Workflow modeling for virtual processes: an order-preserving process-view approach. *Inf. Syst.*, 28(6):505–532, 2003.

[230] D.C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[231] C. Lynch. Big data: How do your data grow? *Nature*, 455(7209):28–29, 2008.

[232] L. Ma, Z. Su, Y. Pan, L. Zhang, and T. Liu. RStar: an RDF storage and query system for enterprise resource management. In *CIKM*, pages 484–491, 2004.

[233] B. Mahleko and A. Wombacher. Indexing business processes based on annotated finite state automata. In *ICWS*, pages 303–311, 2006.

[234] G. Malewicz, M.H. Austern, A.J.C. Bik, J.C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD Conference*, pages 135–146, 2010.

[235] F. Manola and E. Miller. *RDF Primer*. W3C, http://www.w3.org/TR/rdf-primer/, 2004.

[236] P. Mathiesen, J. Watson, W. Bandara, and M. Rosemann. Applying social technology to business process lifecycle management. In *Business Process Management Workshops*, volume 99 of *Lecture Notes in Business Information Processing*, pages 231–241. Springer Berlin Heidelberg, 2012.

[237] G. Mecca, P. Papotti, and S. Raunich. Core schema mappings: Scalable core computations in data exchange. *Inf. Syst.*, 37(7):677–711, 2012.

[238] A.K.A. Medeiros, W.M.P.V.D. Aalst, and A.J.M.M. Weijters. Quantifying process equivalence based on observed behavior. *Data Knowl. Eng.*, 64(1):55–74, 2008.

[239] A.K.A.D. Medeiros, W.M.P.V.D. Aalst, and C. Pedrinaci. Semantic process mining tools: Core building blocks. In *ECIS*, pages 1953–1964, 2008.

[240] T. Menzies and T. Zimmermann. Goldfish bowl panel: Software development analytics. In *ICSE*, pages 1032–1033, 2012.

[241] M. Mhlen and R. Shapiro. Business process analytics. In *Handbook on Business Process Management 2*, International Handbooks on Information Systems, pages 137–157. Springer Berlin Heidelberg, 2010.

[242] P. Missier, N.W. Paton, and K. Belhajjame. Fine-grained and efficient lineage querying of collection-based workflow provenance. In *EDBT*, pages 299–310, 2010.

[243] T. Mitsa. *Temporal Data Mining*. Chapman & Hall/CRC, 1st edition, 2010.

[244] M. Molhanec. Enterprise systems meet social BPM. In *Advanced Information Systems Engineering Workshops*, volume 112 of *Lecture Notes in Business Information Processing*, pages 413–424. Springer Berlin Heidelberg, 2012.

[245] C. Momm, R. Malec, and S. Abeck. Towards a model-driven development of monitored processes. In *In 8 Internationale Tagung Wirtschaftsinformatik (WI2007*, 2007.

[246] M. Momotko and K. Subieta. Process query language: A way to make workflow processes more flexible. In *ADBIS*, 2004.

[247] L. Moreau. Provenance-based reproducibility in the semantic Web. *J. Web Sem.*, 9(2):202–221, 2011.

[248] L. Moreau, J. Freire, J. Futrelle, R.E. Mcgrath, J. Myers, and P. Paulson. The open provenance model: An overview. IPAW'08, pages 323–326, Berlin, Heidelberg, 2008. Springer.

[249] H.R. Motahari-Nezhad and C. Bartolini. Next best step and expert recommendation for collaborative processes in it service management. In *BPM*, pages 50–61, 2011.

[250] H.R. Motahari-Nezhad, B. Benatallah, R. Saint-Paul, F. Casati, and P. Andritsos. Process spaceship: discovering and exploring process views from event logs in data spaces. *PVLDB*, 1(2):1412–1415, 2008.

[251] H.R. Motahari-Nezhad, R. Saint-Paul, B. Benatallah, and F. Casati. Deriving protocol models from imperfect service conversation logs. *IEEE Trans. on Knowl. and Data Eng.*, 20:1683–1698, December 2008.

[252] H.R. Motahari-Nezhad, R. Saint-Paul, F. Casati, and B. Benatallah. Event correlation for process discovery from web service interaction logs. *VLDB J.*, 20(3):417–444, 2011.

[253] M. Muehlen and M. Rosemann. Workflow-based process monitoring and controlling - technical and organizational issues. In *HICSS*, 2000.

[254] M.Z. Muehlen. *Workflow-based Process Controlling. Foundation, Design, and Application of workflow-driven Process Information Systems.* Logos, July 2004.

