DISSERTATION

submitted

to the

Combined Faculty for the Natural Sciences and Mathematics

of

Heidelberg University, Germany

for the degree of

Doctor of Natural Sciences

Put forward by

M.Sc. Ajinkya Prabhune

Born in: Pune, India

Oral examination:.....

GENERIC AND ADAPTIVE METADATA MANAGEMENT FRAMEWORK FOR SCIENTIFIC DATA REPOSITORIES

Advisors: Prof. Dr. Jürgen Hesser and Prof. Dr. Michael Gertz

Abstract

Rapid progress in technology has led to multifold advancements in data acquisition and processing in various research disciplines. These advancements have led to a tremendous growth in data and metadata that are being generated by scientific experiments. Regardless of any specific research domain, research practices are widely becoming data and metadata driven. As a consequence, research communities, funding agencies, and universities are intensifying their efforts in building data repositories for handling scientific data. Broadly speaking, the goals of a scientific data repository are to provide long-term data archiving, make data searchable for reuse and referencing, capture provenance for enabling data reproducibility, and provide metadata and annotation support for imparting domain-specific knowledge necessary for data interpretation. However, scientific data repositories are highly complex frameworks that comprise components such as algorithms for data compression and long-term data archiving, metadata and annotation management frameworks, workflow provenance and interoperability between heterogeneous workflow systems, authorization and authentication infrastructures, and visualization tools for data interpretation.

In this thesis, we present a modular scientific data repository architecture to support arbitrary research communities in handling their data and metadata lifecycle. This architecture consists of components representing four areas of research. The first component is a data transfer client that provides a generic interface for allowing ingest and access to data from scientific data acquisition systems.

The second component is the MetaStore framework, which is an adaptive metadata management framework that provides handling of both static and dynamic metadata models. For handling arbitrary metadata schemas, the MetaStore framework is designed on the component-based dynamic composition design pattern. The MetaStore is further enriched with an annotation framework for handling dynamic metadata.

The third component is an extension of the MetaStore framework. It provides the automated handling of provenance metadata for Business Process Execution Language (BPEL) based workflow management systems. To automate the translation of BPEL workflows into the ProvONE model, we have designed and implemented the Prov2ONE algorithm. The availability of complete BPEL provenance in ProvONE not only allows aggregate analysis of workflow definitions with their execution traces but also enables provenance interoperability.

The fourth component of the scientific data repository is the ProvONE-Provenance Interoperability Framework (P-PIF). This component enables the interoperability of provenance from heterogeneous workflow management systems. The P-PIF consists of two parts. First, we extend the Prov2ONE algorithm for Simple Conceptual Unified Flow Language (SCUFL) and Modeling Markup Language (MoML) workflow specifications. Second, we provide workflow management system specific adapters that provide extraction, translation, and modeling of retrospective provenance in the ProvONE model. The availability of heterogeneous provenance traces in the ProvONE

model allows us to compare, analyze, and query provenance from different workflow management systems.

Each component of the scientific data repository presented in this thesis is assessed. The performance of the data transfer client using a standard protocol for nanoscopy datasets is evaluated. The MetaStore framework is evaluated for following two conditions. First, the metadata ingest and full-text search performance under different databases configurations are tested. Second, to show the comprehensive coverage of functionalities provided by MetaStore, we present a feature-based evaluation of MetaStore against existing metadata management systems. For the assessment of the P-PIF, first, we proved the correctness and completeness of our Prov2ONE algorithms. Additionally, we evaluated the ProvONE prospective graph patterns produced by the Prov2ONE BPEL algorithm against the established BPEL control-flow patterns. Second, to show that P-PIF is a sustainable framework that adheres to standards, we present a feature-based evaluation of P-PIF against existing provenance interoperability frameworks. These assessments reveal the superiority and advantages of the various components presented in this thesis over existing systems.

Zusammenfassung

Der rapide technologische Fortschritt hat in verschiedenen Forschungsdisziplinen zu vielfältigen Weiterentwicklungen in Datenakquise und -verarbeitung geführt. Hieraus wiederum resultiert ein immenses Wachstum an Daten und Metadaten, generiert durch wissenschaftliche Experimente. Unabhängig vom konkreten Forschungsgebiet ist die wissenschaftliche Praxis immer stärker durch Daten und Metadaten gekennzeichnet. In der Folge intensivieren Universitäten, Forschungsgemeinschaften und Förderagenturen ihre Bemühungen, wissenschaftliche Daten effizient zu sichten, zu speichern und auszuwerten. Die wesentlichen Ziele wissenschaftlicher Daten-Repositorien sind die Etablierung von Langzeitspeicher, der Zugriff auf Daten, die Bereitstellung von Daten für die Wiederverwendung und deren Referenzierung, die Erfassung der Datenquelle zur Reproduzierbarkeit sowie die Bereitstellung von Metadaten, Anmerkungen oder Verweisen zur Vermittlung domänenspezifischen Wissens, das zur Interpretation der Daten notwendig ist. Wissenschaftliche Datenspeicher sind hochkomplexe Systeme, bestehend aus Elementen aus unterschiedlichen Forschungsfeldern, wie z. B. Algorithmen für Datenkompression und Langzeitdatenarchivierung, Frameworks für das Metadaten- und Annotations-management, Workflow-Provenance und Provenance-Interoperabilität zwischen heterogenen Workflowsystemen, Autorisierungs und Authentifizierungsinfrastrukturen sowie Visualisierungswerkzeuge für die Dateninterpretation.

Die vorliegende Arbeit beschreibt eine modulare Architektur für ein wissenschaftliches Datenarchiv, die Forschungsgemeinschaften darin unterstützt, ihre Daten und Metadaten gezielt über den jeweiligen Lebenszyklus hinweg zu orchestrieren. Diese Architektur besteht aus Komponenten, die vier Forschungsfelder repräsentieren. Die erste Komponente ist ein Client zur Datenübertragung (“data transfer client”). Er bietet eine generische Schnittstelle für die Erfassung von Daten und den Zugriff auf Daten aus wissenschaftlichen Datenakquisesystemen.

Die zweite Komponente ist das MetaStore-Framework, ein adaptives Metadaten-Management-Framework, das die Handhabung sowohl statischer als auch dynamischer Metadatenmodelle ermöglicht. Um beliebige Metadatenschemata behandeln zu können, basiert die Entwicklung des MetaStore-Frameworks auf dem komponentenbasierten dynamischen Kompositions-Entwurfsmuster (component-based dynamic composition design pattern). Der MetaStore ist außerdem mit einem Annotationsframework für die Handhabung von dynamischen Metadaten ausgestattet.

Die dritte Komponente ist eine Erweiterung des MetaStore-Frameworks zur automatisierten Behandlung von Provenance-Metadaten für BPEL-basierte Workflow-Management-Systeme. Der von uns entworfene und implementierte Prov2ONE Algorithmus übersetzt dafür die Struktur und Ausführungstraces von BPEL-Workflow-Definitionen automatisch in das Provenance-Modell ProvONE. Hierbei ermöglicht die Verfügbarkeit der vollständigen BPEL-Provenance-Daten in ProvONE nicht nur eine aggregierte Analyse der Workflow-Definition mit ihrem Ausführungstrace, sondern

gewährleistet auch die Kompatibilität von Provenance-Daten aus unterschiedlichen Spezifikationssprachen.

Die vierte Komponente unseres wissenschaftlichen Datenarchives ist das Provenance-Interoperabilitätsframework ProvONE - Provenance Interoperability Framework (P-PIF). Dieses gewährleistet die Interoperabilität von Provenance-Daten heterogener Provenance-Modelle aus unterschiedlichen Workflowmanagementsystemen. P-PIF besteht aus zwei Komponenten: dem Prov2ONE-Algorithmus für SCUFL und MoML Workflow-Spezifikationen und Workflow-Management-System-spezifischen Adaptern zur Extraktion, Übersetzung und Modellierung retrospektiver Provenance-Daten in das ProvONE-Provenance-Modell. P-PIF kann sowohl Kontrollfluss als auch Datenfluss nach ProvONE übersetzen. Die Verfügbarkeit heterogener Provenance-Traces in ProvONE ermöglicht das Vergleichen, Analysieren und Anfragen von Provenance-Daten aus unterschiedlichen Workflowsystemen.

Wir haben die Komponenten des in dieser Arbeit vorgestellten wissenschaftlichen Datenarchives wie folgt evaluiert: für den Client zum Datentrasfer haben wir die Daten-übertragungsleistung mit dem Standard-Protokoll für Nanoskopie-Datensätze untersucht. Das MetaStore-Framework haben wir hinsichtlich der folgenden beiden Aspekte evaluiert. Zum einen haben wir die Metadatenaufnahme und Volltextsuchleistung unter verschiedenen Datenbankkonfigurationen getestet. Zum anderen zeigen wir die umfassende Abdeckung der Funktionalitäten von MetaStore durch einen funktionsbasierten Vergleich von MetaStore mit bestehenden Metadaten-Management-Systemen. Für die Evaluation von P-PIF haben wir zunächst die Korrektheit und Vollständigkeit unseres Prov2ONE-Algorithmus bewiesen und darüber hinaus die vom Prov2ONE BPEL-Algorithmus generierten Prognose-Graphpattern aus ProvONE gegen bestehende BPEL-Kontrollflussmuster ausgewertet. Um zu zeigen, dass P-PIF ein nachhaltiges Framework ist, das sich an Standards hält, vergleichen wir außerdem die Funktionen von P-PIF mit denen bestehender Provenance-Interoperabilitätsframeworks. Diese Auswertungen zeigen die Überlegenheit und die Vorteile der einzelnen in dieser Arbeit entwickelten Komponenten gegenüber existierenden Systemen.

Acknowledgments

Undertaking this Ph.D. has been an amazing experience for me. Achieving this feat would not have been possible without the advice and support of many people. Firstly, I would like to express my sincere gratitude to my supervisor Dr. Rainer Stotzka for giving me the opportunity to work as a Ph.D. student at Karlsruhe Institute of Technology. I am grateful to him for supporting, motivating and enlightening me with the first glance of research. He always believed in me and stood by me during the difficult time, and pushed me to achieve beyond my limits. Besides my supervisor, I would like to express my sincere gratitude to my advisors, Prof. Dr. Jürgen Hesser and Prof. Dr. Michael Gertz, at the Faculty of Mathematics and Computer Science of Heidelberg University. There is so much I have learned from them that I could easily write one more chapter explaining the endless support and guidance they have given me throughout my thesis. Their encouragement made me push myself to my true potential. What amazes me is that despite their busy schedule, they were able to reserve time to read and provide detailed feedback and comments for all the publications, presentations and the thesis. Not only did they guide me in my research, but also taught me a great deal in academic writing and presentation skills. I would also like to thank the dissertation committee for their time, efforts, and feedback.

During my time at Karlsruhe Institute of Technology, my colleagues contributed in different ways to complete my thesis. I am grateful to Thomas Jejkal and Volker Hartmann for helping me integrate into the team and always being available for discussions and providing me feedback.

I would like to thank my friends and family for supporting me throughout my thesis and encouraging me to strive towards my goal. My parents Prof. Dr. Asmita Prabhune and Dr. Ashutosh Prabhune, you both are my inspiration. The benchmark that you have set in the family made me challenge myself to reach my potential, many thanks to my sisters Anushree and Apoorva for often visiting me and giving me the needed mental support. Many thanks to my close friend, Prof. Dr. Barbara Sprick with whom I could share my ideas and get valuable insights on them. Also, special thanks to the staff and my students from SRH Hochschule.

Last but not least, I would like to thank my beloved wife, Priyanka Sakundarwar for her support and faith in me. Priyanka, you have been the pillar of strength for me and helped me get through this period in the most positive way. You made sure that there is more to life other than the Ph.D. Thank you to my mother-in-law and father-in-law for raising such a wonderful girl.

To my parents and my wife.

Contents

List of Abbreviations

BPEL	**B**usiness **P**rocess **E**xecution **L**anguage
DAQ	**D**ata **A**cquisition **S**ystem
DC	**D**ublin **C**ore
GCS API	**G**eneric **C**lient **S**ervice **A**pplication **P**rogramming **I**nterface
HPC	**H**igh **P**erformance **C**omputing
IR	**I**nstitutional **R**epository
METS	**M**etadata **E**ncoding and **T**ransmission **S**tandard
MoML	**M**odeling **M**arkup **L**anguage
OAI-PMH	**O**pen **A**rchives **I**nitiative–**Protocol** for **M**etadata **H**arvesting
OPM	**O**pen **P**rovenance **M**odel
PREMIS	**PRE**servation **M**etadata: **I**mplementation **S**trategies
KIT DM	**K**arlsruhe **I**nstitute of **T**echnology **D**ata **M**anager
LSDF	**L**arge **S**cale **D**ata **F**acility
P-PIF	**P**rovONE–**P**rovenance **I**nteroperability **F**ramework
RDF	**R**esource **D**escription **F**ramework
SDR	**S**cientific **D**ata **R**epository
SCUFL	**S**imple **C**onceptual **U**nified **L**anguage
SKOS	**S**imple **K**nowledge **O**rganization **S**ystem
SPARQL	**SPARQL P**rotocol and **R**DF **Q**uery **L**anguage
WADM	**W**eb **A**nnotation **D**ata **M**odel
WfMS	**W**ork**f**low **M**anagement **S**ystem

Chapter 1

Introduction

The rapid growth in modern e-Science infrastructures and applications in various fields of research has led to an exponential growth in the volume of data that is now generated. Not only the technology for data acquisition has drastically improved but also the infrastructure to handle this data, and the numerous applications for processing the data have increased. However, to effectively use these huge volumes of data and to facilitate an overall control over the scientific data, metadata has become a critical aspect of scientific research. The commonly adopted definition of Metadata is "the data that provides information about other data." However, there is more to this definition, in scientific research, metadata is a research topic in itself, and serves multiple purposes throughout the life cycle of data. For example, for discovering the data, metadata describing the data is required, for organizing complex data elements in a digital storage system, the metadata describing its structure and relationship is required, and for orchestrating complex scientific workflows through Workflow Management Systems (WfMS), metadata describing each data processing task and the execution order is necessary. To give an idea of the volume of data that is generated in various scientific areas, in the research field of astronomy, the Large Synoptic Survey Telescope (LSST) is generating around 13 terabytes each day. In genome analysis, a single sequenced human genome produces approximately 140 gigabytes of data, and for sequencing multiple genomes and tracking gene interactions, it is estimated that about few hundreds of petabytes of data will be generated. The climate and weather research group at the NASA Center for Climate Simulation (NCCS) has collected around 32 petabytes of data with various tools and applications for data analysis and generating valuable results. In the research area of health sciences, the biology data repository in the European Bioinformatics Institute UK is currently hosting 20 petabytes of data describing genes, proteins, and small molecular structures [40].

A similar explosion of data can be seen in the field of localization–microscopy (nanoscopy). With the advancements in the e-Science infrastructures and the development of a novel imaging technique in nanoscopy has resulted in the generation of huge volumes of data for each investigation. Currently, the data volume generated by each investigation, i.e., the images generated for approximately 50 sections (1 section covers only a few µm2) of a cell membrane is around 150-200 terabytes of data. This

volume of data is generated for only a single color channel, however, with the activation of all the color channels, the amount of data for a single investigation is expected to be approximately one petabyte [35, 36]. Regarding the lifecycle of a nanoscopy investigation, a routine nanoscopy investigation is a time and effort intensive process that can last for few days or a couple of weeks. Regular research activities comprise the following data curation steps: (a) configuration of the microscope, (b) in vitro preparation of the specimen and the dyes, (c) generating the raw-data and metadata, (d) execution of different data-processing workflows over the raw-data to generate valuable results. Not only in the research area of nanoscopy is such data curation routine observed but also similar data curation steps are seen in various other research communities. For example, (1) In the case of radiology clinical research (angioscopy), patient images are acquired by various modalities; for example, the Computer Tomography (CT) scanner generates CT scans for the patient under investigation and stores these images in a Picture Archiving and Communication System (PACS) server for further reuse. The images and its associated metadata are formatted in the Digital Imaging and Communications in Medicine (DICOM) standard. However, for enabling further analysis of these scans, it is necessary to make the DICOM datasets available to the researchers, and for that, there is a need to extract, model, and store the DICOM metadata in a database. Using the metadata to discover the Computer Tomography (CT) scans, custom image-processing workflows, defined by the research community can be triggered. The results generated by these workflows are beneficial for the physicians in composing patient-specific treatment plans [103, 126]. (2) In the case of eCodicology, for performing image analysis, the medieval manuscripts are digitized to create an image stack. Each manuscript is associated with bibliography metadata that uniquely describes each manuscript. This image stack is the input for image processing workflows that are responsible for extracting different micro-structural features from the manuscript, such as color calibration, page layout analysis, text and image segmentation, and feature extractions [31].

Considering the use cases mentioned before, it is clear that the data and metadata generated throughout the life cycle of the experiment need to be curated for producing additional valuable results, archived for long-term maintenance, accessed for enabling reuse and knowledge dissemination, and reproduced for validating the research results. Moreover, in e-Science, data curation, archiving, preservation, access, and metadata management are key research activities [99, 150]. Among the various areas in the field of e-Science, this thesis primarily focuses on designing an extensible scientific data repository framework with the capability for handling heterogeneous metadata standards. Secondarily, as a subtopic in metadata research, for enabling reproducibility of experimental results, the thesis focuses on enabling automated capturing of provenance in BPEL-based WfMS, and with an extension for provenance interoperability among heterogeneous WfMSs. That is for allowing analysis of heterogeneous provenance traces by modeling them in a common provenance model.

We present the motivation and research objectives of our study in Section 1.1. Then, the key-challenges faced in research areas of scientific data repository framework, metadata management, provenance tracking and interoperability in WfMSs are presented in Section 1.2. The contributions in each of these research areas are summarized in Section 1.3, with the thesis outline in Section 1.4.

1.1 Motivation and Research Objectives

Motivated by the numerous applications in the research areas of scientific data repository frameworks for big data in research, metadata management framework, and provenance interoperability, in this section we present the various aspects that led us to build comprehensive scientific data repository framework.

Scientific data repository framework

In e-Science, the scientific data repository framework is the backbone that supports a successful data curation lifecycle for scientific data [78]. Buyya and colleagues define a repository as simple data store for datasets. However, we consider a data repository more than just a simple data store, and define a repository as a modular system with a set of services for transferring, archiving, and persisting the datasets. Various research areas are driven by data, and with the steady increase of data, the tools and frameworks for handling this data are also on the rise [79]. However, these tools and frameworks are independent of each other and have to be orchestrated by IT experts, which is not only time-consuming but also a costly task. Moreover, as these tools and frameworks do not provide the functionalities necessary for storing and organizing the data and metadata, it leads to the adoption of haphazard storage techniques. Typically, the data is stored and shared using local hard drives, CD-ROMs, DVDs, or USB drives [17]. To overcome this haphazard handling of scientific data, research communities have started adopting scientific data repository frameworks. Below are some of the applications of a scientific data repository framework:

- long-term archival and organization of the data,
- data transfer (human and machine ingest) from geographically separated locations,
- controlled access and sharing of data,
- extensible metadata schema support with OAI-PMH compliance.

Metadata management framework

One of the basic functionality that a scientific data repository framework should provide is its support for handling heterogeneous metadata standards. Metadata is crucial for managing the lifecycle of data; numerous data management tasks are directly dependent on the metadata. For example, descriptive metadata is necessary for discovering and identifying data, complex data organization can be described using structural

metadata, and access rights and preservation details can be specified using the administrative metadata [70]. Additionally, the prospective provenance metadata is highly valuable for tracing the history of data and essential for reproducibility of the results. Currently, organized as per the different metadata categories, there exist more than two hundred different metadata standards [93]. To support research communities in handling their metadata, the research in metadata is bifurcated, on the one hand, the scientific data repository are enriched with functionalities for handling metadata [55, 76, 149, 162], and on the other hand, standalone metadata management frameworks that aim at providing a generic metadata management solution are developed [19, 158]. Following are some of the applications of metadata in scientific research:

- discovering, access, sharing and reusing of data,
- visualization of complex relationships within data,
- OAI complaint large scale metadata harvesting.

WfMS and Provenance

In addition to handling the large and complex data and its associated metadata, scientific experiments composed of multiple steps are critical for researchers to produce valuable results that assist in improving the research domain. The systematic description of such multi-step experiment is typically expressed using a workflow specification (workflow language). Workflows allow researchers to automate and repeat these complex-processing steps. With the execution of workflows, it is necessary to handle the provenance for allowing reproducibility of the experiment and validating the results. Comprehensive provenance comprises two parts: (1) Prospective provenance that captures the workflow specification or the workflow recipe. (2) Retrospective provenance captures the runtime events that occur during the workflow execution [181]. Until recently, handling the retrospective provenance was the primary focus, and to model the retrospective provenance, the Open Provenance Model (OPM) [114] and the PROV model [112] were established. However, these models are limited in modeling only the retrospective provenance and not the prospective provenance. To overcome this limitation and enable modeling of both prospective and retrospective in the same provenance model, the ProvONE provenance model was proposed. Currently, there exist no technique for automatically capturing ad hoc workflow specifications in the ProvONE model, and especially there exists no solution that can capture the provenance for a BPEL-based WfMS. The prime reason for choosing a BPEL-based WfMS is due to the fact that BPEL is the de facto standard and there are multiple implementations of WfMSs from major vendors that are based on BPEL specification, thus, allowing easy portability among different BPEL-based WfMSs. Moreover, as BPEL supports Service Oriented Architecture (SOA), various scientific communities have adopted BPEL WfMS, especially for orchestrating Grid resources [52, 138, 165]. Integrating a scientific data repository with a WfMS that supports automated capturing of provenance has many applications. Some of the applications are outlined below.

- Automating complex repetitive tasks.
- Automated archiving of results generated during the workflow-execution.
- Documenting workflow and provenance for reproducibility.
- Sharing and analyzing provenance for improving results.

Provenance Interoperability

Provenance interoperability is allowing analysis of provenance traces from heterogeneous sources by modeling them in a common provenance model. Currently, to the best of our knowledge, for enabling provenance interoperability, the existing WfMSs support the export of only the retrospective provenance in either OPM or PROV model [112]. Additionally, initiated by the provenance queries described in the third provenance challenge, various standalone provenance interoperability frameworks based on the OPM were fabricated. However, few of the questions aimed for retrieving prospective provenance could not be answered due to the limitations of the OPM in handling the prospective provenance. Thus, for enabling analysis of heterogeneous provenance traces from different sources, it is necessary to model the complete provenance (prospective and retrospective) in a common provenance model like ProvONE. Following are the applications of having heterogeneous provenance traces in ProvONE.

- Compare and analyze workflows defined in different specifications.
- Validate scientific results by reproducing them in a different execution environment.
- Trace and analyze workflow evolution for determining optimum workflows.
- Enable collective analysis of both prospective and retrospective provenance.

Motivated by the limitations in the research areas of scientific data repository framework, metadata management, workflow and provenance handling with extension towards provenance interoperability, in this thesis, we primarily devote our efforts towards building a comprehensive scientific data repository framework. We consider the scientific data repository framework as the central overarching system that comprises metadata management, workflow and provenance handling, and provenance interoperability. Thus, in this thesis, we expand our efforts in multiple research areas that are relevant to achieving our primary objective, i.e., the scientific data repository framework.

1.2 Key Challenges

A comprehensive scientific data repository framework comprises of multiple components that are by them independent research topics. In this section, we summarize

and aggregate the challenges we faced for each research topic that contributed to the overall outcome of this thesis:

Large scale data transfer and storage
With the availability of distributed e-Science infrastructure, complex scientific experiments can now be performed on distributed resources. A critical aspect of distributed e-Science experiments is that the data needs to be shared among the different geographically distributed resources. Hence, one of the basic challenges in a distributed infrastructure is the seamless transfer of data between different resources. Also, for handling data volumes of different sizes, the challenge is to provision multiple data transfer protocols that can be optimized for maximizing the network utilization.