[255] R. Mukherjee and J. Mao. Enterprise search: Tough stuff. *ACM Queue*, 2(2):36–46, 2004.

[256] J. Muñoz-Gama and J. Carmona. Enhancing precision in process conformance: Stability, confidence and severity. In *CIDM*, pages 184–191, 2011.

[257] T. Murata. Petri Nets: Properties, analysis and applications. In *Proceedings of the IEEE, volume 77, number 4*, pages 541–580, April 1989. NewsletterInfo: 33Published as Proceedings of the IEEE, volume 77, number 4.

[258] S. Nejati, M. Sabetzadeh, M. Chechik, S. M. Easterbrook, and P. Zave. Matching and merging of statecharts specifications. In *ICSE*, pages 54–64, 2007.

[259] T. Neumann and G. Weikum. RDF3X: a riscstyle engine for RDF. In *VLDB*, pages 647–659, 2008.

[260] M. Newman. *Small Worlds: The Dynamics of Networks between Order and Randomness*. Oxford Univ. Press, 2010.

[261] F. Niedermann, S. Radeschtz, and B. Mitschang. Deep business optimization: A platform for automated process optimization. In *ISSS/BPSC*, volume 177 of *LNI*, pages 168–180. GI, 2010.

[262] T. Nykiel, M. Potamias, C. Mishra, G. Kollios, and N. Koudas. MRShare: Sharing across multiple queries in MapReduce. *PVLDB*, 3(1):494–505, 2010.

[263] B.C. Ooi, B. Yu, and G. Li. One table stores all: Enabling painless free-and-easy data publishing and sharing. CIDR'07, pages 142–153, 2007.

[264] Oracle. *Business Activity Monitoring*. http://www.oracle.com/ technology/ products/ integration/ bam/ pdf/ oracle-bam-datasheet.pdf, 2006.

[265] C. Ouyang, M. Thandar Wynn, C. Fidge, A.H.M.T Hofstede, and J.C. Kuhr. Modelling complex resource requirements in business process management systems. In *21st Australasian Conference on Information Systems : Defining and Establishing a High Impact Discipline (ACIS 2010)*, Queensland University of Technology, Brisbane, December 2010. ACIS.

[266] G. Papastefanatos, F. Anagnostou, Y. Vassiliou, and P. Vassiliadis. Hecataeus: A what-if analysis tool for database schema evolution. In *CSMR*, pages 326–328, 2008.

[267] G. Papastefanatos, P. Vassiliadis, A. Simitsis, and Y. Vassiliou. What-if analysis for data warehouse evolution. In *DaWaK*, pages 23–33, 2007.

[268] M.P. Papazoglou and W.J.V.D. Heuvel. Service oriented architectures: approaches, technologies and research issues. *VLDB J.*, 16(3):389–415, 2007.

[269] M.P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *IEEE Computer*, 40(11):38–45, 2007.

[270] C. Pedrinaci, J. Domingue, and A.K.A.D. Medeiros. A core ontology for business process analysis. In *ESWC*, pages 49–64, 2008.

[271] M. Perry, P. Jain, and A.P. Sheth. SPARQL-ST: Extending SPARQL to support spatiotemporal queries. In *Geospatial Semantics and the Semantic Web*, pages 61–86, 2011.

[272] J.M. Petit, F. Toumani, J.F. Boulicaut, and J. Kouloumdjian. Towards the reverse engineering of denormalized relational databases. In *ICDE*, pages 218–227, 1996.

[273] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan. Interpreting the data: Parallel analysis with sawzall. *Scientific Programming*, 13(4):277–298, 2005.

[274] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, and P. Traverso. Planning and monitoring web service composition. In *AIMSA*, 2004.

[275] H. Plattner. A common database approach for OLTP and OLAP using an in-memory column database. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, SIGMOD '09, pages 1–2, New York, NY, USA, 2009. ACM.

[276] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF (working draft). Technical report, W3C, March 2007.

[277] T. Qian, Y. Yang, and S. Wang. Refining graph partitioning for social network clustering. In *WISE*, pages 77–90, 2010.

[278] Q. Qu, F. Zhu, X. Yan, J. Han, P.S. Yu, and H. Li. Efficient topological OLAP on information networks. In *DASFAA*, 2011.

[279] A. Ryman R. Chinnici, J.-J. Moreau and S. Weerawarana. *Web Service Description Language (WSDL) Version 2.0.* W3C Working Draft, http://www.w3.org/TR/wsdl20, June 2007.

[280] E. Rahm and H. Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[281] P. Ravindra, H. Kim, and K. Anyanwu. An intermediate algebra for optimizing RDF graph pattern matching on MapReduce. In *ESWC (2)*, pages 46–61, 2011.

[282] M. Reichert, S. Rechtenbach, A. Hallerbach, and T. Bauer. Extending a business process modeling tool with process configuration facilities: The provop demonstrator. In *BPM (Demos)*, 2009.

[283] M. Reichert, S. Rinderle, and P. Dadam. ADEPT workflow management system. In *Business Process Management*, pages 370–379, 2003.

[284] H.A. Reijers, J.H.M. Rigter, and W.M.P.V.D. Aalst. The case handling case. *Int. J. Cooperative Inf. Syst.*, 12(3):365–391, 2003.

[285] C. Ren, E. Lo, B. Kao, X. Zhu, and R. Cheng. On querying historical evolving graph sequences. *VLDB*, 4(11):727–737, 2011.

[286] L. Resende. Handling heterogeneous data sources in a SOA environment with service data objects (SDO). In *SIGMOD Conference*, pages 895–897, 2007.

[287] U. Riss, A. Rickayzen, H. Maus, and W.M.P.V.D. Aalst. Challenges for business process and task management. Number 2, pages 77–100, 2005.

[288] O. Romero and A. Abelló. A survey of multidimensional modeling methodologies. *IJDWM*, 5(2):1–23, 2009.

[289] A. Rozinat and W.M.P.V.D. Aalst. Conformance checking of processes based on monitoring real behavior. *Inf. Syst.*, 33(1):64–95, 2008.

[290] S. Rozsnyai, A. Slominski, and G.T. Lakshmanan. Automated correlation discovery for semi-structured business processes. In *ICDE Workshops*, pages 261–266, 2011.

[291] S. Rozsnyai, A. Slominski, and G.T. Lakshmanan. Discovering event correlation rules for semi-structured business processes. In *DEBS*, pages 75–86, 2011.

[292] S. Sakr and G. Al-Naymat. Relational processing of RDF queries: a survey. *SIGMOD Rec.*, 38(4):23–28, 2009.

[293] S. Sakr and A. Awad. A framework for querying graph-based business process models. In *WWW*, 2010.

[294] A.D. Sarma, X.L. Dong, and A.Y. Halevy. Data modeling in dataspace support platforms. In *Conceptual Modeling: Foundations and Applications*, pages 122–138, 2009.

[295] A. Satish, R. Jain, and A. Gupta. Tolkien: an event based storytelling system. *Proc. VLDB Endow.*, 2:1630–1633, August 2009.

[296] S. Schaffert, C. Bauer, T. Kurz, F. Dorschel, D. Glachs, and M. Fernandez. The linked media framework: integrating and interlinking enterprise media content and data. In *Proceedings of the 8th International Conference on Semantic Systems*, I-SEMANTICS '12, pages 25–32, New York, NY, USA, 2012. ACM.

[297] A. Schätzle, M. Przyjaciel-Zablocki, and G. Lausen. PigSPARQL: mapping SPARQL to Pig Latin. In *Proceedings of the International Workshop on Semantic Web Information Management*, SWIM '11, pages 4:1–4:8, New York, NY, USA, 2011. ACM.

[298] A.W. Scheer and J. Klueckmann. BPM 3.0. In *BPM*, pages 15–27, 2009.

[299] C.E. Scheidegger, H.T. Vo, D. Koop, J. Freire, and C.T. Silva. Querying and creating visualizations by analogy. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1560–1567, 2007.

[300] H. Schonenberg, B. Weber, B.F.V. Dongen, and W.M.P.V.D. Aalst. Supporting flexible processes through recommendations based on history. In *BPM*, pages 51–66, 2008.

[301] R. Sharma, P. Reynolds, R. Scheepers, P.B. Seddon, and G.G. Shanks. Business analytics and competitive advantage: A review and a research agenda. In *DSS*, volume 212 of *Frontiers in Artificial Intelligence and Applications*, pages 187–198. IOS Press, 2010.

[302] A. Sharp and P. McDermott. *Workflow Modeling: Tools for Process Improvement and Applications Development*. Artech House, 2009.

[303] J. Shen, E. Fitzhenry, and T.G. Dietterich. Discovering frequent work procedures from resource connections. In *IUI*, pages 277–286, 2009.

[304] Z. Shen. Web service discovery based on behavior signatures. In *In Proc. of SCC 2005*, pages 279–286. IEEE Computer Society, 2005.

[305] Y. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34(3):31–36, 2005.