Metadata heterogeneity
Each metadata standard is unique and conforms to a given definition (metadata schema). It is practically impossible to design a metadata management framework that can handle all the metadata standards. Thus, the existing metadata management handling capabilities of the scientific data repository frameworks can support either a single metadata standard or at the most a few metadata standards simultaneously [7]. Following are some of the challenges that had to be tackled when implementing a generic metadata management framework.

1. *Repetitive software development lifecycles.* The basic functionalities that a metadata management framework should provide are the CRUD (create, retrieve, update, delete) operations. However, in practice to implement the CRUD operations for each registered metadata standard are a labor-intensive and a redundant task that involves a complete software development cycle.

2. *Metadata discovery.* A basic problem in metadata access is allowing querying over it, i.e., defining queries for retrieving the metadata stored for each metadata standard. Defining metadata standard specific queries is not a viable option, because the problems are twofold: (a) implementing metadata search queries for each registered metadata standard is a time-consuming and a labor-intensive task; (b) uncertainty to determine at runtime, which query needs to be executed for the search term requested by the user. Thus, the challenge is to provide a scalable metadata search irrespective of the metadata standard.

3. *Flexible data storage model.* For storing heterogeneous metadata standards with different schemas, it is not possible to define a single generic schema that can accommodate all the standards, because the attributes a metadata standard can include is varied. Moreover, each metadata category serves a different purpose, for example, descriptive, structural, technical, administrative metadata are for data interpretation, data discovery, and access management, whereas provenance

metadata is for analyzing the complex lineage of data. Therefore, the challenge is providing the appropriate data model that allows not only efficient storage of the metadata but also retrieval of it.

Provenance tracking in WfMS

In terms of enabling a BPEL-based workflow management system with automated provenance tracking following challenges need to be addressed.

1. *Distinguishing control-flow and data-flow activities.* BPEL supports both, the workflow execution constructs and the data sharing constructs. On the one hand, BPEL specification consists of multiple control structures activities that can be arbitrarily used to define the execution order of a workflow. Moreover, these control structure activities can be arbitrarily nested within each other to define highly intrinsic workflows. On the other hand, the data sharing between BPEL activities is described through variables, wherein the variables correspond to complex message types or element types. In practice, a message type can have an ad hoc data structure, and there is no restriction on the level of granularity that a workflow designer can utilize for sharing the data between the activities. In both the cases, the critical challenge is preserving the exact execution order and the data sharing defined in a workflow in the ProvONE prospective provenance.

2. *Noisy run time event traces (retrospective provenance).* Capturing the retrospective provenance in ProvONE is entirely dependent on the information that a WfMS exposes. For example, Apache ODE generates twenty-three events out of which not all are meant for retrospective provenance. The challenge is to extract the appropriate retrospective provenance from these events.

Provenance interoperability

Workflows are defined in different languages that adhere to the specifications established by the workflow engine. BPEL, SCUFL, and MoML are few of the prominent specifications that are available for defining workflows. For enabling interoperability among heterogeneous provenance traces following challenges have to be tackled.

1. *Heterogeneity of workflow specifications (prospective provenance).* In all the three-workflow specifications, the primary challenge is capturing of the exact execution order defined for both control-flow driven workflows and data-flow driven workflows in the ProvONE model. The secondary challenge is to normalize the attributes of heterogeneous prospective provenance traces to a homogeneous data model in ProvONE, such that it is consistent across all the workflow specifications. Addressing the second challenge is critical because it enables formulation of analysis queries over these heterogeneous provenance traces.

2. *Uniformity in the retrospective provenance traces.* Each WfMS exports the retrospective provenance in a data model that is proprietary to the WfMS. Moreover, the end-points through which each WfMS exposes its retrospective provenance are different. For example, Apache ODE provides a web-service that regularly publishes the run-time events, Taverna and Kepler WfMS stores the retrospective provenance in a database and provides data access objects (DAO) for accessing it. Similar to the normalization of the prospective provenance explained above, the challenge is to normalize the retrospective provenance in a homogeneous data model for enabling analysis querying over the heterogeneous provenance traces.

1.3 Contributions

This thesis includes several contributions, specifically in the field of scientific data repository for handling large volumes of data, metadata management framework, automated provenance tracking in WfMS, and provenance interoperability.

Scientific Data Repository. In the research area of scientific data repository systems, the first contribution of this thesis is the architecture of a scientific data repository framework for handling the nanoscopy data curation lifecycle. The framework is based on the principle of client-server architecture, wherein the complex functionalities necessary for handling the lifecycle of scientific data is implemented on the server-side, and a light-weight client is provisioned for allowing data ingest and access from ad hoc data acquisition systems. For enabling transfer of data from heterogeneous data acquisition systems, the client implements a systematic ingest and download workflow with support to multiple data transfer protocols. On the server-side the nanoscopy scientific data repository framework is built on KIT Data Manager, which is a generic and highly customizable data repository framework. For the storage of the data, the KIT Data Manager provides integration with Large Scale Data Facility (LSDF) infrastructure offering cache as well as an archive. Moreover, cluster-based execution of computing tasks is supported by LSDF framework for data-intensive computing. The proposed scientific data repository framework is adopted by two research disciplines and their results are published in papers "An Optimized Generic Client Service API for Managing Large Datasets within a Data Repository" [123] and "Managing Provenance for Medical Datasets" [126].

Adaptive Metadata Management Framework. In the research area of metadata, the thesis presents an automatically adapting metadata management framework for a scientific data repository system. We introduce this framework in our paper “MetaStore: A Metadata Management Framework for Scientific Data Repositories” [124]. For supporting ad hoc metadata models, we have designed the MetaStore framework entirely on NoSQL database technology. The MetaStore

framework is a comprehensive solution and provides multiple features, each having well-defined execution workflow. First, we show that for allowing automated metadata quality control, it is necessary to register the metadata in a dedicated metadata registry. Second, instead of manually creating the functions (services) for handling the registered metadata, we describe an automated method for generating the services and automatically adapting the MetaStore on the fly. Third, for allowing full-text search over the metadata, we describe the automated index creation process. Fourth, for enabling OAI-complaint metadata harvesting, we implement the services for OAI data provider. Furthermore, the MetaStore framework architecture is extended with an auxiliary layer for handling dynamic metadata and for improving the metadata quality control. This extension of MetaStore is presented in the paper "MetaStore: An adaptive metadata management framework for heterogeneous metadata models" [127], while the adoption of the MetaStore framework by different research communities is presented in the paper "The MASi Repository Service - Comprehensive, Metadata-driven and Multi-community Research Data Management" [73].

WfMS and Provenance Management. For enabling execution of scientific workflows, we integrate a WfMS into the scientific data repository. By integrating a WfMS with the scientific data repository framework, we leverage the functionality of both these systems. The WfMS allows workflows to be systematically described using the workflow specification supported by the WfMS, whereas, a scientific data repository offers the long-term storage and archival of the datasets generated during the execution of a workflow. However, a systematically defined workflow offers only the repeatability of the workflow and not the reproducibility. For enabling data reproducibility, it is necessary to capture the complete provenance trace along with data that is generated (raw-data, intermediate-date, and result data). In this thesis, we propose a provenance management framework for completely automating the capturing, modeling, and storing of provenance generated by a WfMS. This framework is based on the Prov2ONE algorithm that automates the translation of a BPEL workflow specification in ProvONE model. On this contribution, we have described a provenance management framework for BPEL-based WfMS in our paper “Prov2ONE: An Algorithm for Automatically Constructing ProvONE Provenance Graphs” [125]. For consistency reasons, the ProvONE provenance graphs are serialized in W3C specified Resource Description Framework (RDF), and the analysis queries are defined in the W3C recommended SPARQL query language.

Provenance Interoperability. For enabling provenance interoperability (i.e., allow querying over heterogeneous provenance traces), we propose a provenance interoperability framework. This framework serves as a bridge that allows seamless integration of WfMSs for capturing heterogeneous provenance traces in a common provenance model, namely ProvONE. The automated modeling of the comprehensive provenance is performed in two stages. First, we extend the Prov2ONE algorithm

for translating the SCUFL and MoML workflow specifications as ProvONE prospective RDF graph. Second, we propose WfMS specific adapter for capturing the runtime provenance and enriching the ProvONE prospective RDF graph with the retrospective provenance. With the availability of the provenance in a common provenance model, it allows analysis of heterogeneous provenance traces. For this, we also propose a set of SPARQL queries for analyzing these provenance traces.

1.4 Structure of the Thesis

The structure of the thesis is as follows.

Chapter 2. We introduce general concepts in the research areas of scientific data repository framework, metadata management systems, WfMS and provenance, and provenance interoperability.

Chapter 3. We propose the architecture of an extensible scientific data repository framework, its various components and its applicability to different research communities.

Chapter 4. We present the MetaStore framework. The MetaStore architecture is split into two parts: (1) For handling the static metadata, we explain the realization of the MetaStore as an automatically adapting metadata management framework. (2) For managing the dynamics metadata, the extended functionalities of the MetaStore for handling annotations and metadata quality enrichment using controlled-vocabularies are presented.

Chapter 5. We describe the provenance management framework that illustrates the integration of a WfMS in a scientific data repository with the automated capturing of provenance. In this chapter we explain the two-step process of collecting the complete provenance for scientific workflows. First, we describe the Prov2ONE algorithm for translating the workflow specification into the ProvONE provenance model. Second, to capture the runtime provenance the workflow engine provenance collector component of the provenance management framework is presented. We extend this result and present a provenance interoperability framework that encompasses the provenance from three WfMSs. We argue that for enabling querying over heterogeneous provenance traces from different WfMSs it is necessary to translate these provenance traces in a common provenance model. For this, we extend the Prov2ONE algorithm for the SCUFL and MoML specifications. Finally, we also extend the queries put forth by the provenance challenge, and present six novel queries for retrieving the prospective provenance.

Chapter 6. Finally, we present the summary of the dissertation and the future open issues are presented.

Chapter 2

Background and Related Work

This chapter describes the context of this work and how it fits in the broad research area of digital curation in e-Science. Within the complex infrastructure framework for e-Science experiments, this thesis primarily focuses on Scientific Data Repository (SDR) architectures. For this, in Section 2.1 we briefly introduce the concept of SDR or what is also called as Institutional Repository (IR). Also, in this section, we briefly describe an exemplary SDR, namely the KIT Data Manager, which is an extensible framework for building community-specific repositories. Metadata being one of the cornerstones of an SDR, in Section 2.2, we introduce the concept of metadata and the metadata management capabilities of existing SDRs, as well as the standalone metadata management systems.

Positioning the topic of metadata as the fulcrum of this thesis, a vital area of research within the broad research topic of metadata is the sub-topic of provenance. For this, in Section 2.3, we introduce the concept of provenance management in scientific research and specifically in WfMSs and SDRs. Furthermore, extending the topic of provenance for enabling provenance interoperability between WfMSs, in this chapter we introduce the topic of provenance interoperability in Section 2.4, and survey the available solutions with their limitations.

2.1 Scientific Data Repository Systems

In the broad research area of e-Science and cyberinfrastructure, the principle focus is **data** and questions surrounding it are: How to acquire it? How to store it? How to access it? and how to leverage it for scientific discovery and knowledge dissemination [90]. A vital aspect of an e-Science cyberinfrastructure is the realization of an SDR. The term SDR is synonymous to Institutional Repositories (IR), and in this thesis, we consider both SDR and IR to represent the same concept.

There exists a few definitions of IR, following are a couple of the prominent ones: Lynch. C.A. [101] defines IR as:

"In my view, a university-based institutional repository is a set of services that a university offers to the members of its community for the management and dissemination of digital materials created by the institution and its community members. It is most essentially an organizational commitment to the stewardship of these digital

materials, including long-term preservation where appropriate, as well as organization and access or distribution."

Ware. M. [164] defines IR as:

"An institutional repository (IR) is defined to be a web-based database (repository) of scholarly material which is institutionally defined (as opposed to a subject-based repository); cumulative and perpetual (a collection of record); open and interoperable (e.g. using OAI-compliant software); and thus collects, stores and disseminates (is part of the process of scholarly communication). In addition, most would include long-term preservation of digital materials as a key function of IRs"

Broadly speaking, the overarching area of research for which this thesis provides its contributions is the architecture of SDR. So the prime question is, what purpose does an SDR serve?. As stated by Haak et al. [104], *scientific data repositories enable systematic data stewardship practices to foster adequate data collection, curation, preservation, long term availability, dissemination and access.*

As previously perceived, the modern SDRs are no more just a system for storing and preserving data and metadata but offer numerous functionalities that are vital for handling the entire lifecycle of data and metadata. In general, SDRs are large infrastructures that provide researchers the functionalities to manage, share, access and archive their data [116]. Following the design principle of *separation of concerns*, an SDR is divided in task-specific modules, where each module provides the necessary functionality required for handling a specific stage of the research data lifecycle.

As most of the existing SDR are institute or research area specific, till date there exist only a few studies that try to collect the minimum functionalities an SDR should provide. Haak et al. [104] have conducted a study to analyze one hundred SDRs to determine the common attributes under which these SDRs were designed. In their study the following key components (features) of the SDRs were identified: *type of data, format of data, ingest and export, curation, preservation, and storage.* Another study conducted by Assante et al. [10] have analysed the solutions offered by generalist SDRs, wherein they define generalist SDR as *the SDRs that support any type of research data.* The common characteristics among these generalist SDRs are: *formatting, documenting, licensing, publication costs, validation, availability, discoverability and citation.* Similar research was done by Amorim et al. [7], wherein prominent open-source data repositories (DSpace, CKAN, FigShare, Zenodo) were compared based on following key-aspects: *architecture, metadata handling capabilities, interoperability, content dissemination and search features.*

From the different attributes each study has conducted for gauging the features of an SDR, it can be seen that many of these attributes represent the same feature or the functionality offered by the SDRs. For example, the category *format of data, type of data* considered by Assante et al. [10] and the characteristic *formatting* used by Haak et al. [104] are the same. Thus, the first step in understanding the overall area of research is to identify the functionalities offered by the existing SDRs. For this, we

aggregate the diverse characteristics of SDRs presented by various studies, and based on it, in the following we briefly describe these features.

- Data format: From the perspective of long-term storage and reusability of data, it is important for the research community to know the file format accepted by an SDR. SDRs that accept a fixed set of file-formats are a limiting factor for research communities because it puts additional effort on them for converting their data in SDR compatible format. Typically, for supporting a wide range of communities, SDRs should have no constraints on the data format they support.

- Metadata management: A basic feature of a SDR is its support for metadata that describes the stored datasets. For validating and reusing the datasets not only within the research community but also more widely within other research communities, it is a common practice to enrich the datasets with auxiliary contextual information that describes the dataset. Moreover, the information of how the data is obtained (data provenance) also needs to be annotated to the data for allowing data reproducibility.

- Data curation: Lord et al. [99] defines data curation as: *"The activity of managing and promoting the use of data from its point of creation, to ensure it is fit for contemporary purpose, and available for discovery and reuse."* SDRs not only just provide a data storage solution but also provide access to the high-performance computing environment that allows performing custom data analysis workflows. Data curation is either performed automatically with the deployment of WfMS or manually by domain experts, for example, enriching the data quality by adding domain-specific knowledge in the form of annotations.

- Interoperability and content dissemination: Exposing the data and metadata stored in the SDR is critical, as it enables its reuse. Data publishing is one method of making the data public. For effective access and exchange of data and metadata SDRs must support open and interoperable standards. For sharing and large scale metadata harvesting, the SDR must be OAI compliant, i,e., the SDR must support the *de facto* OAI-PMH protocol.

- Ingest and Access: SDRs are typically deployed in a distributed or centralized servers with the possibility for transferring the data from data acquisition system (DAQ), researcher's local machine, or dedicated data processing machines. The methods supported for ingesting and accessing the data may vary across SDRs. Generally, data transfer protocols, such as FTP, SMTP, HTTP, WebDAV, GridFTP are supported by SDRs.

- Search and discoverability: Making the ingested datasets discoverable is critical for reusability and publishing of the results. Both the dataset as well as the metadata describing the datasets should be searchable. As the SDRs store the data in file format, performing search on the contents within the file is

not possible as the contents are specific to a research community. Hence, the documentation (metadata) describing the data needs to be made available for searching and discovering the corresponding data.

Additionally, other non-technical features that contributed towards SDR are: (1) Licenses that describe the data policies, including the access rights are necessary for sharing and reusing of data. (2) Data publication and maintenance cost are associated with storing data in the repository. This cost has to be undertaken by the data provider or the data owner. (3) Data citation is a trivial feature of an SDR, wherein the SDR offers the possibility to reference the stored data. The data concerning an experiment whose results are described in a publication are typically referenced by the PID associated with data.

From the above mentioned key components of a SDRs, the main goal this thesis is to focus on research in metadata management in SDR. The availability of metadata enables data discovery, sharing, reuse, interpretation, and access. However, only the metadata is not sufficient for enabling scientific data reproducibility. Data reproducibility of a particular data analysis workflow has become a critical aspect for validating the claims of scientific research. Moreover, from 2008, SIGMOD has introduced the "experiment repeatability requirement to help published papers achieve an impact and stand as reliable reference-able work for future research"[1]. For this, the thesis focuses on two more fine-grained research areas that are provenance handling in WfMS and provenance interoperability among heterogeneous WfMSs.

As the contributions presented by thesis are foremostly adopted by the KIT Data Manager, in the next section, we briefly describe the KIT Data Manager. The KIT Data Manager is a SDR framework that has already been adopted and extended for handling the data and metadata of nanoscopy [123], angioscopy [126] and ecodicology [31] research communities.

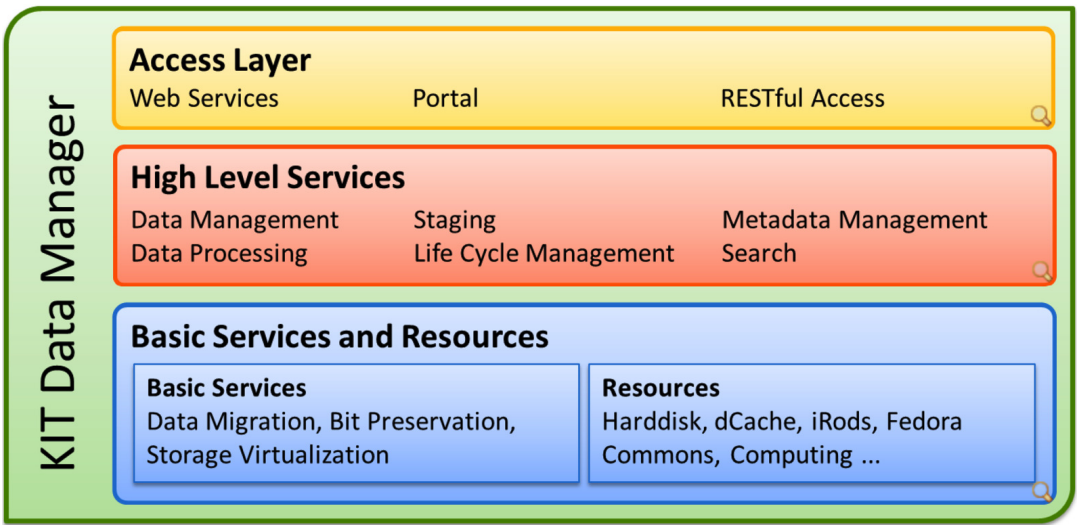


Figure 2.1: Overview architecture of KIT Data Manager [85]

[1] *http : //www.sigmod*08.*org/sigmod*$_r$*esearch.shtml*

2.1.1 KIT Data Manager

The KIT Data Manager is an architecture for building repositories for a broad spectrum of scientific communities in handling their research data [85, 123]. The KIT Data Manager provides a generic service stack that can be customized as per the requirements of the research community. The main components of the KIT Data Manager architecture are illustrated in Figure 2.1, with a brief description for each:

Basic Service and Resource. Principally, the basic services are the low-level services that are responsible for handling the data and the metadata on the file-storage system. This component exposes services for migrating the data between different storage resources. Moreover, data storage on existing repository frameworks, such as iRods and Fedora is supported through adapter services.

High Level Services. The high-level services offer the end-user the possibility to control the lifecycle of the data within the repository. Services for ingesting and accessing the data from heterogeneous systems, typically, the data acquisition systems (DAQ) of the research community are provided by this component. These services support a wide range of protocols, for example, GridFTP, FTP, WebDAV, and Server Message Block (SMB). For providing a generic data ingest and download client that is independent of any *data-format*, we extended the KIT Data Manager with the Generic Client Service API, and the results have been published in the *IEEE BigDataService* conference [123]. Data processing and staging services allow researchers to define custom data processing workflows within the KIT Data Manager framework.

Access Layer. The access layer offers a RESTful API for accessing the high-level services. For integrating user interfaces with the KIT Data Manager, these services provide the appropriate end-points. For example, user management service for creating new users and groups in the KIT Data Manager are provided by this layer, scheduling the ingest and download of datasets can be triggered via the access layer services.

2.1.2 Limitations of KIT Data Manager

From the above description of the features available in KIT Data Manager, it is clear that the KIT Data Manager is not capable of handling the entire research data lifecycle. Regarding metadata, only the minimalist metadata based on the Core Scientific Metadata Model (CSMD) is supported by the KIT Data Manager. Currently, the KIT Data Manager clearly lacks a comprehensive metadata management functionality with an OAI-compliant metadata harvesting mechanism. Regarding executing complex scientific workflows, the KIT Data Manager does not provide integration with a WfMS, with no functionality to trace the provenance. As there is no provenance captured by the KIT Data Manager, it is not possible to publish scientific results with their corresponding provenance trace, workflows cannot be repeated, and provenance traces cannot be queried and analyzed for improving the research.

2.2 Metadata Management

Extensive research efforts have been done and are underway in metadata research and software frameworks for handling metadata in scientific research. The wide-spread applications of metadata have its roots in the research area of cultural-heritage. Libraries, archives, and museums have long been using metadata for creating, tracing and sharing the information about books and artifacts [48, 143, 182]. Previously, the metadata was handled using hand-written catalogs or inventory books. However, with the digitization of the books and the artifacts, it was necessary to establish schemas that could maintain the information that was previously written in the catalogs. Subsequently, with the wide-scale acceptance of these schemas by multiple libraries, museums, and archives, metadata standards were established. For example, in the area of digital libraries, the MAchine Readable Cataloguing schema (MARC) [176] evolved to be an international standard that is used for documenting books and journals. MARC was sufficient for modeling metadata for digital libraries. However, these standards were specific to the field of cultural-heritage, and could not be adopted for other research areas. Hence, during the last couple of decades, multiple research communities began designing their own metadata schemas for handling the metadata concerning to their research. These schemas were promoted for adoption from a wider audience, which led to the standardization of these schemas. Currently, there exist more than two hundred metadata standards in research discipline of biology, earth science, physical science, social science and humanities and general research data [30, 93, 128]. Moreover, as metadata was treated as a second-class citizen to data, and the metadata management frameworks built in the context of SDRs were capable of handling only a single or few specific metadata standards. This prevented the generic applicability and reusability of these metadata frameworks by a wider community.

The first generation of digital repositories adopted the Qualified Dublin Core (QDC) metadata schema [87]. However, with the need for describing complex multidimensional digital objects, more complex metadata standards were created, for example, the METS standard that allows modeling of multiple ad hoc metadata standards within a single METS standard [27]. In the next sections, we provide an in-depth survey of existing metadata management systems in SDRs, standalone metadata management system, metadata management systems in Grid infrastructures, and commercial metadata management systems.

2.2.1 Metadata management systems in SDRs

In this section, we consider the metadata management capabilities of the following generalist SDRs, DSpace [149], Archivematica [162], Eprints, Hydra [11], EUDAT project [95], Comprehensive Knowledge Archive Network (CKAN) [170], and Fedora. The term generalist SDRs is defined by Assante et al. [10] as:

"repositories that make no assumptions nor special arrangements for community- or data-type-specific aspects".

The community-specific SDRs are not considered in this thesis due to the following two reasons: (1) The community-specific SDRs are designed for handling only specific metadata standards that are necessary for a given research discipline, whereas the core focus of this thesis is to design an entirely generic metadata management system. (2) Currently, there are more than hundred community-specific SDRs, and already there exists in-depth surveys elaborating the metadata management capabilities of these SDRs exist [13, 102, 104].

DSpace is an open source digital repository system for preserving digital data (image, video and text datasets). The primary metadata standard supported by DSpace is the Dublin Core (DC) standard. DSpace also supports simple non-hierarchical metadata namespaces, with the extension to support MARC and MODS metadata standards using additional external tools. For allowing full-text search over metadata, DSpace provides the possibility to select the metadata fields that are to be indexed manually. Regarding support for existing standards, DSpace is OAI-PMH compliant about allowing metadata harvesting and supports SWORD protocol [3] for repository interoperability. As a metadata storage system, DSpace supports either PostgreSQL or Oracle database. Fedora Commons is another digital repository designed especially for the library community. By default, Fedora supports the DC standard, and through Fedora-specific extensions, the METS standard is supported. In 2009, Fedora Commons and DSpace were merged to form DuraSpace. Many institutional repositories are based on Fedora Commons framework and DSpace, for example, Hydra, DRYAD, and CLARIN [92].

Archivematica is an open-source digital preservation system based on the OAIS functional model [94]. It supports METS metadata standard with the embedding of DC standard for capturing the descriptive metadata and the PREMIS standard for handling the preservation metadata. For enabling full-text search over metadata, Archivematica uses ElasticSearch.

EPrints was designed specifically for archiving research papers, theses, and teaching material, however, it is also possible to store any data. Similar to DSpace and Archivematica, EPrints provides support for DC standard, with the possibility to export the metadata in METS standard. Regarding compliance with standards, EPrints is OAI-PMH complaint and support SWORD protocol for repository interoperability. Large volume data ingest through batch importing is challenging and requires PERL scripting knowledge. For allowing full-text search over the metadata, extra effort is required for integrating Xapian[2] engine with EPrints. However, the full-text search feature is only available for few data formats like PDF, Word, and HTML.

The European Data Infrastructure (EUDAT) is a Pan-European project that aims at providing common data services and a collaborative research environment for multiple research communities. Of the various common data services, the metadata related services are the B2SHARE and B2FIND. The B2SHARE service is for ingesting, storing, preserving and sharing the data. The metadata fields have to be filled in manually

[2] *https://xapian.org/*

during the ingest process. Moreover, for enabling full-text search over metadata, the B2FIND uses SOLR indexing, wherein the metadata is harvested using the OAI-PMH protocol from various metadata provider and indexed in SOLR [69].

2.2.2 Standalone metadata management systems

In this section, we survey the standalone metadata management systems that aim at providing a generic metadata management solution.

Metacat is an open source metadata catalog and data repository, which aims at catering the metadata needs of the National Center for Ecological Analysis and Synthesis (NCEAS) [19]. Metacat provides a schema-independent data storage for metadata ingested in XML format. The Metacat framework uses a hybrid storage approach wherein it extracts, models, and stores the metadata from the XML in an RDBMS schema that conforms to the XML format. Metacat provides a set of SQL queries that can be executed over this schema for retrieving the metadata.

The XMC Cat metadata catalog of the LEAD cyberinfrastructure follows a hybrid XML/relational approach that stores the XML metadata as a Character Large Object (CLOB) and further shreds the XML using inlining [142] and stores it in a relational database schema to enable execution of complex queries [158].

Yang et al. [177] presents DIstributed MEtadata Server (DIMES) that manages XML-based scientific metadata in different formats, with support for sophisticated search queries. The DIMES is based on the concept of representing the XML document containing the metadata in its natural Document Object Model (DOM) tree structure. For querying purposes, the XML4J package is integrated into DIMES.

2.2.3 Metadata management in Grid infrastructures

In the research area of distributed computing, there exist multiple systems for handling metadata in the Grid environment. In the following, we describe these systems with their features for handling scientific metadata.

The Storage Resource Broker (SRB) of the San Diego Supercomputer Center is a middleware that provides an API for accessing heterogeneous distributed storage resources [16]. For supporting attribute-based access of the data, the SRB employs a Metadata Catalog Service (MCAT) [147]. The MCAT metadata schema for modeling descriptive and system related metadata is similar to the Dublin Core metadata standard. This schema is modeled in a relational database (DB2) and a set of APIs are provided for querying and updating the metadata.

The Metadata Catalog Service (MCS) is a Grid-based metadata service that is designed for handling the metadata generated in the Grid environment [45]. Principally, MCS provides functionalities for storing, accessing, and querying descriptive metadata based on user defined attributes. The MCS schema is an extension of the MCAT schema of the SRB middleware, and is implemented in a relational database (MySQL). For systematically organizing the metadata, the MCS metadata schema is

divided in the following logical categories: *logical file metadata, collection metadata, view metadata, authorization, user, audit, provenance metadata, and user-defined and annotation metadata*. Out of these logical categories, the *user-defined* and *annotation* metadata allows the MCS schema to be extended beyond its default attributes. As a generic solution for handling community-specific metadata, MCS provides two tables, one that contains the common attributes and an extension table with a set of predefined attribute types (Integer, String, Float, Date, Time, DateTime) for storing additional community-specific attributes. The extension table contains three columns (object id, attribute name, and attribute value). Each entry in this table has a reference to the object id in the common table with an attributed name and value (basically a key-value pair). The annotation service of MCS allows users to create a key-value pair of the string type that can be associated with an object of type *logical file, collection or view*.

The Science Object Linking and Embedding (SOLE) tool is specifically designed for linking research papers with science objects for making research data reproducible [118]. In SOLE, a science object can be language objects (source code), annotated PDFs and datasets, web-services, or virtual images. For retrieving the data analysis pipelines, SOLE offers users to link the results presented in a research paper with the workflows that were used to derive them. Metadata in the form of tagged annotations from PDFs are automatically extracted and stored in the SOLE database, whereas datasets can be manually tagged with annotations for imparting additional description.

2.2.4 Commercial metadata management systems

With the wide-spread realization of the importance of metadata, a few commercial solutions are available that offer comprehensive metadata management capabilities. In the following, we briefly describe these solutions.

Stardog[3] is an RDF database for handling enterprise data using an enterprise knowledge graph. It provides semantic tools based on OWL 2 ontologies, with SPARQL support. For allowing full-text search, the data is indexed in Apache Lucene [108]. For querying legacy systems through SPARQL, Stardog provides a Virtual Graph that allows mapping between the Stardog graph and external data sources. For enabling data quality control, Stardog uses Integrity Constraint Validation (ICV) that allows an explicit definition of data rules (constraints). Moreover, Stardog supports handling of database revision history using the PROV model, and domain-specific controlled-vocabularies can be integrated using the SKOS specification.

PoolParty is another commercial semantic technology platform that supports an enterprise in organizing enterprise knowledge with data analytics features [137]. Similar to the Stardog system, the PoolParty platform is natively designed on an RDF

[3] http://www.stardog.com/

database and primarily offers the following features: (a) The Thesaurus Server provides the typical CRUD operations for handling domain-specific taxonomies and thesauri based on SKOS, wherein the SKOS taxonomies can be enriched with ontologies (for example, the Friend of a Friend (FOAF)[4] ontology). (b) Users can create custom schemas from the existing ontologies using the ontology and schema editor service. (c) Metadata vocabulary quality can be measured using the qSKOS specification. (d) The Graph Search Server collects data from the various PoolParty services and transforms it into RDF, which is indexed in either Apache SOLR or Elastic Search for allowing full-text search. (e) The Extractor service analyses documents and texts, with the metadata schemas that are mapped to predefined SKOS thesauruses, and automatically extracts meaningful phrases, named entities, or other relevant entities.

2.2.5 Limitations of the existing metadata management systems

In this section, we summarize the limitations of the both the metadata management systems in SDRs and the standalone metadata management systems. In the case of the metadata handling systems of SDRs, it is observed that systems are limited in handling only basic non-hierarchical metadata standard like DC, with some repositories extending their metadata capabilities in exporting metadata in few other metadata standards, especially in METS. As the architectural design of these repositories is based on a SQL database, a major limitation is the flexibility of these repositories in handling community-specific metadata standards; because for handling any new metadata standard, the database schema and the repository framework needs to be updated, which is a time-consuming and labor-intensive task.

In the case of the standalone metadata management systems, the Metacat and the XMC Cat metadata catalog attempt to provide a generic metadata solution, but have the following limitations: (1) Both of these systems do not support handling of provenance in an accepted provenance model like PROV, OPM, or ProvONE. (2) As the metadata, irrespective of a research community is stored in the same table; it is not possible to segregate the community relevant metadata. (3) By design, these solutions either utilize RDBMS, which prevents horizontal scalability with increasing loads, or native file system that is I/O intensive.

In the case of DIMES metadata management framework, the long-term sustainability of XML4J is not guaranteed because the project was migrated to the Xerces project. Hence, the compatibility of the existing query engine in DIMES needs to be verified. Moreover, as XML4J and Xerces are both XML parsers that perform read and write operation on XML files located on the file-system, it raises major doubts about their query performance and scalability. Finally, none of these systems support handling of dynamic metadata in the form of annotations, especially with support to the standard WADM.

[4] http://xmlns.com/foaf/spec/

2.3 Provenance in WfMSs

Provenance in e-Science is a highly vast topic that covers various areas of research such as provenance in WfMSs, standalone provenance management systems and provenance in Service Oriented Architectures (SOA). In the context of scientific workflows, there are two aspects of provenance [33]: prospective provenance and retrospective provenance. **Prospective provenance** captures the workflow definition or the recipe describing the various processing steps that need to be followed to generate the required results. **Retrospective provenance** captures the actual runtime events that occur during the execution of a workflow. These events include details of the input, intermediate, and result data, the various processing tasks, and the execution environment.

In this section, first, we describe key characteristics for data provenance, namely, the applications of provenance, the techniques used to represent provenance, provenance storage systems, and provenance dissemination. Then, considering the focus of the thesis, which is the integration of a WfMS, especially for the Grid infrastructure, we describe the provenance management capabilities of the existing WfMSs and standalone provenance management system. Finally, a summary of the limitations of these existing provenance systems is highlighted.

Applications of provenance

Provenance has a wide-range of applications, for example, provenance allows reproducibility of scientific results, it enables researchers to validate the claims made by other researchers, and the quality of data can be estimated using provenance. In the following, we summarize the applications put forth by Goble. S. [65]:

- Substantiating data quality: By tracking the data transformations, provenance can be used to estimate the reliability and quality of the result data generated in an experiment. The granularity of the provenance that is captured helps determine the quality of data. Basic provenance details, such as the transformation applied and reference to the parent data, can help the user estimate the authenticity of the data, whereas, provenance containing semantic knowledge can enable automated evaluation of the data quality based on quality metrics. Hartig and Jun [75] provide a novel provenance model for evaluating specific quality criteria, such as timeliness and accuracy. The provenance model is designed for the Web data provenance and used in assessing the quality of the data.

- Tracking the audit trail: With provenance, the audit trail of data can be traced, i.e., the usage of resources, the movement of data between data storage units, and detecting errors generated during data transformation step [60]. The SPADE framework provides a cross-platform data provenance collection, filtration, storage and querying service [62]. This framework provides a cross-platform kernel that allows collection of provenance various operation systems (Windows and Linux/Mac OS X), with the support towards the OPM.

- Experiment repeatability: The systematically defined data derivation steps not only allow the automation of the experiment but also offer the possibility for others to repeat the same experiment. Quan et al. [119] present Provenance-To-Use (PTU) tool to minimize the computation time during repeatability testing. The PTU allows authors to assemble and their code, data, environment, and provenance into a single package that can be distributed easily. Their approach is specifically targeted for testing software programs that are submitted to conference proceedings or journals for verifying the results. Moreover, there exists a plethora of WfMSs that enable repeatability of *in silico* scientific experiments.

- Data reproducibility: With the availability of the data derivation steps (tools, algorithms, execution-order) and the run-time details (execution environment, parameters, and the conditions under which the experiment was performed), the reproducibility of the experiment is guaranteed. In the research area of life sciences, the Galaxy framework enables data analysis by allowing users to apply tools to datasets and ensure these analyses are reproducible [67]. For this, the metadata generated during the workflow is automatically tracked and can also be enriched with user information in the form of annotations.

- Data ownership: With provenance, the authorship (ownership) and copyright of the data can be established. As data migrates, so does its provenance. Thus, the entire chronology of the ownership of the data can be traced. With the availability of the provenance (ownership history), the genuineness on the data can be established. For example, in the area of arts, archaeology, paleontology, archives, and manuscript the ownership plays a critical role in determining the legitimacy of an artifact. Kairos is a framework for providing secure provenance data by integrating digital signatures and the Time-Stamping Protocol services into grid computing environment [88]. With Kairos, forging the authorship is prevented by maintaining the user's digital signature.

Provenance Representation

The technique used for the representation of provenance determines the granularity of details that can be recorded and the extent to which it can be used (queried). The two major approaches that are used to represent the provenance information are annotation or inversion [145]. In the annotation approach, which is an eager provenance collection technique the derivation history of data, with the description of source data and the various processes involved, is collected as annotations [20]. In the annotation approach, the provenance is pre-computed and available as metadata for querying and analysis. In the case of inversion approach, the derivation that generated the output data is inverted to trace the input data. User-defined function, explicit function, queries are used to automatically invert the derivations [38, 169, 175].

With the availability of the OPM, PROV, P-PLAN and the ProvONE provenance model, it now possible to have a common model for representing provenance across various research disciplines. As these provenance model can be serialized in XML,

it is beneficial for use in SOA architectures where XML is the primary format for message exchange. Moreover, for capturing the semantic details within the provenance, these models support RDF representation, which is useful especially where the domain-specific ontologies are already defined in RDF and OWL. With the enrichment of provenance with ontologies, concept and relations can be precisely defined for enabling enhanced use of the provenance. Various research disciplines, as well as workflow engines, have enriched their provenance models with ontologies [51, 61, 135].

Provenance Storage

The underlying storage system used for persisting provenance determines the granularity of provenance that can be modeled and the expressiveness of the queries that will allow retrieval of complex provenance details. In the simplest case, provenance can be embedded within the data it describes and stored on the same data storage, or tightly coupled to the data it describes. On the one hand, this allows easy handling of provenance integrity, but on the contrary, searching, querying, and publishing the provenance is difficult. RDBMSs are the most widely adopted storage systems used for storing provenance [15, 57, 89, 180]. However, with the obvious benefits of graph databases for efficient modeling and querying of provenance graphs, few of the new systems have adopted graph database [63, 141, 174].

Provenance dissemination

In order to share and use provenance, a system should provide various means to access it. The existing provenance systems support SQL queries, APIs, or graph-visualization user interfaces for accessing and reusing the provenance. Additionally, some systems allow enrichment of provenance with semantic information for enabling defining complex queries capable of retrieving nested structures [8].

2.3.1 Provenance handling in WfMSs

Provenance has been extensively studied in several different areas of both WfMSs [23, 42, 145] and in databases [26, 160]. In this section we present the provenance management capabilities of the existing WfMSs, and describe them based on the key-characteristics stated above.

Vistrails is a Python-based WfMS that provides data process management support for exploratory computational tasks [57]. The workflows are defined in a proprietary Vistrails specification, and the provenance traces are modeled in a proprietary schema and stored in an RDBMS. Vistrails is the only WfMS that supports capturing the provenance of a workflow evolution. That is, Vistrails allows the capturing and storing of both the prospective provenance in XML and a Vistrails specific relational database schema. For querying the stored provenance, Vistrail provides a proprietary query language.

Taverna is a graphical workflow creating and execution environment [172]. Taverna is used by a wide-range of communities, for example, bioinformatics [152, 153], cheminformatics [161], astronomy [82], social science, music [107], digital preservation are some of the communities using Taverna. Taverna is shipped with the SCUFL workflow specification for describing the workflows. The Taverna engine for executing the workflow interprets the SCUFL XML containing the prospective provenance. The retrospective provenance traces are captured and stored in an internal database, and via the Taverna-PROV plugin, only the retrospective provenance traces are exported as PROV-O RDF graph.

Kepler is a WfMS based on the Ptolemy II engine [100]. Kepler provides its own MoML specification for defining the workflows (prospective provenance) [96]. For collecting the retrospective provenance, Kepler provides the Provenance Recorder (PR) that consists of several event listeners interfaces. These interfaces are based on different "concerns" for which the corresponding event listener's object is invoked. Kepler stores in its internal SQL database with a pre-defined provenance schema. For querying the stored provenance, an API is exposed by the Kepler WfMS.

Pegasus WfMS provides a bridge between the scientific domain and the Grid execution environment [47]. Pegasus automatically maps high-level workflow description onto distributed resources. The Pegasus framework automatically locates the input data and computational resources that are required for the execution of a workflow. The provenance in Pegasus is automatically captured by the Virtual Data System (VDS) when the jobs are launched using Kickstarter wrapper, and the prospective provenance is modeled in OWL. The retrospective provenance is stored in a relational database and can be queried or visualized using pegasus-statistics, pegasus-plots, or directly using SQL, and the prospective provenance can be queried using SPARQL.

REDUX is a closed-source WfMS that is based on the Windows Workflow Foundation (WinWF), and it extends the WinWF to capture the provenance at different levels of abstraction [15]. REDUX follows a multi-layered approach for incrementally capturing the provenance. The first layer captures the abstract description of the workflow, the second layer captures the specific services and data sets that are bound to the abstract description, the third layer captures the runtime information, i.e., the input data and the supplied parameters to each activity, and the fourth layer captures the workflow administrative information such as start and end time of the workflow. REDUX captures both prospective as well the retrospective provenance in an SQL database but does not capture the workflow evolution.

Karma is BPEL-based WfMSs that was specifically developed for supporting dynamic workflows for weather forecasting simulations [146]. The Karma provenance service is the focal web-service that subscribes to the notifications of the WS-Messenger service. For exchanging the provenance among the different Karma component, Karma provides XML schema. Finally, for persisting the XML containing the retrospective provenance information, it is decomposed and stored in a relational schema, thus allowing querying using SQL.

Sedna framework is an attempt to simplify the creation of workflows by providing a visual language and a modeling plugin tool-box that extends the existing Eclipse IDE plugin environment [165]. Using Sedna, researchers can easily define complex workflows in BPEL and deploy them on the ActiveBPEL, a BPEL-based workflow engine. However, in terms of handling the provenance, neither Sedna nor the ActiveBPEL workflow engine provides the functionality of capturing the provenance during the execution of the workflow.

2.3.2 Provenance in Grid workflow execution environment

In this section, we describe the Grid workflow execution environments that are based on BPEL workflow engine.

UNICORE is a Gird middleware that supports the orchestration of stateful Web Service Resource Framework (WSRF) [156]. For enabling handling of provenance in the normative PROV model, UNICORE is extended with the UniProv functionality that is responsible for tracking both the prospective and retrospective provenance [63]. They claim to support the ProvONE for modeling the workflow templates from an existing repository of PROV templates, but this is achieved is not explained in their paper. For storing and querying the provenance, the Neo4j database is integrated with UniProv.

Rajbhandari et al. [132] proposes an architecture for composing Grid services using BPEL4WS, wherein the Grid service clients are wrapped web services called as Proxy Web services. The workflows are deployed on Collaxa BPEL orchestration server or BPWS4J engine. The Collaxa is a closed-source WfMS from Oracle that provides no details about its ability to capture the workflow provenance traces. On the other hand, the BPWS4J is enabled with the Provenance Collection Service (PCS) and Provenance Query Service (PQS). The provenance is modeled using a pre-defined provenance RDF schema. Whether their pre-defined provenance schema is compatible with PROV, OPM or ProvONE is not explicitly stated by the authors.

Emmerich et al. [54] describe their experience in using BPEL for orchestration of grid services using the ActiveBPEL workflow engine. For testing the integration of ActiveBPEL with Grid service, they use a workflow from theoretical chemistry - *computation prediction of organic crystal structure from the chemical diagram.* They have stated that using a BPEL-based workflow engine is suitable for executing distributed grid web services but have nowhere mentioned how their framework captures the provenance.

Aleksander S. [148] describe the integration of an arbitrary BPEL engine with the Open Gird Service Infrastructure (OGSI) and WSRF. In their paper, they explain that BPEL is extensible for OGSI services that are based on web service standards. For this, they propose an extension to the BPEL engine in tracking the expiration time of the Grid Service Reference (GSR) and using the Grid Service Handle (GSH) for retrieving the active and valid GSRs. The integrating of WSRF with BPEL engine is straight forward because WSRF uses the WS-Addressing specification that is also

used in BPEL. Thus, there is no longer the need to map between GSHs and GSRs service locator and WS-Addressing Endpoint References. The tracking of provenance in the OGSI or in the BPEL-engine is not addressed by their system. One could consider the events from the WSRF as a potential source of provenance but this is not mentioned in their paper.

2.3.3 Limitations of WfMSs in handling provenance

Considering the provenance handling capabilities of the above stated WfMS, one distinct limitation is none of these systems support capturing of both prospective and retrospective provenance in a generic and interoperable provenance models such as P-PLAN or ProvONE. In the following, we summarize the other limitations of these systems. (1) All of these systems define their proprietary provenance schema (mostly SQL schemas) for storing and querying the provenance. Hence, for sharing the provenance between other systems, additional plugins or software extensions have to be developed for translating the provenance in PROV or OPM. (2) Except for Vistrails none of the WfMSs provide the possibility for tracing workflow evolution. However, Vistrails uses its custom data model for tracking the workflow evolution. Hence, sharing a workflow evolution trace is a major limitation in Vistrails. (3) In the case of BPEL-based WfMSs, it is clear that except UNICORE and Karma the other systems, especially the BPEL Grid-based systems do not provide any support for provenance handling. Moreover, the workflow enactment engine that these systems utilize (ActiveBPEL, Apache ODE, or BPWS4J) do not provide any mechanism for capturing the provenance. The maximum what these engines offer are the run-time events during the execution of the workflow. In the case of the UNICORE Grid middleware; recently they have started supporting capturing of provenance in PROV and storing it in Neo4j, a graph database. However, it is not clear which provenance analysis queries are supported by UniProv. Moreover, the PROV ontology is mapped to the RDF namespace and is a recommendation from W3C. Hence, sticking to the standards, an RDF store with the SPARQL querying capabilities are better suited for modeling, storing and querying the PROV graphs. Karma support provenance handling by capturing and normalizing the run-time events from the WS-Messenger and WS-Eventing publishing interface implemented by the enactment engine. However, Karma does not directly support any of the existing provenance models (OPM, PROV or ProvONE) but claim that their model is compatible with the OPM [28].

Focusing on provenance management in BPEL-based WfMS, in Chapter 5, we present the architecture for integrating a BPEL-based WfMS with the Prov2ONE algorithm that automatically translates the prospective provenance from BPEL specification to ProvONE.

2.4 Provenance Interoperability

The successful handling of provenance from BPEL-based workflows led us to extend our research in the field of provenance interoperability. As stated in the previous section, most of the existing WfMSs support handling of provenance, but each adopts its own data and storage model [58]. These models range from semantic web languages like variants of RDF and OWL, XML dialects, to relational databases. For accessing the provenance, the user is provided either with provenance visualization interface or needs to write queries using the query language supported by the WfMS. Typically, the query language is determined by the data storage system adopted by the WfMS. For example, few systems support querying using SPARQL [68, 89, 179], while others use SQL for querying the provenance [15, 180]. Hence, for cumulatively analyzing the provenance traces from such diverse systems, it is necessary to integrate the provenance from these systems and their respective querying interfaces.