[306] N. Siu, L. Iverson, and A. Tang. Going with the flow: email awareness and task management. In *CSCW*, pages 441–450, 2006.

[307] T. Stoitsev, S. Scheidl, and M. Spahn. A framework for light-weight composition and management of ad-hoc business processes. In *TAMODIA*, pages 213–226, 2007.

[308] M. Stonebraker. SQL databases v. NoSQL databases. *Commun. ACM*, 53(4):10–11, 2010.

[309] S. Subramanian, P. Thiran, N.C. Narendra, G.K. Mostéfaoui, and Z. Maamar. On the enhancement of BPEL engines for self-healing composite Web services. In *SAINT*, pages 33–39, 2008.

[310] Y. Sun, C.C. Aggarwal, and J. Han. Relation strength-aware clustering of heterogeneous information networks with incomplete attributes. *PVLDB*, 5(5):394–405, 2012.

[311] Y. Sun, J. Han, P. Zhao, Z. Yin, H. Cheng, and T. Wu. RankClus: integrating clustering with ranking for heterogeneous information network analysis. In *EDBT*, pages 565–576, 2009.

[312] Y. Sun, Y. Yu, and J. Han. Ranking-based clustering of heterogeneous information networks with star network schema. In *KDD*, pages 797–806, 2009.

[313] K. Swenson, L. Fischer, S. Kemsley, N.L. Palmer, and C. Richardson. *Social BPM: Work, Planning and Collaboration Under the Impact of Social Technology*. Bpm and Workflow Handbook Series. CreateSpace, 2011.

[314] K.D. Swenson, N. Palmer, and B. Silver. *Taming the Unpredictable Real World Adaptive Case Management: Case Studies and Practical Guidance*. Future Strategies Inc, 2011.

[315] J. Tappolet and A. Bernstein. Applied temporal RDF: Efficient temporal querying of RDF data with SPARQL. In *ESWC*, pages 308–322, 2009.

[316] E. Thomsen. *OLAP Solutions: Building Multidimensional Information Systems*. John Wiley & Sons, Inc., New York, NY, USA, 2nd edition, 2002.

[317] Y. Tian, R.A. Hankins, and J.M. Patel. Efficient aggregation for graph summarization. In *SIGMOD Conference*, pages 567–580, 2008.

[318] S. TriBl and U. Leser. Fast and practical indexing and querying of very large graphs. SIGMOD '07, pages 845–856, NY, USA, 2007. ACM.

[319] P. Trkman, K. McCormack, M.P.V. de Oliveira, and M.B. Ladeira. The impact of business analytics on supply chain performance. *Decis. Support Syst.*, 49(3):318–327, June 2010.

[320] H.L. Truong and S. Dustdar. On analyzing and specifying concerns for data as a service. In *APSCC*, pages 87–94, 2009.

[321] R. Vaculín and K.P. Sycara. Towards automatic mediation of OWL-S process models. In *ICWS*, pages 1032–1039, 2007.

[322] P. Vassiliadis. A survey of extract-transform-load technology. In *Integrations of Data Warehousing, Data Mining and Database Technologies*, pages 171–199. 2011.

[323] K. Vergidis, A. Tiwari, and B. Majeed. Business process analysis and optimization: Beyond reengineering. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 38(1):69–82, 2008.

[324] H. Wang and C.C. Aggarwal. A survey of algorithms for keyword search on graph data. In *Managing and Mining Graph Data*, pages 249–273. 2010.

[325] J. Wang and A. Kumar. A framework for document-driven workflow systems. In *BPM*, pages 285–301, 2005.

[326] L. Wang, R. Ranjan, J. Chen, and B. Benatallah. *Cloud Computing: Methodology, Systems, and Applications*. CRC Press, Taylor and Francis Group, In Print (anticipated hardcopy publication date), January 15 2012.

[327] D.J. Watts. *Networks: An Introduction*. Princeton University Press, 2003.

[328] F. Wei. TEDI: Efficient shortest path query answering on graphs. In *Graph Data Management*, pages 214–238. 2011.

[329] L. Wen, J. Wang, W.M.P.V.D. Aalst, B. Huang, and J. Sun. A novel approach for process mining based on event types. *J. Intell. Inf. Syst.*, 32(2), 2009.

[330] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, original edition, June 2009.

[331] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, pages 262–276, 2005.

[332] K. Wilkinson, C. Sayers, H.A. Kuno, and D. Reynolds. Efficient RDF storage and retrieval in Jena2. In *SWDB*, pages 131–150, 2003.

[333] A. Witkowski, S. Bellamkonda, T. Bozkaya, G. Dorman, N. Folkert, A. Gupta, L. Sheng, and S. Subramanian. Spreadsheets in RDBMS for OLAP. In *SIGMOD Conference*, pages 52–63, 2003.

[334] A. Wombacher, P. Fankhauser, B. Mahleko, and E.J. Neuhold. Matchmaking for business processes based on choreographies. In *EEE*, pages 359–368, 2004.

[335] M.T. Wynn, M. Dumas, C.J. Fidge, A.H.M.T. Hofstede, and W.M.P.V.D. Aalst. Business process simulation for operational decision support. In *Business Process Management Workshops*, pages 66–77, 2007.

[336] D. Xin, Z. Shao, J. Han, and H. Liu. C-Cubing: Efficient computation of closed cubes by aggregation-based checking. In *ICDE*, 2006.

[337] X. Yan, P.S. Yu, and J. Han. Graph indexing: A frequent structure-based approach. In *SIGMOD Conference*, pages 335–346, 2004.

[338] X. Yan, P.S. Yu, and J. Han. Substructure similarity search in graph databases. In *SIGMOD Conference*, pages 766–777, 2005.

[339] Z. Yan, R.M. Dijkman, and P.W.P.J. Grefen. Fast business process similarity search with feature-based similarity estimation. In *OTM Conferences (1)*, pages 60–77, 2010.

[340] Z. Yan, R.M. Dijkman, and P.W.P.J. Grefen. Business process model repositories - framework and survey. *Information & Software Technology*, 54(4):380–395, 2012.

[341] J. Yu, B. Benatallah, F. Casati, and F. Daniel. Understanding mashup development. *IEEE Internet Computing*, 12(5):44–52, 2008.

[342] T.L. Yu and D.E. Goldberg. Dependency structure matrix analysis: Offline utility of the dependency structure matrix genetic algorithm. In *GECCO (2)*, pages 355–366, 2004.

[343] Y. Yuan, X. Lin, Q. Liu, W. Wang, J.X. Yu, and Q. Zhang. Efficient computation of the skyline cube. In *VLDB*, pages 241–252, 2005.

[344] J. Zhang, X. Long, and T. Suel. Performance of compressed inverted list caching in search engines. WWW '08, pages 387–396, USA, 2008.

[345] Q. Zhang, F.M. Suchanek, L. Yue, and G. Weikum. TOB: Timely ontologies for business relations. In *WebDB*, 2008.

[346] S. Zhang, S. Li, and J. Yang. GADDI: distance index based subgraph matching in biological networks. In *EDBT*, pages 192–203, 2009.

[347] S. Zhang, S. Li, and J. Yang. SUMMA: subgraph matching in massive graphs. In *CIKM*, pages 1285–1288, 2010.

[348] P. Zhao and J. Han. On graph query optimization in large networks. *PVLDB*, 3(1):340–351, 2010.

[349] P. Zhao, X. Li, D. Xin, and J. Han. Graph cube: on warehousing and OLAP multidimensional networks. In *SIGMOD'11*, pages 853–864, 2011.

[350] P. Zhao, J. Xu Yu, and P.S. Yu. Graph indexing: Tree + delta >= graph. In *VLDB*, pages 938–949, 2007.

[351] W. Zhou, Q. Fei, S. Sun, T. Tao, A. Haeberlen, Z.G. Ives, B.T. Loo, and M. Sherr. NetTrails: a declarative platform for maintaining and querying provenance in distributed systems. In *SIGMOD Conference*, pages 1323–1326, 2011.

[352] L. Zou, L. Chen, M. Tamer Özsu, and D. Zhao. Answering pattern match queries in large graph databases via graph embedding. *VLDB J.*, 21(1):97–120, 2012.

[353] L. Zou, P. Peng, and D. Zhao. Top-K possible shortest path query over a large uncertain graph. In *WISE*, pages 72–86, 2011.

Appendix A

FPSPARQL Experimental Evaluation

In this section we provide an experimental evaluation of FPSPARQL query engine. We utilized IBM DB2 as a back-end database. All experiments were conducted on a HP system with a 2.67Ghz Core2 Quad processor, 4 GBytes of memory, and running a 64-bit Windows 7. We use the e-Enterprise course dataset introduced in Chapter 4 as an evaluation example. We compare our system with HyperGraphDB[1] [190] (an open-source graph database) and present query running time measurements in Section A.0.1. We provide the ability to call external graph reachability algorithms for path node queries. We discuss the quality of finding paths by different approaches in Section A.0.2.