Additionally, the importance of sharing provenance is observed in the collaborative research environment, where data and metadata (provenance) published by scientific workflows of one research group needs to be used by other groups [4, 6, 111, 157]. In collaborative researcher, complex workflows are divided among multiple research groups and based on the requirements of the research groups and the features provided by the WfMSs; each research group may adopt a different WfMSs. Adopting a different WfMS is not the issue, but the problem arises when these groups plan to share their results and the associated provenance among other research groups. As explained above, this is because each WfMS has its workflow language and data model for storing the provenance, and exchanging the provenance among the WfMSs is not possible. Thus, a major limitation in the research area of provenance interoperability is the ability to share and analyze provenance traces from heterogeneous WfMSs.

In this section, first, we present the existing systems and frameworks that attempt to enable provenance interoperability from heterogeneous provenance sources (Section 2.4.1), followed by their limitations in Section 2.4.2. The key characteristics presented in Section 2.3 are also applicable for these systems, and we summarize each of the existing provenance interoperability frameworks based on these key characteristics.

2.4.1 Provenance interoperability frameworks

There are several approaches described in the literature for enabling provenance interoperability among heterogeneous sources. In the following, we present a brief description of these approaches.

Ellqvist et al. [53] describes a mediator architecture for integrating the provenance from the Vistrails, Taverna, and PASOA [72] WfMSs. A core component of this mediator architecture is the Scientific Workflow Provenance Data Model (SWPDM), which acts a global schema capable of capturing both the prospective as well as the retrospective provenance. For validating their concept, they describe the translation of provenance from Vistrails, Taverna, and PASOA to SWPDM. This translation is

based on the mapping between the respective WfMS provenance model and SWPDM and is implemented in each of WfMS specific wrapper. For analyzing and querying the provenance traces over the global SWPDM, a wrapper specific query API is provided by the mediator architecture.

The Common Provenance Framework proposed by Braun et al. [24] aims at providing query interoperability between the provenance collected from the PASS and the PLUS provenance system. The Common Provenance Framework is built on OPM, with few extensions for enabling sharing, querying and visualization of the provenance traces. The visualization and querying of the provenance are achieved through the PLUS system for the provenance collected by the PASS system.

Ding et al. [51] propose a Semantic Web-based approach, also known as Linked Provenance Data, for addressing the challenges from the Third Provenance Challenge [144] and the research efforts in the field of Inference Web project [109]. They argue that for enabling provenance reuse, both the generic provenance and domain specific data. For this, they present the PC3OPM ontology for rich and interoperable domain-specific provenance descriptions. This rich provenance is imported to an RDF store for allowing querying and enriching of the provenance using the SPARQL Inferencing Notation [91].

For addressing the issue of sharing workflows in the collaborative research environment, where results from a workflow execution are used as input by other workflows, Missier et al. [111] presents an approach based on the common abstract model. The common abstract model is an extension of OPM that is mapped to the local provenance traces from the Taverna and Kepler WfMSs. With the availability of provenance in the common provenance model, the data dependencies across multiple workflow runs, systems, and user groups can be traced. Similarly, Altintas et al. [6] presents a provenance interoperability approach based on a common data model that is compatible with the OPM for capturing implicit user collaborations. Their approach is specifically tailored for handling provenance interoperability in a collaborative research environment. To enable querying over this model, they also present an extension to Query Language for Provenance (QLP).

Oliveira et al. [115] introduces a provenance integration architecture that provides a mapping of provenance from e-Science WfMS and SciCumulus WfMS provenance to ProvONE. The provenance in e-Science is stored in relational tables in the PostgreSQL database, and in the case of SciCumulus, the provenance is modeled as a graph in the Neo4J database. These mappings are implemented as WfMS-specific cartridges, and the provenance collected by the WfMSs is translated to the ProvONE model. The provenance modeled ProvONE is serialized as Prolog facts with a set of Prolog queries for retrieving the provenance. This approach is similar to the mediator approach proposed by Ellqvist et al. [53], with the only difference being that instead of using a proprietary model like SWPDM, the ProvONE model is used.

For enabling workflow interoperability (prospective provenance) between multiple

WfMSs, the SHIWA project presents the *coarse-grained* and *fine-grained* workflow interoperability. For enabling coarse-grained workflow interoperability, multiple WfMSs are assembled under a common platform and treated as sub-workflows with a master workflow that orchestrates the execution of the workflow over these sub-workflows. For this, the meta-workflow is defined in a particular specification that interfaces to other sub-workflow systems. However, as this approach requires immense technical and infrastructure coordination, the SHIWA project also proposed a mechanism for fine-grained workflow interoperability through conversion of native workflow to a common representation. For this, the Interoperable Workflow Intermediate Representation (IWIR) was introduced. IWIR is a low-level workflow representation language that was designed to represent as many constructs from different workflow languages as possible.

2.4.2 Limitations of existing provenance interoperability frameworks

It is observed that many of the existing approaches for enabling provenance interoperability are inherently limited, this is because these approaches use OPM, which is a provenance model that can model only the retrospective provenance. On the other hand, few of the approaches have implemented their proprietary provenance model for modeling provenance from heterogeneous sources, and whether these proprietary models are compatible with OPM, PROV, P-PLAN or ProvONE is certainly not addressed in the literature. In the following, we summarize the limitations of these approaches:

- Few of the provenance interoperability frameworks are based on proprietary data models that are not compatible with the recommended and adopted provenance models like OPM or PROV. Thus, the long-term sustainability and adoption of these frameworks by wider-range of communities is a major challenge.
- The majority of these frameworks are not designed for capturing the prospective provenance, and neither do they support the widely-adopted P-PLAN or ProvONE provenance models. Thus, only supporting partial (retrospective) provenance interoperability.
- Instead of adopting the W3C recommended SPARQL for querying the provenance, the approaches presented by Altintas et al. [6] and Oliveira et al. [115] use QLP and Prolog queries respectively. The major limitation in their approach is the long-term sustainability of these non-standard querying languages.

The approach presented by Oliveira et al. [115] is comparable to our approach. However, a major limitation of their approach is that any changes to the provenance database schema of e-Science and SciCumulus will require a remapping between the provenance models and the re-implementation of the WfMS-specific cartridges. Especially, the mapping between the retrospective provenance from the e-Science WfMS to

ProvONE. The e-Science WfMS stores the retrospective provenance in Neo4j, which is property graph database. In a property graph database, there is no constraint on the properties that can be added to the nodes or edges of a graph (i.e., any ad hoc key-value string pairs can be added as properties). However, adding new properties or modifying the already existing ones will break the existing mapping to ProvONE, thus, making the implementation of e-Science cartridge obsolete. In our approach, we automate the capturing of the exact workflow definition by mapping the entire workflow specification of BPEL, SCUFL, and MoML to the ProvONE prospective provenance, thus, avoiding any weak data-model mappings. Furthermore, in their approach, the ProvONE translated provenance is stored as Prolog facts, whereas considering the long-term sustainability of a solution, our approach supports the W3C standard RDF model, and the W3C recommended SPARQL query language.

2.5 Summary

In this chapter, we placed the thesis into its primary research context of metadata management for scientific data, in particular for SDRs. First, we presented the concept of SDR and the critical features that are necessary to build a comprehensive SDR. Out of these features, we observed that majority of the tasks in an SDR are governed by metadata. Then, we briefly introduced the features of KIT Data Manager, a generic SDR framework, wherein we identified that the existing metadata handling capabilities of the KIT Data Manager are limited. Similarly, we also observed the same limitations in the metadata management capabilities of the available SDRs, as well as the standalone metadata management systems. This led us to concretize the primary focus of this thesis, which is metadata management for SDRs.

In our research in the area of metadata, an important sub-topic that emerged was of provenance. For understanding the topic of provenance in-depth, we described the various applications of provenance with the available provenance representation, storage, and dissemination techniques. However, in the vast research area of provenance, we directed our efforts in the area of provenance management in WfMSs. Again as inital adopters of our contributions, we realized that the KIT Data Manager currently lacks an integration with a Grid infrastructures suitable WfMS. For this, first, we presented the state-of-the-art of provenance management capabilities of the existing WfMSs, followed by the provenance handling features of the Grid-based WfMSs, which are typically based on BPEL-based workflow engines. From the survey of these WfMSs, we realized that currently none of these systems support handling of workflow provenance. Thus, these limitations led us to establish the secondary focus of this thesis, which is provenance management for BPEL-based WfMS.

As our research was directed in the area of provenance handling in WfMS, we realized that in collaborative research, wherein multiple WfMSs are simultaneously used for large-scale and distributed research, it is necessary to share and analyze provenance traces from heterogeneous sources. For this, we extended our efforts in

provenance interoperability. For this, first, we presented the existing provenance interoperability systems with their limitations. From the literature it was clear that existing approaches either do not support complete provenance interoperability or if they do support, as in the approach presented by Oliveira et al. [115], they do not adopt the standard representation (RDF) and querying techniques (SPARQL). Thus, considering these limitations, our tertiary focus of thesis that is provenance interoperability among WfMSs was concretized.

Chapter 3

Scientific Data Repositories

In distributed e-Science infrastructures where data and metadata are generated and consumed by arbitrary data processing tasks, tools or systems, SDRs have become mandatory for efficiently handling the data and metadata lifecycle. The basic functionality of a scientific data repository is to ensure long-term preservation and access to data and metadata [104].

The goal of this chapter is to introduce the architecture design of a generic SDR and present our initial contributions on this research topic. Instead of developing and promoting institution-specific or discipline-specific repository systems, the focus is to design a simple, generic, and extensible SDR architecture that can be adapted for a wide-range of disciplines. The design of SDR architecture presented in this chapter is based on the principles of modular design. For this, we have divided the architecture into task-specific components that individually or collectively realize a specific functionality within SDR. The four critical components within the framework of an SDR are data transfer handling, metadata management, workflow with provenance, and provenance interoperability. However, as each component of this architecture is a research topic in itself, we have individually addressed these components in separate chapters. In this chapter, we present the data transfer component and the adoption of the SDR architecture for two research disciplines.

3.1 Introduction

The advent of novel data acquisition systems with the potential to produce and process enormous amounts of data has led to an exponential growth in the volume of data and metadata that is generated in *in silico* scientific experiments [40, 78]. This data and metadata have become an essential component of scholarly and scientific research. Recently, the major conferences and journals have enforced the mandatory submission of data and metadata along with the article, in order to accept it for publishing [12, 74, 83, 154] These steps were taken for making the entire data processing workflow transparent to the scientific community to enable reproducibility and verification of the published results.

On the one hand, making research data referenceable, accessible, reproducible, and publish worthy are major factors for gauging the quality and authenticity of the data.

However, on the contrary, to realize these factors, there is a need to design and build state-of-the-art data and metadata management systems that would not only assist researchers for publishing and referencing purposes but also in handling the complete data lifecycle.

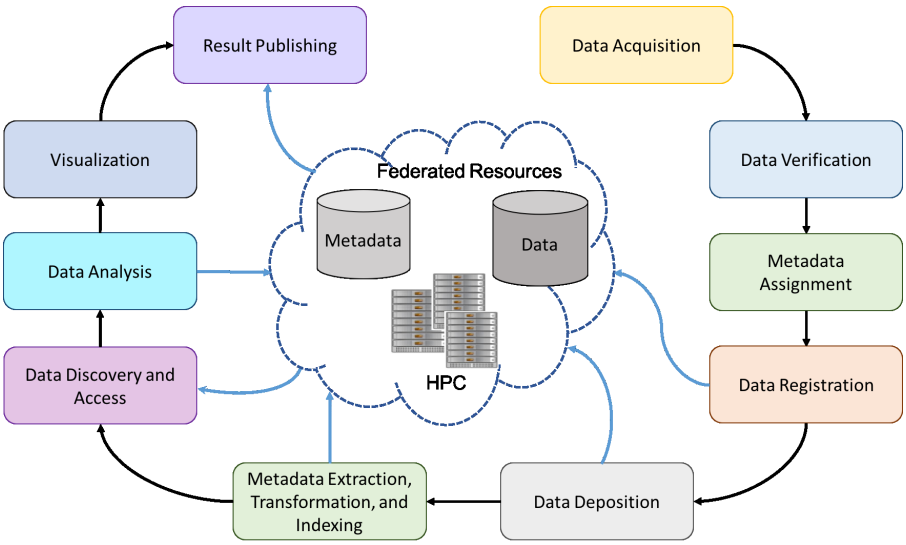


Figure 3.1: Abstract view of scientific data lifecycle

In Figure 3.1, we present an abstract view of a scientific data lifecycle described by Hey et al. [80] in his book *The fourth paradigm: data-intensive scientific discovery*. Regardless of the research discipline, the first step is to acquire the data from the discipline-specific DAQ systems such as sensors, microscopes, medical modalities, digitization infrastructures, simulations, etc. The acquired data is verified against the research process policies by automated methods or by the domain-experts. Once this data is accepted, the necessary metadata either through the DAQ software or via the researcher is tagged to data. The metadata usually includes descriptive information elaborating the acquisition hardware, software and parameters, researcher and organization details, and experiment setup. Until this step, data is stored on computing infrastructure connected to DAQ system. However, the DAQ computing infrastructure is not suitable for long-term data preservation, processing, and data-intensive analysis. This is because, the DAQ computing systems are built for a single purpose, which is to acquire data from the DAQ instrument's interface. The provisioning of tools for preservation, access, and processing of data is not the task of these systems. Hence, to prevent mismanagement and corruption of data, it needs to be registered and deposited in a storage unit that is configured to handle such kind of data and is under the administration of the organization's federated resources. After the deposition of data, remaining steps in a scientific data lifecycle are governed by the functionality provided by underlying data management system. Tasks such as extraction of metadata from data, transforming it into discipline-specific metadata standard and making it searchable through indexing are some of the necessary steps that have to be performed to make the data discoverable, shareable, and reusable. With access to

the High-Performance Computing (HPC) Cluster, data is analyzed to gain additional insights. These results are deposited in the data storage unit with prior registration for allowing data traceability and reproducibility. For human-interpretation of the results, data and metadata are visualized. Visualizing techniques are employed for domain-experts to gain more knowledge about the data and if required impart additional information on the data in the form of annotations. Regardless of the choice of visualization and annotation tool adopted by the research community, the system should provide the conversion of data and metadata in standard formats. Proprietary formats should be avoided as they often lead to solutions that are not interoperable. For example, heterogeneous metadata received from diverse sources needs to be systematically structured and exposed through standard metadata harvesting protocols like OAI-PMH. Finally, for knowledge dissemination, after extensive exploration and analysis, the results with its metadata describing the entire procedure needs to be published and made referenceable.

It is clear that for efficiently handling the data deluge, research institutes, funding agencies, data specialist, and researchers are impelled to find solutions for managing their research data with compliance to the policies stated by the research process. This has led to the adoption of SDR in a majority of the research institutes and universities. SDR should not only assists researchers in their routine activities but also provide highest-level of automation in order to hide the SDR's complexity and eliminate redundant manual tasks. An SDR should expose standard interfaces for seamlessly integrating it in existing research environment, and provide extension possibilities for adding new functionalities. Therefore, in the context of the architectural design of SDRs, we present the following contributions: (a) a generic and extensible architecture of an SDR that is based on KIT DM framework, (b) automated data and metadata agnostic data transfer client that can be integrated with arbitrary discipline-specific DAQ, (c) adoption of the complete SDR architecture for three research discipline, of which two are presented in this chapter.

The chapter is organized as follows. First, in Section 3.2, research efforts related to the design of generic SDRs are presented. Then, in Section 3.3, the complete architecture of the SDR is presented. In this section, we present the first contribution of this chapter. However, except for the GCS API, only a brief explanation of the other components is presented. The details of each component are treated as a separate research topic, and their realization is described in the succeeding chapters. In Section 3.4, we present the evaluation of the GCS API with an optimized implementation of the WebDAV protocol for nanoscopy datasets. In Section 3.5, we present the second contribution of this chapter that is the adoption of SDR architecture for the nanoscopy and angioscopy research data. Finally, we summarize the chapter in Section 3.6.

3.2 Related Work

The generic design of an SDR is a primary criterion for adoption by multiple research disciplines. In this section, we present existing generic SDRs that are developed independently of any particular research discipline.

The Fedora Commons Repository is a flexible, modular, and open-source repository platform that has native support for linked data. Most of the functionality in Fedora is available through REST APIs. Regarding supported data size and ingest protocol, Fedora enforces no limit on the size of data that can be ingested in the repository. Currently, HTTP REST, Network File System (NFS), and Secure Copy (SCP) are the three data transfer protocols that are supported by Fedora. The ingested data can be searched using SOLR indexing that offers full-text search over the metadata or using a standalone triplestore like Jena Fuseki for searching the linked data graphs.

Archivematica is web-based and standards compliant open source repository architecture that provides long-term access, authenticity, and reliable access to digital content. The architecture design of Archivematics is based on micro-service pattern, with compliance to the Open Archival Information System (OAIS) functional model. Regarding support towards metadata standards, metadata in DC, METS, and PREMIS standards can be handled by the framework. The metadata is indexed in ElasticSearch for allowing a full-text search.

Compared to other generic repositories, the DSpace repository shares the same vision as of the KIT DM. DSpace is a software package containing a set of tools that allow communities to build discipline specific repositories for preserving their digital content (scientific data) lifecycle within the repository. The data in DSpace is stored in an asset repository, while the metadata is stored in a dedicated metadata store. By default, the DC metadata standard is supported by DSpace, with extended support using add-on plugins for the MARC and MODS standards. The metadata is indexed in Apache SOLR for allowing full-text metadata search. The entire functionality of DSpace is exposed through REST APIs that are integrated into a web user interface. Dryad is a data repository built on the DSpace technology stack for storing data associated with a published article. In Dryad, after data ingest, a Digital Object Identifier (DOI) is assigned to data for allowing referencing and citation.

Samvera, formerly known as Hydra is an open-source digital asset management solution for libraries, museums, and archives. Samvera is built on the Fedora software stack and consist of four major components: (a) Fedora repository layer for persisting and managing digital content. (b) SOLR indexing for full-text search and access to preserved resources. (c) Blacklight, a Ruby on Rails plugin that provides faceted search over the Solr indexes. (d) Samvera gems that integrate the various building blocks of the repository to enable an extensible digital repository solution.

The first step for any research discipline before choosing an SDR is to evaluate the features provides by it. Currently available SDRs are primarily focused on providing only a data storage and access solution with support towards very few metadata

standards. These SDRs do not support advanced capabilities such as execution of complex data-processing workflows with provenance handling, and extensible static and dynamic metadata model support. Moreover, these SDRs are capable of handling only small to medium size datasets in the range of few GBs. However, research disciplines such as nanoscopy, angioscopy, and eCodicology are regularly producing data volumes in the range of a couple of hundred TBs and even few PBs. Thus, to address the limitations of these SDRs, we describe the generic architecture of an SDR that is based on KIT DM. The KIT DM provides the low-level services especially for data storage and deployment of workflows on HPC clusters.

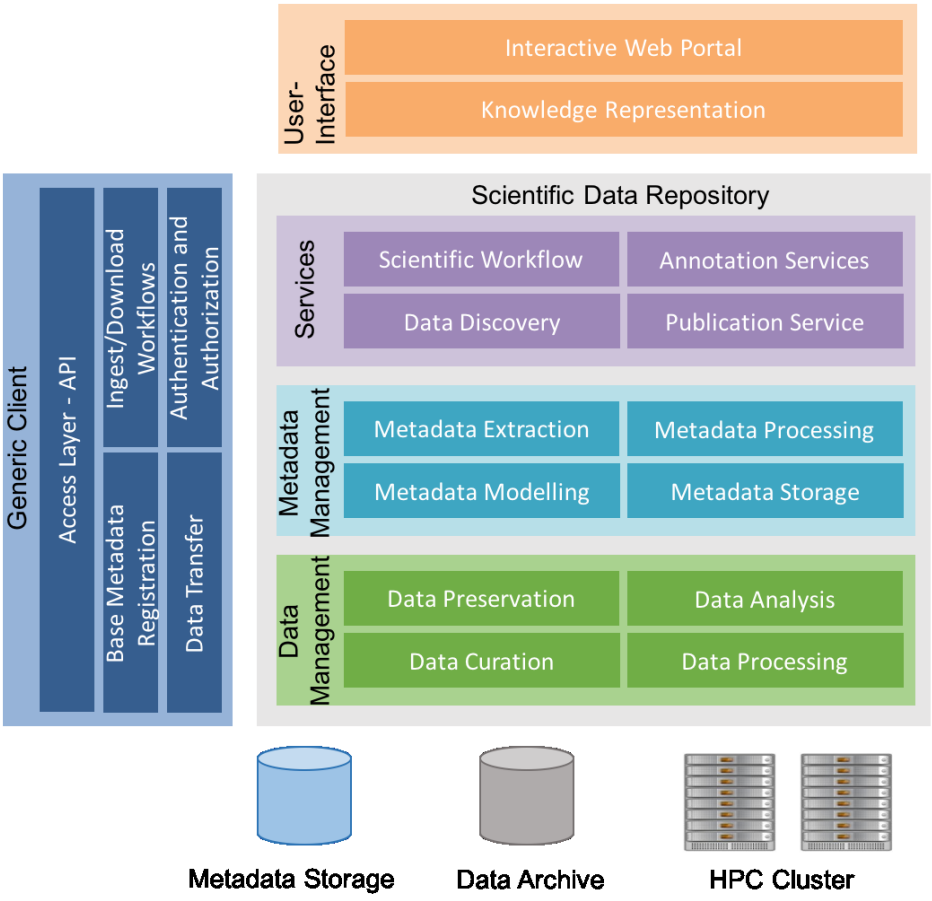


Figure 3.2: Layered architecture of scientific data repository [126]

3.3 Architecture of Scientific Data Repository

The scientific data repository is a server-side multi-layered architecture that is divided into task-specific components. An SDR needs to provide not only the data storage and retrieval functionality but also other functionalities that are required for efficiently handling the lifecycle of scientific data. In this section, we present an overview of the SDR architecture. The entire architecture is designed on principles of modular architecture design. In order to ensure long-term sustainability, each component in

this architecture is designed considering the standard tools, protocols, and metadata standards. The details of each component are presented in the succeeding chapters.

3.3.1 Scientific Data Repository

Services. The Services layer provides an interface to the external discipline specific tools and frameworks that have to be integrated with the SDR.

- Scientific Workflow: Through this module, an external WfMS can be integrated with the SDR. Provenance metadata which is a part of the scientific workflow is also handled through this module. In Chapter 5, Section 5.2.4 we present the detailed architecture explaining the integration of a BPEL-based WfMS with automated handling of provenance in comprehensive provenance model like ProvONE. This led to our first contribution in the research area of provenance management for WfMS. Furthermore, in Chapter 5, Section 5.3, we extend this module to a complete provenance interoperability platform that is capable of handling provenance from heterogeneous WfMS.
- Annotation Services: The annotation services are an extension of the underlying Metadata Management component. The focus of this component is to assist researchers in handling dynamic metadata in the form of annotations. Considering the interoperability of annotations with an adoption of a standard data model, this component is designed conforming to the Web Annotation Data Model (WADM) standard with storage based on RDF data model.
- Data Discovery: The data discovery service is an abstract wrapper service that primarily provides functionality for allowing full-text search over the stored metadata. The possibility to execute custom user-defined queries is also exposed through this service. For example, for querying and analyzing annotations and provenance stored in RDF database, a SPARQL endpoint is exposed.
- Publication Service: The publication service is for allowing sharing and citing the dataset stored in the SDR. For this, each dataset is assigned a unique PID, and the data is either open-access or close-access for specific groups of researchers depending on the policies defined by the funding and research institutes.