A.0.1 Query Execution Time

We evaluated the performance of the FPSPARQL query engine compared to one of the well-known graph databases, the HyperGraphDB. There is no query language for HyperGraphDB and querying is performed through special purpose APIs. These APIs are based on conditional expressions that a user creates, submits to the query system and receives a set of nodes as the result. We extracted and simulated over one million events (about 25 million triples) out of e-Enterprise course database to

[1] http://www.hypergraphdb.org/

generate a large RDF file (1.9 GByte). It took 22.8 minutes to load the input RDF file into FPSPARQL relational RDF store. HyperGraphDB manages storage as a set of files in a directory. To create a database and load the same file as input into HyperGraphDB, we have implemented a loader. The loader took 52.2 minutes to load the input file. In the appendix we present FPSPARQL query samples that were useful for our e-Enterprise course collaborators. For each query expressed in English, we construct a FPSPARQL query and its equivalent SQL queries, generated by FPSPARQL-to-SQL translation algorithm.

Figure A.1 illustrates the query execution time for each FPSPARQL query, its SPARQL equivalent, and HyperGraphDB API. Query1 is a folder node construction query. Query1 runs a bit faster on SPARQL, compared to FPSPARQL. The reason could be a small overhead for storing the folder in FPSPARQL query. Both SPARQL and FPSPARQL executed faster than HyperGraphDB. Query2 is a folder node selection query. The execution time of FPSPARQL shows that applying queries on folder nodes, improves the query processing time of many complex queries. The equivalent SPARQL query should apply the condition on the whole graph which takes longer to execute. The execution time of HyperGraphDB is much better than SPARQL query, but not comparable to FPSPARQL query. Query3 is a folder node selection query. In FPSPARQL, the query applied on the composition of two constructed folder nodes. For HyperGraphDB we generate same folders as hypergraphs and applied a query on the composition of them. Figure A.1 shows the better +performance of FPSPARQL compared to its equivalent SPARQL query and HyperGraphDB API.

Query4 is a path node construction query. FPSPARQL provides the ability to call external graph reachability algorithms in path node queries (see Section A.0.2). It took 15.7 minutes, for the FPSPARQL engine, to parse the regular expressions and explore potential paths. As the result one path was discovered. HyperGraphDB has APIs providing the traversal algorithm (breadth-first or depth-first). The performance for these APIs depends on the incidence index and the efficient caching of incidence sets. We applied efficient index and caching to run query4 on HyperGraphDB. The query took 63.8 minutes to explore potential paths. As the result

one path was discovered. We stored the path (manually) as a hypergraph to use in query5. Query 4 is not supported in SPARQL query language.

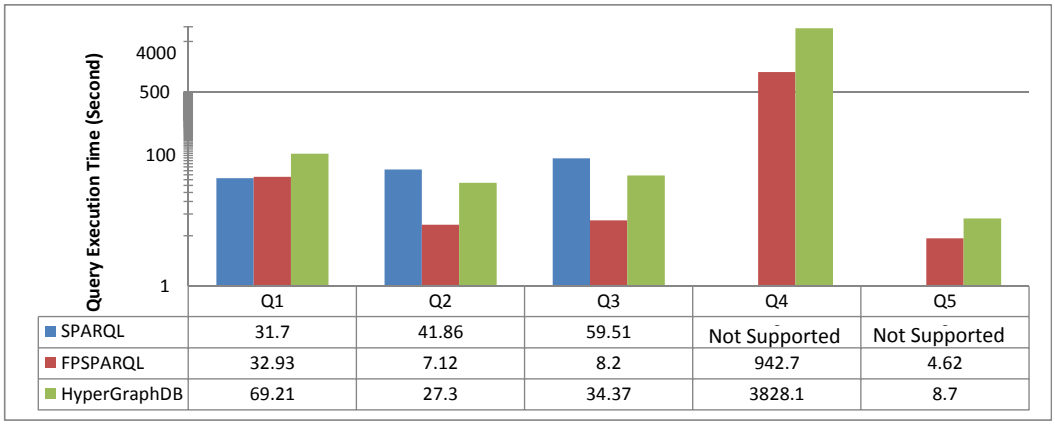
Query5 is a path node selection query. In FPSPARQL, the query applied on the path node constructed in query4. In HyperGraphDB, the query applied on the path generated in query4 which stored manually as a hypergraph. HyperGraphDB does not support the automatic construction and selection of paths. Also it does not provide the ability to call external traversal algorithms. Query 5 is not supported in SPARQL query language. Figure A.1 illustrates the performance of these queries.

A.0.2 Graph Reachability Analysis

We developed an interface to support various graph reachability algorithms [19] such as Transitive Closure, GRIPP, Tree Cover, Chain Cover, Path-Tree Cover, and Shortest-Paths [158]. In general, there are two types of graph reachability algorithms [19]: (1) algorithms traversing from starting vertex to ending vertex using breadth-first or depth-first search over the graph, and (2) algorithms checking whether the connection between two nodes exists in the edge transitive closure of the graph. Considering $G = (V, E)$ as a directed graph that has n nodes and m edges, the first approach incurs high cost as $O(n + m)$ time which requires too much time in querying. The second approach results in high storage consumption in $O(n^2)$ which requires too much space. In this experiment, we used the GRIPP [318] algorithm which has the querying time complexity of $O(m - n)$, index construction time complexity of $O(n + m)$, and index size complexity of $O(n + m)$.

A.0.3 FPSPARQL Queries

In this section we present FPSPARQL query samples that were useful for our e-Enterprise course collaborators. For each query expressed in English, we construct a FPSPARQL query and its equivalent SQL query, generated by FPSPARQL-to-SQL translation algorithm.



	Q1	Q2	Q3	Q4	Q5
SPARQL	31.7	41.86	59.51	Not Supported	Not Supported
FPSPARQL	32.93	7.12	8.2	942.7	4.62
HyperGraphDB	69.21	27.3	34.37	3828.1	8.7

Figure A.1: Query Execution Times.

Query 1. [Folder Node Construction] Group all the events happened in the context of *brainstorming* phase during "semester 2, 2009", in a folder named "brainstorming09s2". Brainstorming phase *start time* is '19 July 2009' and *end time* is '8 August 2009'.

FPSPARQL:

```
1  fconstruct brainstorming09s2 as ?fn
2  select ?e
3  where{
4   ?fn @description 'related events...'.
5   ?e @type Event.
6   ?e @timestamp ?date.
7   FILTER (?date > "2009-07-19" ^^xsd:date &&
8   ?date > "2009-08-08" ^^xsd:date).
9  }
```

SQL:

```
1   @folderID <- generate a unique folderID
2   ...
3   insert into NULLID.EntityStore
```

```
4   (subject , predicate , object)
5   values
6   (@folderID , '@Name' , 'brainstorming09s2');
7   insert into NULLID.FOLDERSTORE
8   (folderid , subject , predicate , object)
9   select @folderID as folderid ,
10                      subject ,
11                      predicate ,
12                      object
13  from NULLID.GraphStore
14  where subject in
15  (SELECT r.e AS e
16  FROM (
17  Select es1.subject AS e, es2.object AS date
18  From NULLID.EntityStore es1, NULLID.EntityStore es2
19  Where es1.predicate = '@type' AND es1.object = 'Event' AND
20  es2.predicate = '@timestamp' AND es1.subject = es2.subject  AND
21  (DATE(SUBSTRING(es2.object,1,10,CODEUNITS32)) > DATE(2009-07-19)
22  AND
23  DATE(SUBSTRING(es2.object,1,10,CODEUNITS32)) > DATE(2009-08-08))
24  ) AS r );
```

Query 2. [Folder Node Selection] Return the list of artifacts that have been part of update events which are triggered by a comment event in the context of *brainstorming* phase during "semester 2, 2009", i.e. the folder we created in Query1.

FPSPARQL:

```
1   (brainstorming09S2) apply (
2   select ?a
3   where {
4    ?e @type 'Event'.
```

```
5    ?e @activityType 'update'.
6    ?e @ArtifactName ?a.
7    ?e wasTriggeredBy ?x.
8    ?x @type 'Event'.
9    ?x @activityType 'comment'.
10  })
```

SQL:

```
1   SELECT r.a AS a FROM (
2   Select es1.subject AS e, es3.object AS a, fs1.object AS x
3   From NULLID.EntityStore es1, NULLID.EntityStore es2,
4   NULLID.EntityStore es3, NULLID.FOLDERSTORE fs1,
5   NULLID.EntityStore es4, NULLID.EntityStore es5
6   Where es1.predicate = '@type' AND
7   es1.object = 'Event' AND
8   es4.object = 'Event' AND
9   es2.predicate = '@activityType' AND
10  es2.object = 'update' AND
11  es3.predicate = '@ArtifactName' AND
12  fs1.predicate in (
13  select subject
14  from NULLID.EntityStore
15  where predicate = '@Label' AND object = 'wasTriggeredBy' ) AND
16  fs1.FolderID in (
17  Select subject
18  from NULLID.EntityStore
19  where predicate = '@Name' AND
20  object = 'brainstorming09S2') AND
21  es5.object = 'comment' AND
22  es1.subject = es2.subject AND
```

```
23  es1.subject = es3.subject AND
24  es1.subject = fs1.subject AND
25  es1.object = es4.object AND
26  es4.subject = es5.subject AND
27  fs1.object = es4.subject AND
28  es1.predicate=es4.predicate AND
29  es2.predicate = es5.predicate ) AS r
```