Metadata Management. The Metadata Management aims at providing an entirely automated metadata management solution for arbitrary metadata standards. This component is natively compliant with the OAI-PMH metadata harvesting protocol. The Metadata Management component is realized through the MetaStore framework, which is presented in Chapter 4. Here, we briefly explain the coarse functionality that is necessary for an SDR in handling metadata.

- Metadata Extraction: The metadata extraction module provides services specific to the format in which metadata is ingested in the SDR. Metadata can

either be embedded within data or submitted in a separate file. For metadata submitted through a separate file, this module offers a set of automated metadata extractors using the Apache TIKA library [106]. However, for domain-specific data formats, it is necessary to implement the metadata extractors. For example, the HDF5 metadata extractor service is available for extracting the nanoscopy descriptive metadata.

- Metadata Modelling: The metadata modeling module ensures that the extracted metadata conforms to a given metadata schema registered by the research community. This module performs registration of metadata schema and automated quality check for completeness and schema conformance. The metadata is modeled in the registered metadata schema (standard) and submitted to the metadata storage module for persistence.

- Metadata Processing: The preparation of metadata for harvesting purposes and the categorization of metadata as per their applications is done in this module. The different queries for analyzing metadata are exposed through this module. For example, the SPARQL queries for provenance graph matching queries and the annotations analytical queries are made available through this module.

- Metadata Storage: The SDR architecture is designed to be database technology agnostic. For this, in the metadata storage module, we provide database-specific adapters. For example, for storing the static metadata, a NoSQL database ArangoDB adapter is implemented. For storing the provenance graphs modeled in RDF data model, an Apache Jena adapter is provided.

Data Management. The data management is a low-level component that directly interacts with the interfaces exposed by the KIT DM. In principle, this component can be seen as a bridge-interface between the KIT DM and the community-specific realization of an SDR. Configurations for the HPC Cluster environment, service deployment registry for the data processing tasks, and access to large scale data storage are made available via this component.

- Data Preservation: The data preservation component is responsible for maintaining the fixity of data that is represented as a digital object in SDR. For each dataset ingested in SDR, the checksum is calculated and modeled in the PREMIS preservation standard. During data ingest, download, migration operations, the preservation module provides the required checks for verifying the fixity of the content within a digital object.

- Data Analysis and Curation: This module acts as a service registry for registering the community-specific data processing tasks. Each data-processing task is exposed as a REST service making it discoverable and accessible for the research communities in assembling workflows. The description of a service comprises a unique service address, input parameters, and accepted data format.

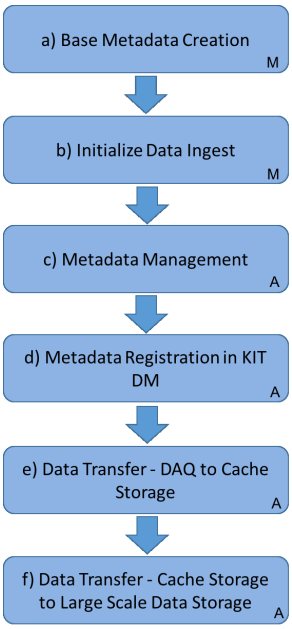


Figure 3.3: Ingest workflow [123]

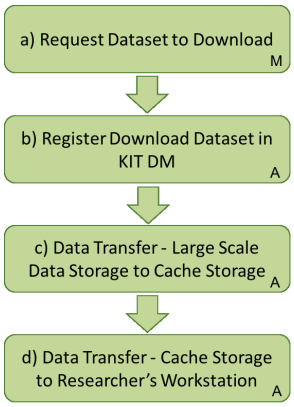


Figure 3.4: Download workflow [123]

– Data Processing: This module handles the integration with HPC cluster, with the possibility to configure the execution environment for each data-processing task.

3.3.2 Generic Client Service API

With the availability of e-Science infrastructures, *in silico* research is carried out over distributed resources under various federated zones. This has led to a separation of data acquisition systems (DAQ) from data processing and storage systems.

The primary requirement in any distributed execution environment is to enable the transfer of data between various systems within the federated zones. Hence, in this section, we present the Generic Client Service API (GCS API). The GCS API is a data agnostic transfer client that allows transfer of data acquired from different DAQ, tools, or systems used by the researchers to the scientific data repository. The GCS API follows a systematic workflow for ingesting and downloading data from external systems to and from SDR.

Ingest workflow. The ingest workflow, shown in Figure 3.3 comprises of six steps. The steps that are to be performed manually through user interaction are marked with 'M', while the steps that are automated by the GCS API are marked with 'A'. Except for the first two steps the entire ingest procedure is automated.

(a) Base Metadata Creation: In the first step, a user needs to create basic administrative metadata required for registration of data that is to be ingested. In order to comply with the standards supported by the KIT DM, the metadata

needs to be described in the Core Scientific Metadata Model. There are multiple options for creating the base metadata. For example, metadata embedded within data can be used for registration, a community-specific user interface can be integrated with GCS API for researchers to enter the metadata or an XML file containing the metadata can be submitted together with the data.

(b) Initialize Data Ingest: After creating base metadata, the data to be ingested can be initialized for transfer. In this step, the user can either manually select the appropriate protocol required for transferring the data, or the GCS API automatically analyses the data, and as per the size of data that has to be transferred, a transfer protocol is chosen. Currently, WebDAV, FTP, and GridFTP protocols are supported. Moreover, for optimizing the data transfer, the fine-tuning of the protocol through the selection of protocol instances can be configured in this step.

(c) Metadata Management: Depending on the format in which metadata is submitted for registration, in this step, the adapters necessary for extraction and translation of the metadata into the CSMD standard are deployed. For example, in the case of medical datasets, metadata embedded in the DICOM file is extracted and mapped to the CSMD standard. These metadata translation adapters have to be implemented by the research communities for datasets where the data is embedded within the data.

(d) Register Ingest-Data in KIT DM: The underlying KIT DM service stack is responsible for managing the physical storage of data, allocation of storage resources, and assignment of Persistence Identifier (PID) to each digital object. For allowing data discovery and referencing, the metadata extracted in the previous step is registered in the KIT DM.

(e) Data Transfer - DAQ to Cache Storage: After registration of data in KIT DM, the data acquired from the scientific instrument is transferred to the cache storage connected to the data repository. The cache storage is under the control of the KIT DM and provides access to Grid computing cluster for allowing processing of data. In this step, any additional data curation required before the archival of data can be performed. Data-specific staging processors can be implemented and deployed in the KIT DM service stack. These staging processors are activated on the arrival of data in the cache storage.

(f) Data Transfer - Cache Storage to Large Scale Data Storage: For data that needs to be archived, an automated server-side process is triggered. This process transfers data from the cache storage to the allocated large scale data storage. For example, at KIT a Large Scale Data Facility (LSDF) with the storage capacity of 6 PB is provided by the Stenbuch Center for Computing (SCC) [155]. Moreover, in this step, the low-level services execute various data preservation activities, where the preservation metadata is verified and stored in the PREMIS standard.

Download Workflow. Data stored in the repository needs to be made accessible either for further analysis or for sharing and disseminating the knowledge gained through obtained results. For this, in Figure 3.4, we present the download workflow followed for each dataset requested either by a system or a user.

(a) Request Dataset to Download: For simplicity reasons, the user only needs to provide the unique persistence-id for initiating data download. In the case of a dataset that is retrieved via full-text search, the workflow internally fetches the persistence-id of the requested dataset.

(b) Register Download Dataset in KIT DM: Similar to the data registration step explained in the ingest workflow, the first step in the download workflow is the registration of data requested for download. This is necessary for keeping track of data that has been exported by SDR. Minimum metadata such as user-id, time-stamp, data-transfer protocol, and persistence-id of data is registered.

(c) Data Transfer - Large Scale Data Storage to Cache Storage: After successful registration of download data in KIT DM, an internal process with the requested transfer protocol triggers the transfer of data from the large scale data storage to the cache storage. Due to data management policies implemented by the KIT DM, external user or system has no direct access to the large scale data storage. Moreover, a copy of original data is transferred, and this data at a given location is never modified or removed.

(d) Data Transfer - Cache Storage to Researcher's Workstation: The final step in the download workflow is the transfer of data from the cache storage to researcher's workstation or application. The selection of appropriate data transfer protocol with multithreading enabled and monitoring for a fail-safe transfer are automatically configured in this step.

3.4 Evaluation

In this section, we focus on performance evaluation of the GCS API, the evaluation of other components namely metadata management and workflow provenance are presented in their respective chapters. For evaluating data transfer performance of the GCS API, we use the nanoscopy dataset. Each dataset ingested or downloaded using the GCS API consists of a stack of high-resolution images that are in TIFF, KDF, or HDF5 format. The ingest and download tests are based on the workflows defined in the previous section. Following setup is used for performing the tests. The SDR is deployed on an Intel Xeon server machine with six cores, 132 GB RAM and 11 TB of cache storage. The client machine hosting the GCS API is an Intel i7 machine with four cores, 16 GB RAM and 500 GB of storage. The client machine represents a typical DAQ system that is connected to the high-resolution microscopes. The machines are connected using a 1 Gigabit Ethernet connection. The tests are

composed using Apache JMeter and the ingest and download results are shown in Table 3.1 and Table 3.2 respectively.

Table 3.1: Data Ingest Workflow Result

Data Size [GB]	Total Execution Time	Data Transfer Time
1	30 sec	17 sec
10	215 sec	184 sec
20	373 sec	344 sec
40	775 sec	758 sec
80	1445 sec	1434 sec

Table 3.2: Data Download Workflow Result

Data Size [GB]	Total Execution Time	Data Transfer Time
1	45 sec	18 sec
10	364 sec	198 sec
20	654 sec	353 sec
40	1355 sec	778 sec
80	2658 sec	1457 sec

The results shown in Tables 3.1 and 3.2 are obtained with the WebDAV implementation in GCS API. Data transfer rate ranging between 55-60 MB/s is achieved over a 1 GB/s network connection. There is a noticeable difference in the Total Execution Time seen in Tables 3.1 and 3.2, and this is because during the download process the requested dataset needs to be transferred from the large scale data storage to the cache storage. Even though data is transferred via a high-throughput connection, additional time is required as per the size of the requested dataset. The GCS API can be optimized by configuring it to use multiple instances of WebDAV protocol instances to run in parallel. Various configurations are tested, and optimum results are achieved by configuring the GCS API to use five instances of WebDAV protocol instances in parallel. Other configurations are also tested, with one, two and ten WebDAV protocol instances for the same data sizes. It was observed that configuring the GCS API to spawn a single WebDAV protocol instance for transferring multiple files is a bottleneck. This is because, a single instance of WebDAV instance is shared between multiple files, and only a single file can be transferred at a given time. Theoretically, having the same number of WebDAV protocol instances as the number of files to transfer should provide optimum results. However, data transfer rate for each WebDAV protocol instance is significantly small (2-5 MB/s). This drastic reduction in data transfer rate is due to the overhead associated with the creation of separate WebDAV instance for each file. It is yet to be seen what performances are achieved

using other data transfer protocols and with a higher bandwidth network connection, for example, a 10 Gb/s network connection.

3.5 Use cases

The SDR architecture presented in Section 3.3 has been adopted for three research disciplines, namely nanoscopy [123], angioscopy [126], and eCodicology [31]. In this section, we present two published results describing the adoption of the SDR architecture for handling data and metadata of nanoscopy and angioscopy experiments.

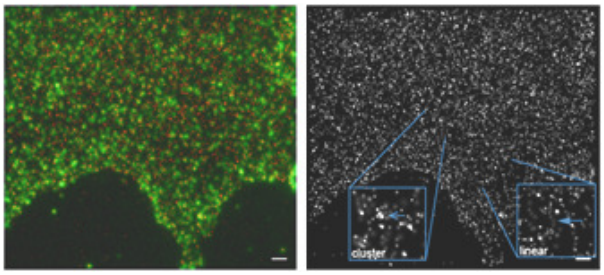


Figure 3.5: Image section of the membrane of a breast cancer cell after specific labeling of the Her2-receptors. Left: Merged image of wide-field (green) and nanoscopy image (red); right: nanoscopy image (each point represents one single molecule) with typical structures (insets).

3.5.1 Nanoscopy

The first adoption of the SDR and the GCS API was undertaken for handling data and metadata generated in a nanoscopy workflow. The results of this work are published in: *An optimized generic client service API for managing large datasets within a data repository* (BIGDATASERVICES 2015) [123].

Light microscopy has become a routine imaging technique in biological and medical research as well as in medical diagnosis. The resolution gap between conventional light microscopy (∼200 nm) and electron microscopy (∼10 nm) can be circumvented by novel approaches of super-resolution fluorescence light microscopy. One of these sophisticated techniques is Localization Microscopy (nanoscopy) bridging this resolution gap while maintaining the native cellular environment. An example nanoscopy image is shown in Figure 3.5. Fluorescent markers accurately tag molecules undergoing blinking processes during the acquisition of time stacks of about 1,000 image frames of the same image section. The registration of the blinking event allows optical separation of individual molecules. After image processing, spatial positions of all individually tagged molecules are determined with nanometer precision leading to a resolution in 10 nm range. Hence, based on the generated large datasets new insights into cellular systems can be gained.

During the last decade, nanoscopy has made rapid progress towards routine applications in biophysics. The high-resolution techniques are based on a combination of sophisticated computer image analysis with a dedicated imaging strategy. One

embodiment of nanoscopy is called Spectral Precision Distance Microscopy/Spectral Position Determination Microscopy (SPDM) [36]. Here, we refer to SPDM systems established at the University of Heidelberg and the Institute of Molecular Biology Mainz. For historical reasons, the data-format in which the datasets are produced vary for each microscope. Each dataset is a collection of heterogeneous files, which can be Hierarchical Data Format (HDF5) files, KDF files or Tagged Image File Format (TIFF) image-stack files. Presently, large datasets produced by an execution of a nanoscopy workflow are in the range of few TBs, but with activation of multiple-color channels, the volume of data is expected to approach 150-200 TB. This data volume is 100 times more than the data volume generated using conventional fluorescence microscopes. For example, high-resolution images produced for just 50 sections (1 section covers only a few μm^2) of a cell membrane ($50 \times 1,000$ image frames) lead to a data volume of 50 GB. A systematic series of measurements that is the acquisition of 100 cells per slide and 20 slides per study leads to a total data volume of about 100 TB. The metadata is a critical component of each dataset, as it enables further reuse and reference. The associated metadata is partly embedded in the dataset itself and partly produced in an additional file during the experiment.

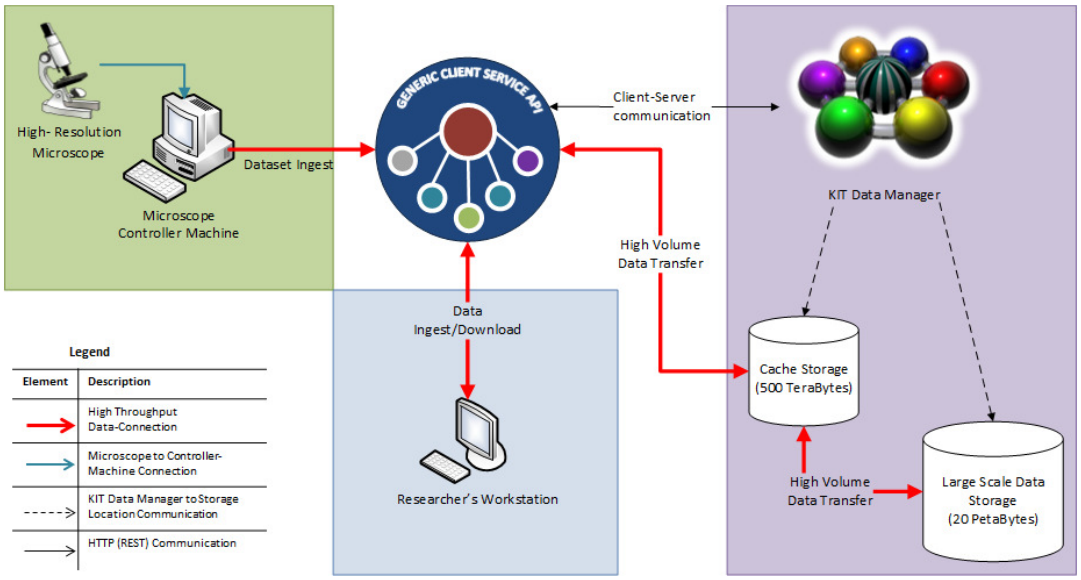


Figure 3.6: Integration architecture of nanoscopy research systems with GCS API and NORDR [123]

Nanoscopy infrastructure architecture layout

The principle idea behind the setup of the nanoscopy infrastructure is to harmonize the interaction between various heterogeneous systems that span multiple locations. The architecture layout as shown in the Figure 3.6, is designed around the GCS API.

– Microscope Controller Machine: The microscope controller machine is connected to the high–resolution microscopes (DAQ system). The experiment-datasets

are generated and temporarily stored on this machine. The main focus is to provide an immediate data transfer between this machine and the large scale data storage. For this, we integrated the GCS API in the interface of the controller machine.

– Researcher's Workstation: The researcher's workstations are located at multiple institutes other than those of the microscopes. The main focus is on high-throughput data access so that the researchers can download datasets for further analysis and upload of the processed results. With the realization of the GCS API as a command-line tool, the requested dataset can be downloaded, and the processed data can be uploaded.

– Generic Client Service API: The GCS API is an extension of SDR. It provides generic interfaces for interacting with disparate systems and can be seamlessly integrated with various user interfaces. The GCS API is designed in a way that it is system and end-user device independent. Hence, it can easily support new microscope controller machines or researcher's workstation from different places.

– KIT Data Manager: KIT Data Manager (KIT DM) repository system is located in Karlsruhe; it offers various functionalities via RESTful web services. The KIT Data Manager is further integrated with LSDF and the high-performance computing cluster.

– Cache Storage: The cache storage is a part of LSDF located in Karlsruhe. It is a high-performance and a high-data-throughput storage facility for enabling data-intensive computing. The cache storage provides storage capacity of 500 TB, is connected to the large scale data storage via a high speed data network (HSDN) (10-Gigabit Ethernet), has better latency and provides better workload control.

– Large Scale Data Storage: The large scale data storage also a part of LSDF is located in Karlsruhe. In the large scale data storage, datasets are stored for long-term archival. It offers a storage capacity of approximately 6 PB and is scalable for further demands.

3.5.2 Angioscopy

In handling datasets for the angioscopy scientific workflow, in this section, we present adoption of SDR in the medical environment of the Mannheim medical center. This work is published in: *Managing Provenance for Medical Datasets* (BIOSTEC 2017) [126]. Electronic health records (EHR) offer a digital documentation of the diagnostic and therapeutic history of a patient. Parts of these records are managed by Hospital information systems (HIS) and subsystems like radiology information systems (RIS).

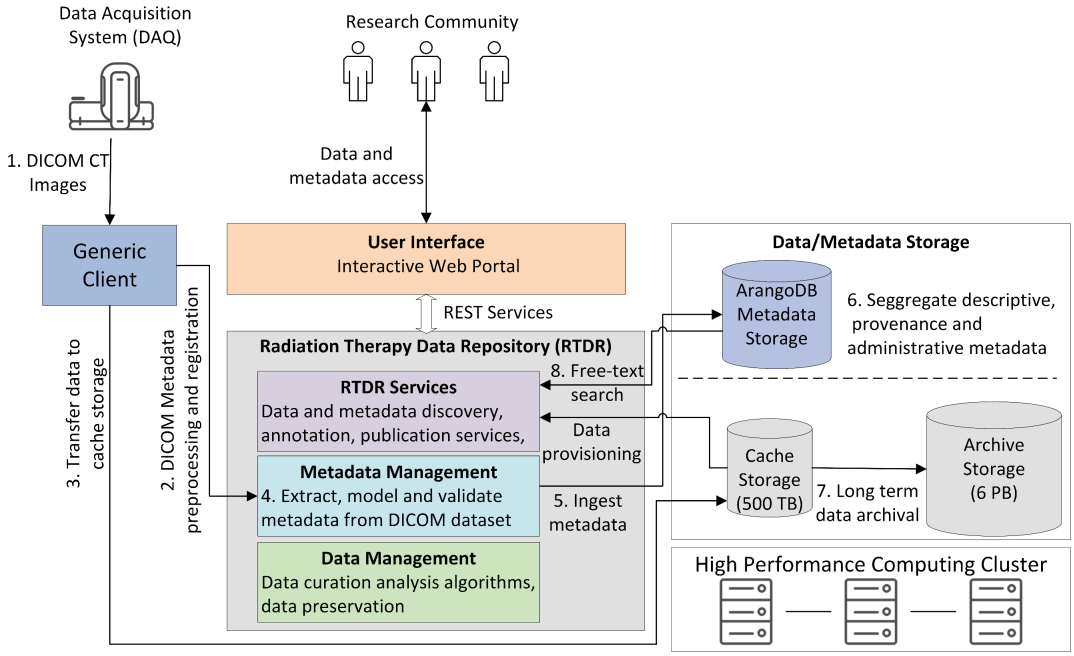


Figure 3.7: Integration architecture of clinical data acquisition systems with GCS API and RTDR [126]

Over the initiative integrating the healthcare enterprise (IHE) defined workflows enable standardized records on one hand side and as a final goal a complete coverage of all procedures in a clinic.

The datasets in interventional radiological clinics follow a systematic workflow involving the Generic Client and the Radiation Therapy Data Repository (RTDR). The Generic Client is an extension of the GCS API that we used for the nanoscopy use case. The workflow begins with the acquisition of datasets in DICOM format to the final generation of a treatment record. During each step of the workflow, the DICOM images are subject to image manipulations, with the extraction of diagnostic information. These datasets are enriched with essential metadata describing the diagnostic details of each step. Moreover, supplementary and related datasets might be created in any step during the diagnosis process. The complete workflow described in our example case consists of eight steps as shown in Figure 3.7: (1) The workflow begins with the acquisition of raw DICOM dataset from the CT scanner system. (2) The Generic Client performs the registration of the dataset to be ingested by extracting the base metadata from the DICOM metadata section, translating it into the CSMD standard and registering the dataset in RTDR. (3) A successful registration of base metadata triggers the transfer of data to the PACS server. (4) The descriptive metadata from the DICOM dataset is extracted, modeled and validated. (5) The metadata is ingested to a dedicated metadata storage database. (6) Metadata is segregated as descriptive or provenance metadata and stored either in document data model or graph data model of ArangoDB. (7) The DICOM dataset from the cache storage is transferred to archive storage for long-term preservation. The preservation metadata associated with the dataset is updated in the metadata storage. (8) The metadata is

indexed for enabling full-text search and allowing discovery of the datasets from the RTDR. The data and metadata are accessible in a Vaadin based user interface which is connected to the RTDR through the REST services.

3.6 Summary

This chapter presented the overall picture of a generic SDR architecture, and in the context of SDRs, we presented our first contribution of this thesis. The architecture of SDR follows the principle of modular design pattern that enables enrichment of its functionality by adding task-specific modules to the architecture. The first module that we developed for the SDR architecture is the GCS API and its extension the Generic Client. The GCS API provides an integration interface for ad hoc systems to submit their data and metadata directly from DAQ systems. For fail-safe ingest and download of data, the GCS API follows systematic workflows. For registration of ingest-data, the GCS API uses the simplified version of the CSMD metadata standard. The KIT DM realizes this standard for capturing the basic administrative metadata associated with each ingested dataset. The data transfer rates using WebDAV protocol on a 1 Gb connection at peak network bandwidth usage was ∼60 MB/sec. This data transfer and access rates are adequate for the nanoscopy, angioscopy, and eCodicology communities and the GCS API was deployed with this configuration on their respective DAQ systems. To describe the practical usage of the GCS API with the SDR, we presented detailed description of the nanoscopy and angioscopy use cases, with the adoption of SDR and GCS API in their architecture layout. The eCodicology adopted a similar architecture design, and the results are published as scientific contributions in the paper: *Software workflow for the automatic tagging of medieval manuscript images (SWATI)* (DRR 2015).

From the architecture it is clear that to realize a comprehensive SDR, there are multiple areas of research that need to be addressed. Hence, in the next chapter, we present an automatically adapting metadata management framework that can not only be used as a standalone system but also can be integrated into an SDR. Moreover, devoting our efforts in the research area of provenance handling in WfMSs, in the succeeding chapter we present a novel methodology for enabling provenance interoperability. Finally, the results from the various research areas are integrated to realize the complete SDR.

Chapter 4

Generic Metadata Management

A critical aspect of eScience systems, and particularly of scientific data repository systems, is the ability to handle metadata. Metadata is essential for managing various aspects of the life cycle of research data. For this, in this chapter, we propose MetaStore, an adapting metadata management framework that is able to handle heterogeneous metadata models. The MetaStore framework can be either integrated with a scientific data repository framework or can be used as a standalone metadata management system. In comparison to existing metadata management systems, MetaStore has a number of features making it a scalable and a sustainable metadata framework for handling static as well as dynamic metadata.

First, in Section 4.1, we briefly present the classification of metadata models, followed by the requirements put forth by various research communities that motivated the design of the MetaStore framework. In Section 4.2, we introduce the technologies and standards that are used in designing the framework. In Section 4.3, we present the contribution of this chapter, i.e., the modular design of the MetaStore architecture with a detailed explanation of each component.For systematically highlighting the advancements that MetaStore contributes over existing state-of-art metadata management systems, in Section 4.4, we present a feature-based and a performance evaluation of MetaStore with existing metadata management systems. Moreover, to show the generic applicability of MetaStore in handling diverse requirements from research communities, in Section 4.5, we present three use cases for which the MetaStore framework is adopted. Finally, in Section 4.6, the chapter presents an in depth discussion on the various software engineering design decisions and benefits of the MetaStore framework, concluding with the summary of the chapter in Section 4.7.

4.1 Introduction and Motivation

Currently, with advancements in research techniques and novel data acquisition and processing systems at our disposal, there is an exponential growth of data and metadata that are generated by scientific experiments [40, 78]. Previously, due to data-driven research, data was considered as the first-class entity with prime importance,

whereas metadata was considered as auxiliary information of lesser significance. Moreover, many research communities have adopted data-driven strategies, with the supporting technologies that are developed considering data as the focal point. In recent years, the availability of comprehensive metadata models has led scientific communities to orchestrate complex scientific data life cycle using metadata. However, the advances in metadata research were limited within the scope of a research community, and the appropriate channels for disseminating these results for a wider adoption is not possible. For this, multiple international research groups such as the Metadata Standards Directory Working Group, Metadata Interest Group, Empirical Humanities Metadata Working Group, Metadata Standards Catalog Working Group within the Research Data Alliance (RDA)[1] came into existence. These groups are exclusively focused on providing outcomes for various aspects within the broad spectrum of metadata research. For example, the Metadata Interest Group has provided a minimal metadata model as an outcome, whereas the Metadata Standards Directory and Metadata Catalog Working Groups have established a metadata registry containing more than hundred metadata models. Our active participation in these working groups is one of the reasons for investing our efforts in metadata research.

In the research area of digital curation, Higgins. S. [81] states that: "Metadata is the backbone of digital curation, without it, a digital resource may be irretrievable, unidentifiable or unusable." Moreover, Qin et al. [131] elaborates the necessity of metadata, and explains the broad range of functions that metadata serve in large eScience infrastructures. In the following, we list important purposes of metadata: (1) adequately described metadata facilitates data discovery, selection, aggregation, and reuse, (2) for allowing data verification, metadata should describe the provenance allowing researchers to verify the quality of data and enable its reuse, (3) with context-specific metadata description, data can be made available for analysis purpose, and (4) with an adoption of a standard metadata harvesting protocol, data and metadata can be shared and made interoperable with external systems [131].

Metadata can be broadly classified into static metadata and dynamic metadata [71]. Static metadata is not subject to any changes over the entire life of data, even if the underlying data evolves. For example, metadata describing the preparation of a specimen and the configuration of instruments for a nanoscopy investigation is static metadata that will never be changed, and any changes to this metadata will invalidate the investigation and subsequent results. Contrarily, dynamic metadata is subject to change, i.e., as data evolves its metadata may also change. For example, an annotation describing a sub-cellular structure of a cancer cell may change due to advancement in data acquisition techniques, modification to image processing algorithms may yield better quality images, or with the availability of detailed domain knowledge the annotation might be updated by a domain expert.

[1] https://www.rd-alliance.org/

Based on applications of metadata, there exists a classification of metadata standards from the field of digital libraries. The existing metadata standards are categorized into three main types: descriptive metadata, structural metadata, and administrative metadata [128]. Descriptive metadata includes details of the resource (data) and is used for searching and identifying the latter. Additionally, descriptive metadata can also be tagged to a given resource in the form of annotations. For example, in medical research, it is a common practice for medical-experts (physicians) to attach additional information on the patient's medical records in the form of annotations [50, 84]. Structural metadata describes the internal structure of a compound resource and how it is formulated. For example, structural information among the pages of a medieval manuscript can be described via structural metadata, while details necessary to manage a resource, such as access and rights management, resource preservation information, provenance and technical information are described using administrative metadata. For example, file format, data schema, and file checksum.

In the research area of metadata, provenance has become an independent and a well-established area of research. The reason is that various scientific communities have realized the importance of provenance in substantiating, reproducing, and qualitatively assessing their scientific results. As a part of metadata research in this thesis and to advance the research in provenance, we will describe our contributions thereupon in the next chapter.

The motivation behind design and implementation of the MetaStore framework is to provide a generic and a reusable metadata management system that can be adopted by multiple scientific communities with their different needs. This aim is further strengthened by the requirements put forth by the various research communities. In the following, we summarize these requirements:

Metadata schema support

Research in several disciplines is rapidly changing, especially the nanoscopy and eCodicology research areas are continuously evolving and their metadata schemas are frequently modified. In nanoscopy research, a community-specific metadata schema is created to model the experiment description[2], and workflow provenance is described using ProvONE. Similarly, the results of eCodicology workflows are modeled using the standard PAGE XML schema[3]. Hence, the requirement is to design MetaStore to support both, standard metadata schemas as well as community-specific metadata schemas. Additionally, to systematically handle the evolution of metadata schemas in order to mitigate redundant modifications and enable tracing of different schema versions, MetaStore should provide the registration of each metadata schema with its version. Moreover, nanoscopy and eCodicology research communities have adopted METS to systematically organize multiple metadata schemas that are required by the community to describe their data comprehensively. Hence, an extension to basics

[2]http://datamanager.kit.edu/masi/localizationmicroscopy/2016-03/LocalizationMicroscopy.xsd
[3]http://www.primaresearch.org/schema/PAGE/gts/pagecontent/2017-07-15/pagecontent.xsd

metadata schema support requirement is for MetaStore to also handle community-specific METS-profiles[4,5].

Full-text search

For the majority of the communities, the data is huge and serialized in machine-readable file formats, and frequently accessing the data is not efficient due to hardware and network limitations. In nanoscopy and eCodicology research, raw data acquired either from high-resolution microscopes or scanners is ingested in a data repository. For enabling further reuse of this data by either data-processing workflows or domain-specific applications, the data needs to be made discoverable. For this, metadata is critical for searching, identifying, and retrieving required data. A standard approach is to provide a set of queries for each metadata schema. However, this is not a sustainable approach, because for frequently changing metadata schemas, the implemented queries will quickly become obsolete and return either incomplete or incorrect results. Moreover, implementing a set of queries for each metadata schema is an arduous and resource intensive task. Thus, the requirement from the communities is to have a full-text search over all the metadata.

Metadata quality control

The next requirement is to provide automated metadata quality control for verifying the well-formedness of metadata, schema conformance, and content verification for communities possessing controlled vocabularies. Metadata can be automatically generated by a data acquisition system or during the execution of workflows, or metadata in the form of annotations can be manually created and tagged with the data by domain experts. Thus, irrespective of the source from which the metadata is generated, for maintaining a basic level of metadata quality, the MetaStore framework should provide different levels of automated quality control checks.

Metadata harvesting

The metadata should be provided for allowing harvesting, i.e., from the collected metadata, MetaStore should provide either partial metadata harvesting, in the case of sharing only a specific part of metadata to an external system or complete metadata harvesting, in the event of data migration from one data repository to another. For enabling seamless integration of MetaStore with existing systems, and compliance with existing metadata harvesting standards, the MetaStore framework should support the OAI-PMH specification. In the eCodicology project, the processing of digitized-manuscripts results in large volumes of multi-dimensional metadata. As this metadata needs to be visualized using the CodiViz [32] tool, selective metadata harvesting is requested by the humanities scholars. Thus, a common requirement from the research communities is to allow large-scale metadata sharing.

[4] http://datamanager.kit.edu/masi/localizationmicroscopy/mets/nanoscopy-METS-profile.xml
[5] http://zimks68.uni-trier.de/stmatthias/T1108/T1108-digitalisat.xml

Provenance support

The nanoscopy data processing workflows are described using the BPEL language, while the eCodicology workflows are described in SCUFL. During the execution of these workflows, it is necessary to capture and model the comprehensive workflow provenance. For both research communities, provenance is critical metadata that assists in validating the quality of results, enables data reproducibility, provides systematic tracking of workflow evolution, and allows execution of graph pattern queries for analyzing provenance traces. Moreover, as MetaStore can also be coupled with a scientific data repository, it is necessary to support the PREMIS provenance model used in long-term data and metadata preservation. Hence, the requirement is to not only provide automated workflow provenance handling capabilities using a comprehensive provenance model like ProvONE but also support the data preservation in the PREMIS model.

Automatic/Semi-automatic annotations

Currently, in the field of digital humanities, research communities often apply image processing workflows that generate metadata at each step. For the eCodicology community, the SWATI workflow [32] is employed to extract Optical Layout Recognition (OLR) features from the digitized medieval manuscripts. As the SWATI workflow is continuously evolving with improved layout recognition algorithms, the extracted OLR features (metadata) are also frequently changing. Hence, the first requirement for MetaStore is to store this metadata in the form of annotations and make it available for analysis. Furthermore, these annotations are subject to modification by research experts through the CodiLab toolset[6], which is an annotation framework providing visual interpretation of annotations by overlaying them over digitized manuscripts. Thus, the second requirement is to allow semi-automated annotations of images and documents, with an extension for controlled vocabularies.

Miscellaneous

In the following, we list the additional requirements requested by the research communities: logically linking the metadata with the data through Persistence Identifier (PID) generated from a PID management systems like the Handle System[7], high-performance metadata access and ingest, scalable storage for handling increasing metadata volume, and a REST API for integrating MetaStore with existing systems.

4.2 Preliminaries

In this section, we present the technologies and concepts used in designing a generic metadata management framework. In principal, for handling both static and dynamic metadata, the MetaStore framework is based on NoSQL database technology. For

[6] https://github.com/JochenGraf/CodiLab
[7] http://www.handle.net/index.html

understanding the principle of NoSQL database, first, we explain the CAP theorem and the BASE principle of the NoSQL databases, followed by the description of the WADM, and the OAI-PMH metadata harvesting specification.

4.2.1 NoSQL Databases

NoSQL stands for Not Only SQL. It is a new type of database system that is a paradigm shift from the typical relational database management systems. The prime reason for choosing a NoSQL database system for the MetaStore framework is due to the need of a flexible data model in handling arbitrary metadata models. Moreover, with integral support in horizontal scalability for handling large metadata volumes, replication for metadata backup, and efficient query performance as compared to SQL databases led us to choose a NoSQL database system. However, there are more than thirty different types of NoSQL databases, each serving a different purpose. Thus, to systematically organize these NoSQL databases, they are categorized by data model and application they support.

- Key-value stores: Data in these systems is stored as a pair of key-value, where the value contains the data, and the key is a unique identifier index for retrieving the value. For example, the two prominent key-value databases are RIAK and REmote DIctionary Server (REDIS).
- Document stores: These systems store the information as a semi-structured document. Typically, the data is serialized in JSON format, with an automatic assigning of a unique identifier for identifying each document.
- Graph stores: A graph store models the data structure for schema and instances as graph structures or generalization of them, with the graph containing nodes, edges, and properties to represent and store the data [9].
- Wide-column stores: In a columnar database, each database table column is stored contiguously in a separate location on disk. For improved read efficiency, the columns are densely packed with some compression techniques [1].

The existing NoSQL databases, irrespective of the data model which they support are based on the principle of the CAP theorem, explained below.

CAP Theorem. The CAP theorem put forth by Brewer. E. states that any network-distributed data system can guarantee at the most two out of the following three properties, Consistency, Availability and Partition tolerance [25], the theorem is further proved by Gilbert et al. [64]. Thus, any network-distributed data system can guarantee the following pair of properties simultaneously: Consistency and Availability (CA), Consistency and Partition tolerance (CP), or Availability and Partition tolerance (AP). In the following, we briefly explain each property of the CAP theorem, followed by the pair-wise explanation of them.

Consistency. The Consistency property of the CAP theorem represents Linearizability [77]. Linearizability provides a strong recency guarantee on data but is a weak consistency model [64]. For the CAP theorem, this means that each request to the system will return the correct response, i.e., the response will return an up to date copy of data.

Availability. The availability property of the CAP theorem states that the system should be continuously available, i.e., even if there are failed nodes in a system, the requests received by the system must return a (non-error) result in the response.

Partition tolerance. A distributed system is said to be partition tolerant if it continues to work despite loss of an arbitrary number of messages exchanged within the system. For the CAP theorem, the partition tolerance is the key property that confirms an uninterrupted use of distributed system and hence, is considered as the default property.

For any distributed system, the CAP theorem states that out of the three properties only two can be satisfied by a system. In the following we explain the various configurations that a distributed system can have:

- CA: For a CA configured system, the partition-tolerance property is sacrificed, i.e., the entire system is running on a single machine, which contradicts the notion of the CAP theorem.
- CP: For a CP configured system, the availability property of the system is sacrificed. This is because for a transaction (write or read) to be consistent across a network partitioned system, it is necessary that all nodes be synchronized with the latest write operation, or with the current result of a read operation. Thus, for a CP system, the nodes separated by a network partition have to be unavailable until the entire system is in a consistency state. For example, in a CP configured system S with three nodes n_1, n_2, n_3, when a write operation w_j is performed on node n_1 at time instance t_n then any read operation on nodes n_2, n_3 cannot be fulfilled by the system until the write w_j is not propagated to the remaining nodes n_2, n_3, i.e., the system is locked until all the nodes are updated.
- AP: For an AP configured system, the consistency of a system is sacrificed. This is because, for keeping a system continuously available, it is necessary for all the nodes in a system to respond to any arbitrary number of requests. For such type of system, whether the response data is up to date or stale is not vital. What is important is that each request should return a response. For example, in an AP configured system S with three nodes n_1, n_2, n_3, if a write w_j is performed on node n_1 at time t_n, then any read or write operation requested to S at the

same time t_n should be fulfilled, even if the write w_j has not propagated to the node n_2, n_3, or even if any number of *N-1* (N is total number of nodes) nodes have failed.

BASE Consistency Model. The BASE consistency model stands for Basically Available Soft state Eventual consistency. The BASE principle is logically opposite to the ACID principle of the relational database management system [129]. In the following, we explain each term of the BASE principal.

- Basically Available: The constraint state that the system guarantees the availability of data, even if one or few of the nodes have failed. However, there is no guarantee that data retrieved is up to date. For example, in a network-distributed system N with three nodes n_1, n_2, n_3, if a write w_j operation is performed at time t_n on node n_2, and the node n_2 crashed without propagating this write to other nodes. Then, a read operation at time $t_n + 1$ will have a response, but as the read operation is provisioned either by n_1 or n_3, a response containing stale data will be returned.
- Soft state: The Soft state in BASE principle completely abandons the Consistency property of the ACID principle, i.e., the overall state of the system is volatile. During times when there are no active write or modification operations on a system, there can still be changes to the data due to the pending write or modify operations. For example, at time instance t_n a write operation w_j took place on node n_2. Let's say that the propagation of this write to other network-distributed nodes in the system require two seconds each. Then for the next four seconds the complete system is in a soft state, until the nodes n_1 and n_3 are successfully updated.
- Eventual consistency: At a certain point in time when there are no write or modify operations been performed, the system will eventually become consistent. That is, all the network-distributed nodes will have the same state and return the last updated value. This form of consistency in a database system is a type of weak consistency, the most popular system that implements eventual consistency is the Domain Name System (DNS). For example, if a write operation w_j is performed on a node n_1 at time t_n, then after a certain time interval $t_n + \Delta$, this write operation will be communicated to remaining nodes.

4.2.2 Web Annotation Data Model

In many research areas, additional information in the form of annotations is assigned to data that is either provided by research experts (data curators) or automatically during data-processing stages [21, 50, 84]. Annotations are typically semi-structures texts that make use of keywords and controlled vocabularies. Annotations typically represent research experts' accumulated knowledge and is a vital part of data that is to

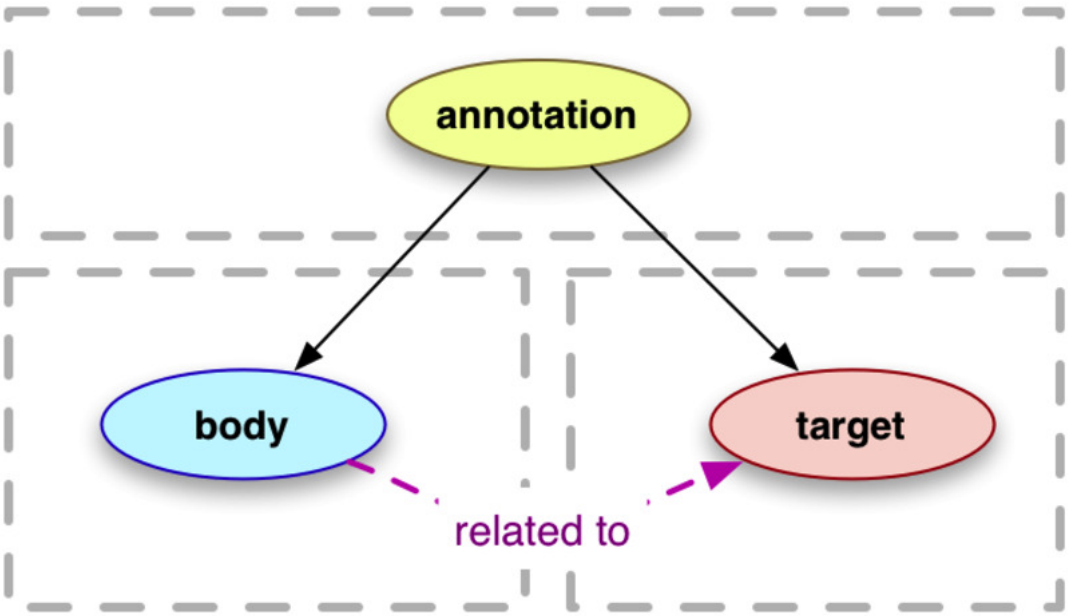


Figure 4.1: Web Annotation Data Model [136]

be published. For systematically modeling and enabling annotations interoperability, i.e., for allowing sharing of annotations between different platforms, it is critical to adopt a standard annotation model. For this, we integrated the WADM in MetaStore. The core specification of WADM is shown in Figure 4.1, and comprises the following three elements.

- annotation: The annotation is the root element that represents a web resource defined by the mandatory *oa:Annotation* class, where *oa* is the namespace that contains the definition of the *Annotation* class. An annotation can have additional descriptive information such as creating agent of an annotation, date of creation, or additional rights assertions.

- body: The body of the annotation contains the description describing a target. For assigning descriptive information, the class *cnt:ContentAsText* should be assigned to the body, where *cnt* is the namespace holding the definition of *ContentAsText*. A body itself can have its own properties and relationships, including the creation or descriptive information.

- target: The target represents an external resource that is to be annotated. Each target contains an Internationalized Resource Identifier (IRI) that points to the location of a resource. For enabling rendering of a resource, the target should be typed with the correct class. For example, a type can be *Dataset, Image, Sound, Text, or Video*.

All the three elements of the WADM are extensible by integrating external namespaces that contain vocabularies for additional classes and properties. For example, the namespace `http://www.w3.org/ns/oa` identified by the prefix *"oa"* is the default

vocabulary for the WADM classes and properties. However, for an enriched description about an annotation, it is possible to use existing vocabularies such as Friend Of A Friend (FOAF), DC, or/and RDF.

4.2.3 OAI-PMH

The OAI-PMH provides a low-barrier mechanism that enables repository interoperability by exposing the metadata through a standard specification. In principal, for realizing the OAI-PMH specification, there are two classes of participation, Data Providers that implement the recommended six verbs as a mean for exposing the metadata, and Service Providers that are clients that harvest metadata from the Data Providers for exposing additional value added services. Typically, the purpose of the harvester is to fetch metadata from a Data Provider and publish it in a search engine or an external database for allowing a search or metadata-based analysis.

OAI-PMH Verbs

The OAI-PMH specification provides six verbs, out of which three are for retrieving administrative metadata describing the metadata repository (ListMetadataFormats, ListSets, Identify), and remaining three verbs are for querying the metadata from the repository (GetRecord, ListRecords, ListIdentifiers). Each verb supports certain parameters for defining the precise details of the request.

- Identify: This verb is used to return administrative information about a repository. The mandatory fields that have to be returned in the response are: (1) `repositoryName:` a human readable name of the repository, for example, MetaStore. (2) `baseURL:` the base URL of the repository. (3) `protocolVersion:` the version of OAI-PMH specification version. (4) `earliestDatestamp:` (5) `deleteRecord:` the manner in which the repository supports deletion of a record. (6) `granularity:` the finest level of harvesting granularity supported by the repository.

- ListMetadataFormats: This verb is used to list the metadata formats available in a repository. An optional argument, `identifier` specifies a unique identifier of a record for which the available metadata formats are retrieved.

- ListSets: This verb is used to retrieve the set structure of a repository that is required when performing selective harvesting.

- ListRecords: This verb is used for harvesting records from a repository. Optional arguments `from`, `until`, `set`, `resumptionToken`, `metadataPrefix` permits selective harvesting of records.

- GetRecord: This verb is used to retrieve a specific metadata record from a repository. The verb has two required arguments; an `identifier` that specifies

the unique identifier of a record, and a `metadataPrefix` that specifies the format that should be included in the record.

– ListIdentifier: The verb is an abbreviated form of ListRecords, retrieving only headers rather than records. Similar to the ListRecords verb, the optional arguments permit selective harvesting of headers based on set membership `set`, and/or datestamp using `from` and `until`.

To enable metadata sharing through a standard protocol, a metadata management system, which is typically integrated with a scientific data repository should implement these six verbs and expose them through the Data Provider. The Data Provider (server) processes the requests sent from the Service Provider (client) and returns a valid XML that conforms to the schema defined by the OAI-PMH specification. Based on the technologies mentioned above, in the next section, we describe the architecture of the MetaStore framework.