Query 3. [Folder Node Selection] Return the list of users who were involved in updating an artifact during *brainstorming* and *design* phase of semester 1, 2010. We construct two folders of all events that happened in the context of brainstorming phase (brainstorming10s1), and design phase (design10s1) during "semester 1 2010".

FPSPARQL:

```
1  (brainstorming10s1 union design10s1) apply (
2   select ?u
3   where {
4    ?e @type 'Event'.
5    ?e @activityType 'update'.
6    ?e @UseName ?u.
7   }
8  )
```

SQL:

```
1   SELECT r.u AS u FROM (
2   Select es1.subject AS e, es3.object AS u
3   From NULLID.EntityStore es1, NULLID.EntityStore es2,
4   NULLID.EntityStore es3
5   Where es1.predicate = '@type' AND
```

```
6   es1.object = 'Event' AND
7   es2.predicate = '@activityType' AND
8   es2.object = 'update' AND
9   es3.predicate = '@UseName' AND
10  es1.subject = es2.subject AND
11  es1.subject = es3.subject AND
12  es1.subject in (
13 select subject
14  from NULLID.FOLDERSTORE
15  where folderid in (
16  select subject
17  from NULLID.EntityStore
18  where predicate = '@name' AND
19  subject = 'brainstorming10s1')
20  union
21  select subject
22  from NULLID.FOLDERSTORE
23  where folderId in (
24  select subject
25  from NULLID.EntityStore
26  where predicate = '@name' AND
27  subject = 'design10s1') ) ) AS r
```

Query 4. [Path Node Construction] Construct a path between the event that generates brainstorming document (brainDoc.doc), and the event that generates design document (designDoc.doc) which were rendered by project4 members during semester 2, 2009. This path should contain the pattern of an event responding to a bug report in the Wiki.

FPSPARQL:

```
1   pconstruct myPathNode
```

```
2   (?startNode,?endNode,(?e ?n)* ?e ?node ?e (?n ?e)* )
3   where {
4    ?startNode @type 'Event'.
5    ?startNode @activityType 'generate'.
6    ?startNode @artifactName 'brainDoc.doc'.
7    ?startNode @UserGroup 'project4'.
8    ?startNode @timestamp ?date.
9    ?endNode @type 'Event'.
10   ?endNode @activityType 'generate'.
11   ?endNode @artifactName 'designDoc.doc'.
12   ?endNode @UserGroup 'project4'.
13   ?endNode @timestamp ?date.
14   ?n @isA 'entityNode'.
15   ?n @type 'Event'.
16   ?n @timestamp ?date.
17   ?e @isA 'edge'.
18   ?node @type 'Event'.
19   ?node @activityType 'response'.
20   ?node @layer 'Wiki'.
21   ?node @layerPart 'bug'.
22   ?node @timestamp ?date.
23   FILTER (?date > "2009-07-19" ^^xsd:date &&
24   ?date > "2009-11-04" ^^xsd:date).
25  }
```

SQL: A graph reachability algorithm used (see Section A.0.2).

Query 5. [Path Node Selection] Return the list of artifacts that generated between the path constructed in Query4.

FPSPARQL:

```
1  (myPathNode) apply (
2   select ?a
3   where {
4   ?e @type 'Event'.
5   ?e @activityType 'generate'.
6   ?e @ArtifactName ?a.
7   }
9  )
```

SQL:

```
1   SELECT r.a AS a FROM (
2   Select es1.subject AS e, es3.object AS a
3   From NULLID.EntityStore es1, NULLID.EntityStore es2,
4   NULLID.EntityStore es3
5   Where es1.predicate = '@type' AND
6   es1.object = 'Event' AND
7   es2.predicate = '@activityType' AND
8   es2.object = 'generate' AND
9   es3.predicate='@ArtifactName' AND
10  es1.subject=es2.subject AND
11  es1.subject = es3.subject AND
12  es1.subject in (
13  SELECT subject
14  from NULLID.PathStore
15  Where PathId in (
16  SELECT subject
17  from NULLID.EntityStore
18  Where predicate = '@name' AND object='myPathNode'))) AS r
```