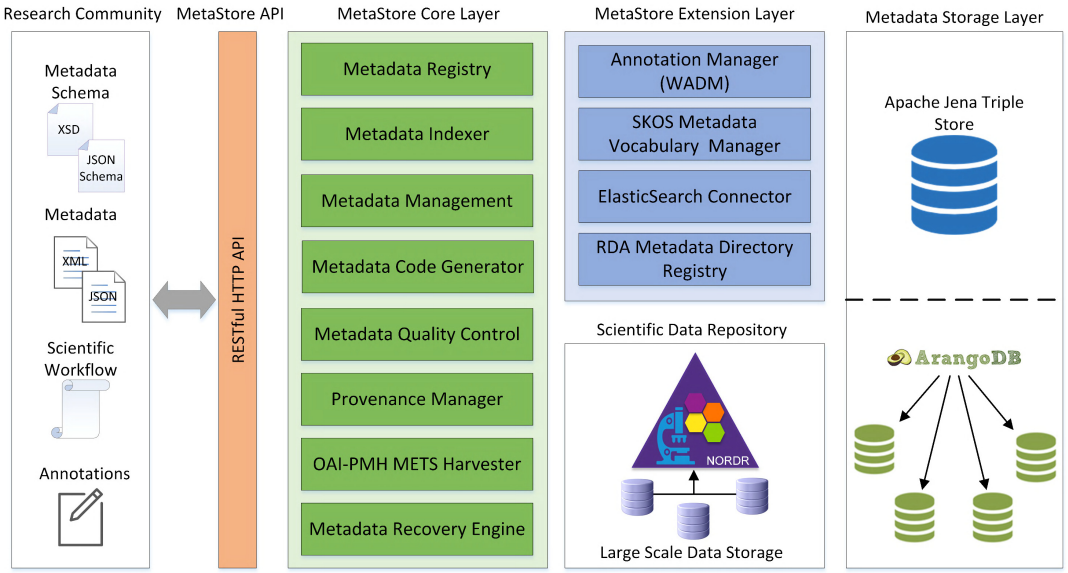


Figure 4.2: Multilayered architecture of MetaStore [127]

4.3 MetaStore Architecture

In this section, we present the multilayered architecture of the MetaStore framework (see Figure 4.2). The architectural design of MetaStore is based on following design principles.

Modular design: Each functionality in MetaStore is available as a separate component (module) so that any enhancement or technology change can be systematically handled by updating or replacing that specific module, and without affecting the functionality of other components in the framework.

Adaptive feature enhancement: MetaStore is designed as an entirely adaptive framework that modifies itself in handling ad hoc metadata models. For this, we

designed MetaStore based on the principle of Compositional Adaption [110]. Compositional Adaption can be realized with the following technologies: Separation of Concerns (SoC), Computational Reflection, and Component-based design. Out of these technologies, we adopt the Component-based design architecture. The Component-based design architecture supports two types of compositions; static and dynamic composition. In static composition, a program is composed at compile time and cannot be changed without recoding, whereas in a dynamic composition, components can be added, removed, modified, or reconfigured at runtime. To design MetaStore as a dynamically adapting framework, we realized the dynamic composition.

The complete functionality of MetaStore is divided into two layers: (1) The **MetaStore Core Layer** provides the basic functionalities necessary for handling the static metadata, and is designed independent of any scientific data repository system or metadata storage technology. (2) The **MetaStore Extension** Layer allows integration of third-party libraries and tools necessary for handling the dynamic metadata and supports handling of SKOS modeled controlled vocabularies that are specific to scientific disciplines. In the following subsections, the description of each layer and the realization of the features through the different MetaStore components are presented.

4.3.1 Research Community

A research community performs metadata related interactions through the REST API exposed by the MetaStore framework. A valid metadata schema defined either as XSD or JSON Schema is submitted by the research community for registering it in MetaStore. Only metadata complying to a registered schema can be inserted in MetaStore. Similarly, for modeling provenance information in the ProvONE model, scientific workflows defined by a research community are submitted to MetaStore. Currently, workflows defined in the Business Process Execution Language (BPEL), Simple Conceptual Unified Flow Language (SCUFL), and Modeling Markup Language (MoML) are supported by MetaStore. The semi-automated annotations provided by domain-experts are captured in the WADM and SPARQL [130] end-points are exposed for searching and retrieving the annotations. An OAI-PMH metadata harvester exposed as a REST interface is provisioned to the research community for sharing their metadata.

4.3.2 MetaStore Core Layer

The MetaStore Core Layer consists of various task-specific components that collectively build the functionality of MetaStore. Each component or group of components in the MetaStore Service Layer follows a well-defined workflow to accomplish a given task. Through these components, we have implemented the features corresponding to the requirements from Section 4.1.

Metadata Registry and Indexer

For supporting systematic handling of heterogeneous metadata schemas, we designed the Metadata Registry component based on the key-value data model. Registering a metadata schema with its namespace and version allows efficient schema validation checks during the metadata quality control process.

The Metadata Registry component allows the registration of either a single metadata schema that exclusively describes administrative, descriptive, structural metadata or a community defined METS XML that may comprise multiple metadata schemas. The Metadata Schema Registry allows research communities to maintain multiple metadata schemas with different versions. In the case of a single metadata schema, the schema is registered as a key-value pair, in the key-value data model of ArangoDB, where the key is the combination of schema namespace and version and the value is the complete schema. In the case of a METS XML, the metadata schemas used in constructing the various sections of the METS profile are extracted and individually registered. Moreover, a relation map of the metadata schemas and the corresponding METS-profile is created. This relation map is necessary for segregating the metadata during the ingest stage, and for reconstructing the metadata during the harvesting stage. A reduced version of the nanoscopy descriptive metadata schema registered as a key-value pair is shown in Figure 4.3.

Typically, in NoSQL databases, indexes have to be created manually, which is an arduous and time-consuming task, especially when dealing with complex scientific metadata schemas. For enabling full-text search over metadata, we coupled the Metadata Indexer component with the Metadata Registry component. The rationale behind this design decision is to enable automated creation of indexes during the registration of a metadata schema in MetaStore. With immediate indexing of a registered metadata schema, we guarantee that full-text search is enabled from the first insertion of metadata in MetaStore. Moreover, an update to a schema does not require manual interference from the user (system administrator), because the index terms are extracted and reapplied on the database collection, thus, updating the index list.

The Metadata Indexer component implements the IndexTermExtractor algorithm

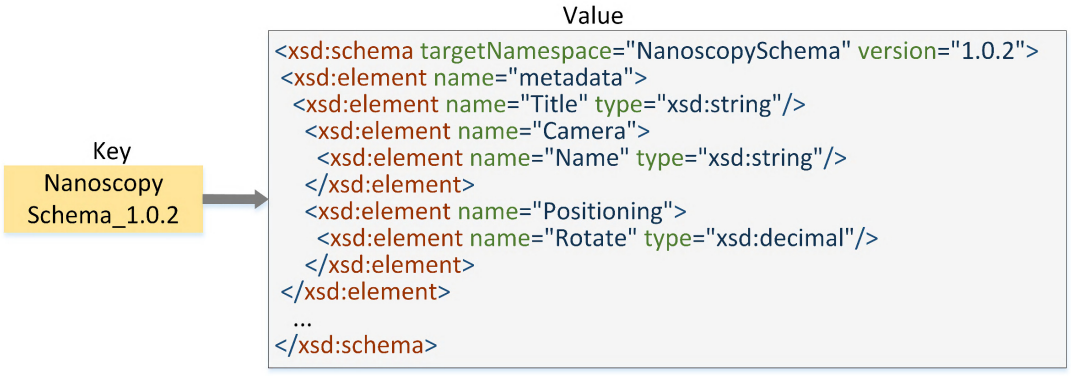


Figure 4.3: Metadata schema registration as key-value

Algorithm 4.1. IndexTermExtractor

Input: Metadata schema X serialized in XSD or XML
▷ $X = \{x_1, x_2, ...x_n\}$ is a vector of nested XML child elements
Output: List L containing index terms

```
1: List S := ∅                          ▷ list containing the various sections in METS
2: String P := ∅                        ▷ fully qualified path of the index term
3: List L := ∅                          ▷ list containing unique index terms
4: function VERIFYSCHEMA(X)
5:     if X.namespace ∈ METS then
6:         S = DECOMPOSEMETS(X)
7: ▷ the ⟨dmdSec⟩,⟨amdSec⟩,⟨structMap⟩, ... sections are extracted by this method
8:         for s_i ∈ S do
9:             P = s_i.rootElement;
10:            EXTRACTTERMS(s_i)
11:                ▷ for each section in METS, the method extractterms is called
12:        end for
13:    else
14:        P = X.rootElement;    ▷ for flat XML structure assign rootElement to P
15:        EXTRACTTERMS(X)                ▷ for a simple metadata schema
16:    end if
17: end function
18: function EXTRACTTERMS(X)
19:    for x_i ∈ X do
20:        if x_i.children ≠ ∅ then
21:            P = P ∪ x_i
22:            EXTRACTTERMS(x_i)
23:        else
24:            L = L ∪ P          ▷ Add all the unique index term paths to list L
25:            P = X.rootElement          ▷ Reset the path to the root element
26:        end if
27:        P = P.SUBSTRING(x_i)  ▷ Reset P to the current path of the element x_i
28:    end for
29: end function
```

illustrated in Algorithm 4.1. The algorithm is similar to the Depth First Search (DFS) algorithm. For an input XSD or XML (in the case of METS), the algorithm recursively iterates the entire depth of the XSD or XML, and during each iteration, a unique index term path is added to the list L. In the first step, the VERIFYSCHEMA method determines if the input schema is a simple schema or a composite schema like METS. In the case of a simple schema, the EXTRACTTERMS method is invoked. For METS files the algorithm first decomposes all the metadata sections (DECOMPOSEMETS method), and for each section, the EXTRACTTERMS method is invoked. The EXTRACTTERMS method recursively constructs a unique index path for all the elements at a depth of $n - 1$ and adds it to the list L. After completion of each recursion, the path P has to be reset to the completed element x_i. For this, the SUBSTRING method is recursively invoked for removing the sub-path that has already been added to the list L.

In ArangoDB, as the indexing of the leaf elements is contained by the element at a depth of $n-1$ (parent element), the algorithm only needs to constructs the unique paths for a depth of $n-1$ for XSD or XML with a nested depth of n. The algorithm terminates when all the elements x_i for the depth of $n-1$ are added to the list L. After the termination of the algorithm, the index terms from list L are applied to the respective metadata collection.

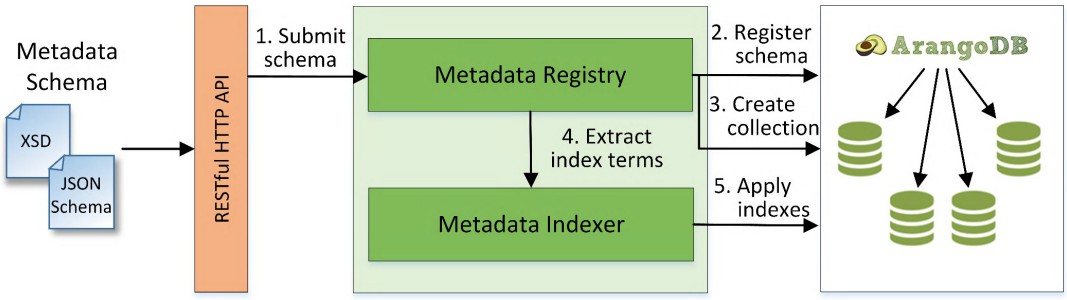


Figure 4.4: Metadata schema registration and indexing

For the default configuration of MetaStore, when ElasticSearch indexing is disabled, the metadata schema registration and indexing process illustrated in Figure 4.4 is followed. This process consists of the following five steps: (1) The scientific community submits their metadata schema through the MetaStore REST API, which is verified for the schema name and the version. (2) If the submitted schema with a given version does not exist in the schema registry, it is added to the schema registry. (3) For each unique schema registered, a corresponding collection is created in the document store of ArangoDB. This collection is required for storing the succeeding metadata, thus allowing a clear separation of community-specific metadata storage. (4) Once the metadata schema is successfully registered, the Metadata Indexer component analyzes the registered metadata schema to generate a list of index terms. (5) These index terms are applied to the respective collection for enabling full-text search. If a new version of an existing metadata schema is registered, the corresponding indexes are updated.

Metadata Code Generator

The principle aim of the Metadata Code Generator is to allow scientific communities to handle heterogeneous metadata schemas without the need to write any software code (services). To realize the dynamic composition of the services, the MetaStore Code Generator follows a two-step processes. In the first step, the MetaStore Code Generator component automatically adapts the existing MetaStore installation by creating the services that are necessary for handling the registered metadata. Once the services are generated, they are dynamically added to the existing pool of services in the Metadata Management component. However, to expose these new services through the REST interface, the Metadata Code Generator invokes the entire compilation and redeployment of MetaStore. In the second step, to prevent

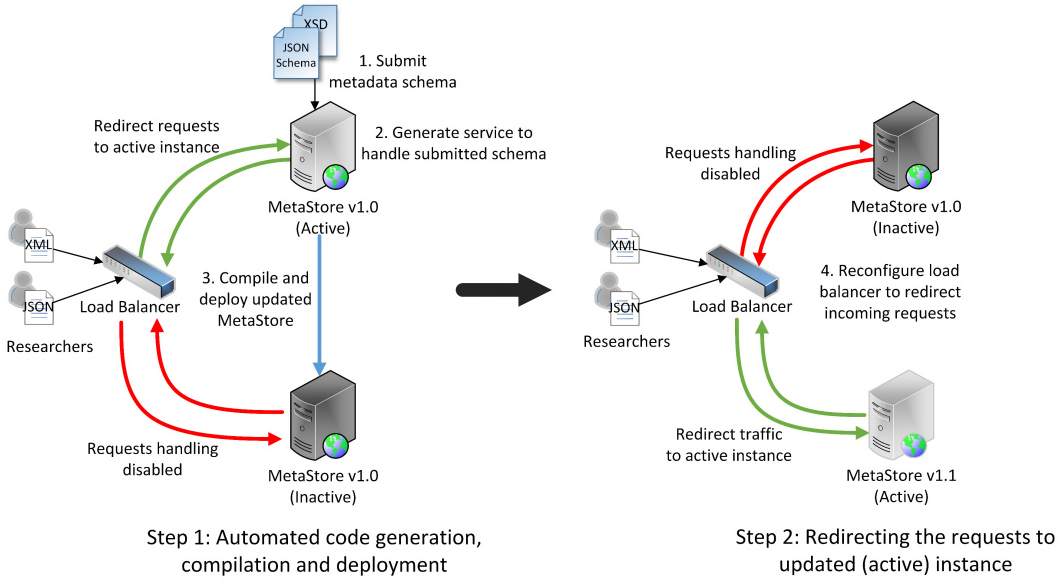


Figure 4.5: Automated code generation and deployment

the disruption of active instance of MetaStore, the recompiled version of MetaStore is deployed on an auxiliary web-server. This adaptive and automated redeployment enables a scientific community to continue an uninterrupted usage of MetaStore. For enabling an uninterrupted update of MetaStore, we use a load balancer with an auxiliary instance of a web-server that is only used during the update process. The automated code generation and deployment process for a single active instance of MetaStore is described in Figure 4.5. (1) After successful registration of the metadata schema, it is forwarded to the Metadata Code Generator component. (2) For the given metadata schema (namespace) and version, the Metadata Code Generator component automatically generates the software code (service) necessary for handling the metadata registered in the Metadata Registry. (3) The entire MetaStore with this new service is compiled and deployed on the auxiliary web-server. (4) The load balancer is configured to redirect incoming requests to the updated instance of MetaStore, simultaneously the old version of MetaStore is disabled after successful completion of the active requests.

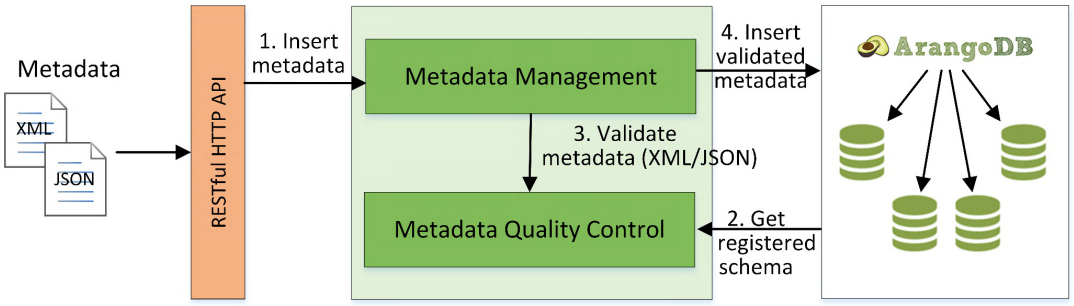


Figure 4.6: Metadata management and quality control

Metadata Management and Quality Control

The Metadata Management component exposes various CRUD (create, retrieve, update, delete) operations that are specific to a given NoSQL database (in this case ArangoDB and SPARQL queries for Apache Jena). The Metadata Management component adds the dynamically created services by the Metadata Code Generator to the pool of existing services. This component also acts as a bridge between the native ArangoDB libraries and the MetaStore REST services[8]. Additionally, the metadata is modeled in a community specified METS-profile[9], and stored with the data in a scientific data repository. To fulfill the quality control requirement from the research communities, MetaStore supports two stages of quality control: (1) schema validation and well-formedness and (2) vocabulary based metadata control. The vocabulary based metadata quality control is explained in the later section.

Figure 4.6 shows the default schema validation and well-formedness quality control process that is followed for each insertion of metadata. (1) Metadata is submitted by the scientific community in either XML/JSON format, or extracted from data and inserted through the REST API of MetaStore. (2) The Metadata Management Component analyzes the metadata to determine its corresponding schema and version from the key-value store. (3) Based on the corresponding version of the available schema, the Metadata Quality Control component performs a schema conformance and well-formedness check of the metadata. (4) The validated metadata is converted into JSON format and inserted into the designated collection.

Table 4.1: Vocabulary mapping between PREMIS and ProvONE

Rule.#	PREMIS	ProvONE	SKOS Mapping
1	Event	ProcessExec	relatedMatch
2	Object	Data	narrowMatch
3	Agent	User	broadMatch
4	relatedObjectIdentification	wasDerivedFrom or hadMember	relatedMatch
5	linkingObjectIdentifier	used	relatedMatch
6	linkingAgentIdentifier	wasAttributedTo	broadMatch
7	relatedEventIdentification	wasGeneratedBy or used	broadMatch

Provenance Manager

Provenance has a wide-range of applications. On the one hand, provenance can be used for determining the quality of the results and for analyzing and improving the quality of workflows, and on the other hand, provenance is critical for the long-term preservation of data. For efficiently handling provenance, as per its application, it is necessary to support the appropriate provenance model where required.

[8] `http://datamanager.kit.edu/masi/localizationmicroscopy/swagger-ui/`

[9] `http://datamanager.kit.edu/masi/localizationmicroscopy/mets/nanoscopy-METS-profile.xml`

The Provenance Manager supports the handling of scientific workflow provenance based on the ProvONE model and provenance for long-term preservation of a digital resource using the PREMIS model. Comprehensive workflow provenance information consists of two parts, the workflow definition (prospective provenance) and the execution details (retrospective provenance) [181]. ProvONE is a provenance model that is capable of modeling both the prospective and retrospective provenance. For efficient storage and querying of the provenance information in the ProvONE model, the ProvONE provenance graphs are stored in the Apache Jena TDB. Various query patterns for retrieving the ProvONE provenance information are implemented and exposed as REST services[8].

However, for allowing interoperability between these models, an exclusive mapping between ProvONE and PREMIS terms is unfeasible, because the ProvONE model is designed to support both prospective provenance and retrospective provenance, whereas the PREMIS standard is intended to model only retrospective provenance. Thus, to enable retrospective provenance interoperability between these models, the vocabulary mapping rules between the ProvONE (retrospective provenance) and the PREMIS model are described using the W3C SKOS mapping vocabulary specification. This mapping is an extension to an existing vocabulary mapping between the Open Provenance Model (OPM) [114] and PREMIS [134]. The mapping rules are presented in Table 4.1, and a brief explanation of each rule is given.

Rule 1: A PREMIS Event is mapped to ProvONE ProcessExec using the SKOS relatedMatch. Both the PREMIS Event and the ProvONE ProcessExec (linked to a Process class) represents an action performed on a data represented by the ProvONE Data or Collection or a PREMIS Object respectively.

Rule 2: A PREMIS Object is mapped to ProvONE Data using the SKOS narrowMatch because a PREMIS Object can only be of the type bitstream, file, or an aggregation, while ProvONE Data can be of any type.

Rule 3: A PREMIS Agent can be a either *software, person, or an organization*, and is mapped using SKOS broadMatch to ProvONE User that can represent only a *person*.

Rule 4: A PREMIS relatedObjectIdentification is mapped using the SKOS relatedMatch to ProvONE hadMember (structural relation) or ProvONE wasDerivedFrom (derivation relation).

Rule 5: A PREMIS linkingObjectIdentifier is mapped to ProvONE used class using the SKOS relatedMatch, as it represents a relation between a PREMIS Event and PREMIS Object, and ProvONE ProcessExec and ProvONE Data, respectively.

Rule 6: A PREMIS linkingAgentIdentifier is mapped using the SKOS broadMatch to ProvONE wasAttributedTo. In PREMIS a linkingAgentIdentifier represents a relationship with any of the Agents, whereas in ProvONE, the wasAttributedTo represents a relation only between a User and a Process that is associated with a ProcessExec.

Rule 7: A PREMIS relatedEventIdentification is mapped using SKOS broadMatch either to ProvONE wasGeneratedBy or used class, based on the PREMIS relationshipSubType.

In this section, we described the mapping rules between preservation provenance model and a workflow provenance model. In the next chapter, we extend these mapping rules between heterogeneous workflow specifications and provenance models to enable provenance interoperability.

OAI-PMH METS Provider and Harvester

A common requirement from the research communities is to have customizable metadata harvesting for sharing partial or entire collections of metadata. For this, we explicitly implemented the six verbs recommended by the OAI-PMH. OAI-PMH is the *de facto* standard for exporting metadata across scientific data repositories. By design, the basic interoperability using Dublin Core metadata standard [167] is supported by MetaStore. However, due to the limited expressiveness of the Dublin Core standard, MetaStore also supports a comprehensive metadata standard like METS. METS is a metadata container format comprising different sections that allow encoding of administrative ⟨amdSec⟩, structural ⟨fileSec⟩, ⟨structMap⟩, ⟨structLink⟩ descriptive ⟨dmdSec⟩ and provenance ⟨digiprovMD⟩ metadata. For example, to support harvesting of nanoscopy metadata, the nanoscopy METS profile[10] is provided.

The primary design consideration behind this approach is to keep the architecture design of MetaStore simple and avoid any dependency to an external OAI-PMH implementation. Adopting the ArangoDB Query Language (AQL) has the following benefits: (a) It provides the flexibility to define and implement precise queries required for metadata harvesting. These queries are exposed through a dedicated REST interface for enabling seamless integration with other systems. (b) With the metadata harvesting implemented on the primary metadata database, the cost and effort of maintaining and synchronizing an auxiliary OAI data provider server are avoided.

In the following, we briefly describe the metadata harvesting process. The OAI-PMH METS Harvester component retrieves the administrative, descriptive and structural metadata from the document store of ArangoDB and assembles it in the ⟨amdSec⟩, ⟨dmdSec⟩, ⟨structMap⟩, ⟨structLink⟩ section of the community specified METS profile. The retrospective provenance metadata is queried from the Apache Jena TDB and based on the vocabulary mappings shown in Table 4.1; the retrospective provenance is translated in the PREMIS standard and assembled in the ⟨digiprovMD⟩ section of the METS profile. Thus, the entire metadata stored in ArangoDB and Apache Jena TDB acts as the default OAI-PMH data provider for harvesting the complete metadata. The OAI-PMH specified six verbs are implemented as REST services[8] based on the following core AQL queries:

[10] `http://www.kitdatamanager.net/masi/nanoscopy/mets/nanoscopy-METS-profile.xml`

GetRecord: This verb retrieves a specific record from the OAI-PMH repository. The required arguments are an *identifier* associated with a record and the *metadataPrefix* specifying the metadata format to be retrieved. The query iterates over all the documents in the *OAIPMH_ Repository* and returns a single record based on the filter arguments.

```
FOR Metadata IN OAIPMH_Repository
FILTER Metadata._key==’<identifier>’ AND
Metadata.prefix==’<metadataPrefix>’
RETURN Metadata
```

Identify: This verb retrieves the administrative information describing the underlying OAI-PMH repository. The query retrieves the administrative information, *AdminInfo* from the *OAIPMH_ Repository*.

```
FOR AdminInfo IN OAIPMH_Repository FILTER
AdminInfo._key==’Identify’
RETURN AdminInfo
```

ListIdentifiers: This verb returns a list of headers, containing a unique identifier of every record. The required argument is the *metadataPrefix* specifying the metadata format. Optional arguments for selective harvesting based on *datestamp* or *set* membership are also allowed. The query iterates over all the documents in *OAIPMH_ Repository* and returns for the given *metadataPrefix* the list of header information of all the records.

```
FOR Metadata IN OAIPMH_Repository FILTER
Metadata.prefix==’<metadataPrefix>’
RETURN {"identifier":Metadata._key}
```

ListMetadataFormats: The verb is used to retrieve the metadata formats supported by the OAI-PMH repository. The query fetches the administrative information *AdminInfo* from the *OAIPMH_ Repository* to return a list of supported formats.

```
FOR AdminInfo IN OAIPMH_Repository FILTER
Admin_Data._key==’ListFormats’
RETURN AdminInfo
```

ListRecords: The verb is used to retrieve all the records from the OAI-PMH repository. The required argument for this verb is *metadataPrefix*. Optional arguments allow selective harvesting based on *set* membership or *datestamp*. The query iterates over all the documents and returns all the records.

```
FOR Metadata IN OAIPMH_Repository FILTER
Metadata.metadataPrefix==’<metadataPrefix>’
RETURN Metadata
```

ListSets: This verb is used to retrieve all the *set* structures supported by OAI-PMH repository. This command is useful for selective harvesting.

```
FOR AdminInfo IN OAIPMH_Repository FILTER
AdminInfo._key==’ListSets’
RETURN AdminInfo
```

Metadata Recovery Engine

The Metadata Recovery Engine performs the restoration of the complete metadata storage in case of a database failure. The Metadata Recovery Engine collects all the METS files from the scientific data repository, where each file is decomposed according to the various METS sections. For example, the descriptive metadata from the ⟨dmdSec⟩ section is extracted and with prior schema registration and validation inserted into the document store. The provenance metadata comprising the workflow definition in XML and the PREMIS retrospective provenance from the ⟨digiprovMD⟩ section is extracted, transformed based on mapping shown in Table 4.1 into the ProvONE model and stored in the Apache Jena TDB. The metadata recovery process is based on the combination of the processes shown in Figure 4.4 and Figure 4.6.

4.3.3 MetaStore Extension Layer

The MetaStore Extension Layer allows the integration of various third-party tools and technologies with the MetaStore Core Layer. The primary aim of the MetaStore Extension Layer is to support reuse of existing tools, software libraries, web-services, or databases in MetaStore. For example, to enable handling of annotations, the Anno4j library providing an implementation of WADM specification is integrated through the extension layer of MetaStore.

Annotation Manager (WADM)

For enriching the quality of the results and imparting additional knowledge about the data, it is necessary for domain experts to associate additional descriptions in the form of annotations with the datasets. Moreover, when the datasets evolve, there are new insights about the data, which require the annotations to be updated. For example, the *text segmentation* step of the eCodicology *Layout feature extraction* workflow (see Figure 4.15) currently implements the Trainable Weka Segmentation[11]. However, with the modification of the workflow to support projection-based segmentation, the annotations (layout features) generated for the same input dataset will be different. As these annotations are subject to frequent changes, we consider them as dynamic metadata, and to model this dynamic metadata we adopt the WADM. For enabling annotations of images and text, we integrate the Annotorious JavaScript plugin of the Annotator library, which is an open source annotation library.

Currently, the Annotorious JavaScript API provides limited functionality, as it does not support the modeling of the annotations in the W3C recommended

[11] http://imagej.net/Trainable_Weka_Segmentation

WADM. Moreover, the storage systems supported by the Annotorious JavaScript API are ElasticSearch or Parse Storage. Thus, we extend the modeling and storing of annotations by integration the Anno4j library that provides an implementation of WADM, and integrate the Apache Jena framework for persisting the annotations and allowing querying using SPARQL.

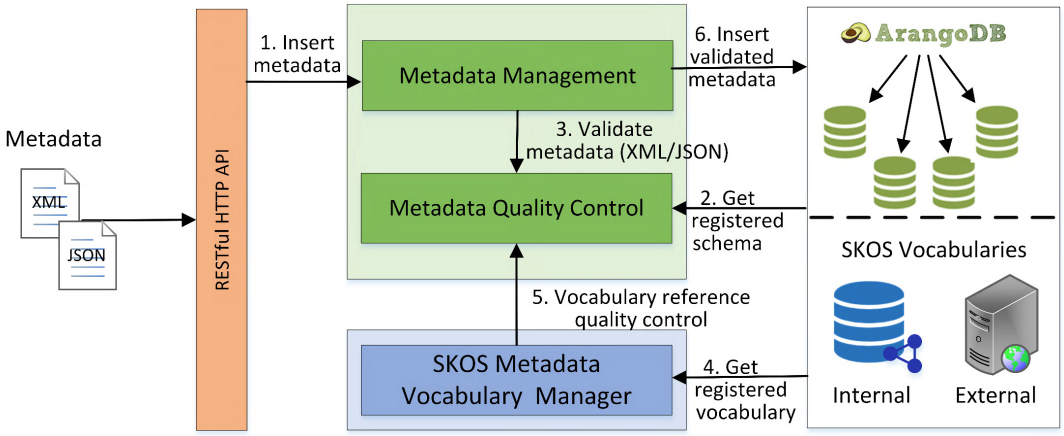


Figure 4.7: Vocabulary supported metadata quality control

SKOS Metadata Vocabulary Manager

For enabling an in-depth vocabulary-based metadata quality control, research communities can extend MetaStore with SKOS Metadata Vocabulary Manager. Currently, MetaStore offers two approaches for integrating SKOS-based controlled vocabularies. In the first approach, research communities can import their existing vocabularies in the Apache Jena TDB, which are modeled as SKOS vocabularies and maintained by MetaStore. During the metadata registration these vocabularies can be linked to their corresponding metadata schemas. This linking is necessary for enabling automated vocabulary-based quality control. The access to these imported vocabularies is provisioned through the integration of the web-based Skosmos tool [159] that offers a REST API. Skosmos exposes a generic REST API for retrieving the vocabularies. In the second approach, MetaStore allows configuring the available metadata vocabulary providers (externally available vocabularies). The REST endpoints for accessing the external vocabularies are linked to their corresponding metadata schemas for enabling vocabulary-based metadata quality control.

For example, in the Medieval Stained Glass Church Windows use case, i.e., the “Corpus Vitrearum Deutschland" project [139, 140], the controlled-vocabularies are modeled in the ICONCLASS [34] standard and exposed through the Skosmos API. For this use case, we follow the first approach described above, i.e., we have imported the ICONCLASS controlled-vocabularies from their vocabulary server into the MetaStore Apache Jena TDB. These vocabularies are linked to the different fields in their metadata schema, and during the metadata quality control process the metadata is validated against these controlled-vocabularies.

The automated vocabulary-based metadata quality control process, as shown in Figure 4.7, is an extension to the default metadata quality control process explained before. The default quality control process is extended with two additional steps that are introduced after step (3) Validate metadata (XML/JSON). Following are the additional steps: (4) For validating the metadata, each element is verified with its corresponding vocabulary. The vocabularies are retrieved either from MetaStore's Apache Jena exposing the Skosmos REST-API or from the configured external services that expose their discipline-specific vocabularies. (5) Based on the retrieved sets of vocabulary, the Metadata Quality Control component validates the metadata. Finally, as explained in step (4) of Figure 4.6, the metadata is converted to JSON format and inserted into the designated collection.

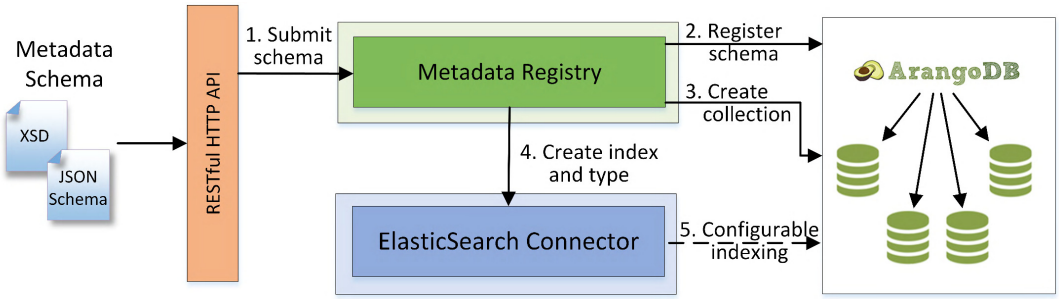


Figure 4.8: Metadata registration and ElasticSearch indexing

ElasticSearch Connector

As a core functionality, MetaStore automatically indexes the complete schema in the database for enabling full-text search over the metadata. However, for allowing a richer full-text search with the possibility of performing faceted and fuzzy search, MetaStore also provides integration with ElasticSearch. Research communities can configure MetaStore to index the complete metadata or a part of the metadata in ElasticSearch. However, enabling this configuration will change the default behavior of MetaStore, i.e., the automated indexing of metadata in the primary database (ArangoDB) will be disabled, and the full-text search queries will be redirected to the metadata indexed in ElasticSearch. Ideally, the configuration should be done during the registration of the metadata schema or the community defined METS profile. For example, in the case of a nanoscopy METS file, the descriptive metadata contains the information necessary for performing full-text search, thus, only the metadata defined in the ⟨dmdSec⟩ section of METS is indexed in ElasticSearch.

For an ElasticSearch enabled configuration of MetaStore, the metadata schema registration is followed by the index and type creation in ElasticSearch, as illustrated in Figure 4.8. The initial process of metadata schema registration is similar to the one shown in Figure 4.4. The process comprises four primary steps and one optional step. Steps (1) Submit schema, (2) Register schema, and (3) Create collection, are the same as described before. (4) For enabling full-text search based on ElasticSearch,

the corresponding index and a type for an individual registered metadata schema are created in ElasticSearch. If a METS-profile is registered, an index is created with multiple types corresponding to the different schemas embedded within METS. It should be noted that with enabling of ElasticSearch in MetaStore, the automated index creation in ArangoDB will be disabled. (5) The configurable indexing is an optional step that allows indexing of existing metadata from ArangoDB in ElasticSearch at any given point in time. This change does not affect the functioning of MetaStore, as existing metadata from ArangoDB will be indexed in ElasticSearch, and full-text searches will be redirected from ArangoDB to ElasticSearch.

RDA Metadata Directory Registry

Research Data Alliance (RDA)[12] is a research community that focuses on building social and technical infrastructures for open sharing of data. The working groups within the RDA regularly contribute recommendations and tools for handling the different aspects involved in a life-cycle of scientific data. Two such working groups in RDA are the Metadata Standards Directory Working Group (MSDWG) and the Metadata Standards Catalog Working Group (MSCWG). The MSDWG has implemented a publicly available metadata registry for collecting a wide-range of metadata standards[13] from different disciplines [14], and the MSCWG is currently working on building a metadata catalog[14] based on this metadata registry. The metadata catalog aims at providing an interface for allowing querying and retrieving of the metadata standards.

Currently, the MSDWG registry does not provide the functionalities necessary for managing metadata but contains an exhaustive list of metadata standards. Thus, integrating the MSDWG registry will leverage the metadata models currently supported by our MetaStore framework. Moreover, with the functionalities provided by MetaStore, the handling of the various metadata standards registered in the MSDWG can be completely automated. Each metadata standard in the MSDWG registry is described using a YAML [18] template, and serialized in a ".md" file. The RDA Metadata Directory Registry component in the MetaStore Extension Layer is a bridge between the MSDWG registry and the MetaStore framework. The RDA Metadata Directory Registry component retrieves the metadata standard (.md file) from the MSDWG registry and submits it to the Metadata Registry of MetaStore. Based on this ".md" file the metadata registration, indexing and code generation processes described in previous sections are executed. Thus, the RDA Metadata Directory Registry component enables a seamless integration for handling the metadata standards registered in the MSDWG registry.

[12] `https://www.rd-alliance.org/`
[13] `https://github.com/rd-alliance/metadata-directory`
[14] `https://github.com/rd-alliance/metadata-catalog-dev`

4.3.4 Scientific Data Repository

In principle, a scientific data repository is a data storage entity that offers various low-level services for storing (long-term archival), converting (format conversion), and transferring scientific data. For example, the Nanoscopy Open Reference Data Repository (NORDR) is a scientific data repository for storing experiment data. Basic administrative metadata for identifying the data is maintained by NORDR. Currently, NORDR provides many low-level services for handling the data in the repository. Following low-level services are available in NORDR: (a) automatically assigning Persistence identifier (PID) to a dataset, (b) generating preservation metadata necessary for long-term archival, and (c) implementing the different data-transfer protocols (GridFTP [2] and WebDAV [168]) for allowing transfer of data between NORDR and high-performance computing cluster. Moreover, the various data processing services necessary for composing a workflow are registered in NORDR and deployed on a high-performance computing cluster.

4.4 Evaluation

In this section, first, we present a comparison of the features of existing metadata management systems with those of MetaStore. Second, we describe the performance of MetaStore for two NoSQL database systems, followed by read and write performance evaluation of MetaStore with XMC Cat metadata catalog, MCS query interface, and the Metacat catalog system.

4.4.1 Evaluation of Features

In Section 2.2, we surveyed several metadata management systems from diverse areas of research. In this section we present a feature-based comparison of the available metadata systems with MetaStore. Tables 4.2 and 4.3 show a summary of the major features of existing metadata management systems, with a comparison to those of MetaStore.

A common architectural design limitation of the metadata management systems in SDRs is that in these systems, the database schema is designed based on a specific metadata model, with interfaces (functionality) that are tightly coupled to this model. For the community, it is an additional effort in either adapting or translating their existing metadata schema to the one supported by the SDR. Such conversions between metadata schemas often lead to a lossy transformation, where some of the schema attributes have to be either discarded or semantically modified. To avoid any schema mappings and conversions, MetaStore is designed independent of any specific metadata schema, and there are no limitations on metadata schema that MetaStore can support, as long as the metadata schema can be serialized in XML or JSON. Furthermore, as the architectural design of these systems is based on relational databases,

it inherently limits the extensibility in handling new metadata schemas. This is because, to handle a new metadata schema the relational schema needs to be modified, the services (functionality) build over this schema need to be updated, and the entire system needs to be upgraded considering the backward compatibility (i.e., the existing functionality should also be supported). In MetaStore, we overcome this design limitation by integrating a NoSQL database that allows handling of ad hoc metadata schemas. With the Metadata Code Generator component, the services necessary for handling the metadata schema are created on-the-fly and are added to the architecture without modifying the existing ones. Regarding OAI-compliance, many of the SDRs support metadata harvesting through OAI-PMH. However, the workflow provenance and annotations using interoperable standards (ProvONE and WADM) are not supported, whereas MetaStore is designed OAI-compliant and also supports the ProvONE provenance model and the WADM for provenance and annotation interoperability.

Comparing the features of the standalone metadata management systems and the Grid-based metadata systems with MetaStore, XMC Cat, Metacat, and DIMES are based on an XML database or a hybrid of XML and SQL databases. On the one hand, all XML-based metadata schemas can be supported by these systems, but on the other hand, in terms of database scalability, handling increasing metadata volumes is a major concern. In terms of metadata quality control, these systems do not implement any quality control mechanism. Moreover, integration of controlled vocabularies for verifying the metadata content with automated rectification is not supported by these systems. MetaStore offers default quality check for schema conformance and well-formedness, and advanced quality control for metadata content validation based on SKOS-based domain-specific controlled-vocabularies.

A critical deficiency among these systems is that they are not OAI-compliant, i.e., large-scale metadata harvesting through standard harvesting protocols like OAI-PMH is not supported. MetaStore is designed OAI-compliant, and the OAI-PMH data provided for allowing metadata harvesting is implemented over the primary metadata storage (ArangoDB). In terms of data reproducibility and long-term data preservation, most of these systems do not support handling of the provenance metadata. The MCS for a Grid environment provides the *Creation and transformation history* metadata attribute that allows a textual description of the data transformation process, but this is an MCS-specific attribute that is not compliant with existing provenance models (OPM, PROV, ProvONE). MetaStore supports two provenance models PREMIS and ProvONE, each for a specific purpose. With PREMIS the provenance required for long-term preservation is handled, and as ProvONE allows modeling of a workflow definition and the runtime provenance, the routine activities such as querying, analyzing and improving workflows can be performed. The MCS schema is an extension of the MCAT metadata management system that provides a special extension table (dynamic metadata) with six attribute types for modeling community-defined metadata schemas as non-hierarchical key-value pairs. However, this approach has multiple limitations: (a) hierarchical metadata schemas with embedded data structures cannot

be stored in the MCS schema. (b) the MCS schema can not handle complex attributes types such as arrays, geospatial information, and time-series data. (c) as the extension attributes are stored in a single table, it is evident and also mentioned by the authors that with increasing data volumes the query performance is expected to decline. We overcome these limitations by designing MetaStore based on a NoSQL database. This allows us to store any type of metadata model (hierarchical, flat key-value pairs) with a wide-range of data types.

Regarding the features of the commercial metadata management systems Stardog and PoolParty, both systems are primarily designed for handling enterprise knowledge data. As both systems are closed-source, enriching these systems with community driven extensions is a challenging task. On the one hand, as these systems are designed to the RDF specification, it is possible to handle dynamic metadata in the form of annotations, but, on the other hand, annotation interoperability through the W3C recommended WADM is not supported. Also, these systems do not provide large-scale metadata harvesting through the OAI-PMH specification. Stardog provides traces of database revision history using the PROV model, whereas for scientific communities it is necessary to capture the workflow description and provenance for enabling data reproducibility. Moreover, for long-term preservation of the data the support towards the appropriate metadata standard (PREMIS) is not compatible with these systems.

We also evaluated MetaStore with the Big Data processing frameworks like LOOM [39] and Apache Falcon[15]. In principal, these frameworks focus on enabling efficient data processing in a distributed environment. The LOOM framework aims at providing in-memory optimization of aggregations within a big data analysis framework. Similar to the MapReduce processing data model, LOOM provides two-phased computation. In the first phase; the individual datasets are processed, followed by consolidation of these results in the second phase. Apache Falcon is a feed and process management framework built on Hadoop that primarily abstracts and automates the redundant tasks involved in data processing pipelines. Comparing these systems to MetaStore, the current version of MetaStore focuses on managing heterogeneous metadata models with support for handling large metadata volumes. However, the LOOM and Apache Falcon frameworks are each a potential extension to the MetaStore framework when large-scale metadata processing is required.

Considering the metadata management approach based on the classification of metadata as per its use, Deelman et al. [46] propose the logical organization of metadata in different layers, in their paper, they claim that by organizing the metadata in layers, it is possible to distinguish the metadata and the source or applications for which the metadata is relevant. For example, the primary layer handles the metadata of the raw datasets, and the secondary and tertiary layers are responsible for metadata describing the process of obtaining derived data and the metadata of the derived data. Moreover, the layered metadata organization allows users to expose the appropriate granularity of detail to the user, with the possibility to track layer-based

[15]https://falcon.apache.org/

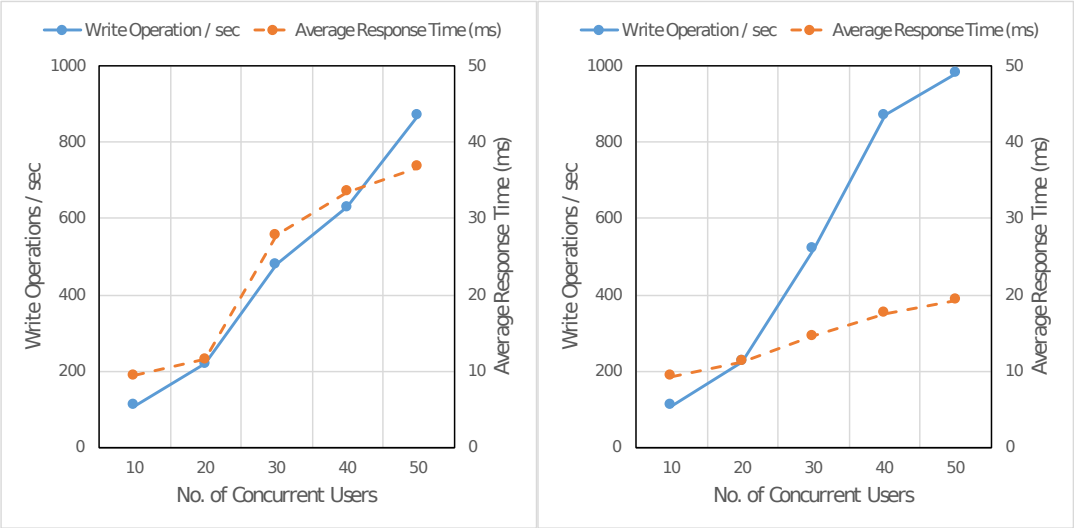
Metadata system	Schema flexibility	Supported schemas	Annotation support	Provenance support	Full-text search	OAI compliance	API	Open source	Quality control
DSpace	✗	DC, MARC, MODS	✗	✓(DC extension)	Manual indexing of DC terms	✓	✓	✓	✗
Archivematica	✗	DC, METS, PREMIS	✗	✓(PREMIS)	✓(ElasticSearch)	✓(REST)	✓	✓	✗
Digital Commons	✗	DC	✗	✗	✓(Apache Luecene)	✓	✗	✗	✗
Hydra	✗	DC, PREMIS, METS	✗	✓(PREMIS)	✓(SOLR)	✓	✓	✗	✗
ICAT Server	✗	CSMD	✗	✗	✓(Apache Luecene)	✓(SOAP & REST)	✗	✓	✗
EUDAT	Partial	DC, ISO 19115, MarcXML, CMDI, DDI	✓(B2Note)	✗	✓	✓	✓	✓	✗
EPrints	✗	DC	✗	✗	Partial (PDF, Word, HTML)	✓	✓	✓	✗
XMC Cat	✓	Any XML schema	✗	✗	✗	✗	✗	✓	✗
Metacat	✓	Any XML schema	✗	✗	Partial	✗	✗	✓	✗
DIMES	✓	Any XML schema	✗	✗	✗	✗	✗	✓	✗
SRB MCAT	✗	Proprietary model	✗	✗	Partial	✓	✗	✓	✗
MCS	✗	Proprietary model	✗	✓(Proprietary model)	Partial	✓(SOAP)	✗	✓	✗

Table 4.2: Feature comparison of several metadata management systems with MetaStore

Metadata system	**Schema flexibility**	**Supported schemas**	**Annotation support**	**Provenance support**	**Full-text search**	**OAI compliance**	**API**	**Open source**	**Quality control**
SOLE	✗	Proprietary schema	✓	✗	✗	✗	✓	✓	✗
LINDO	Partial	DC, EXIF, PDF, MXF, DICOM	✓	✗	✗	✗	✓	✓	✗
Stardog	✓	All RDF serialized schema	✗	✓(transaction provenance)	✓(Apache Luecene)	✗	✗	✗	✓
PoolParty	✓	All RDF serialized schema	✗	✗	✓(Apache Luecene, SOLR)	✗	✗	✗	✓
MetaStore	✓	**All XML and RDF schemas**	✓	✓(ProvONE)	**✓(NoSQL and ElasticSearch)**	✓	✓	✓	✓

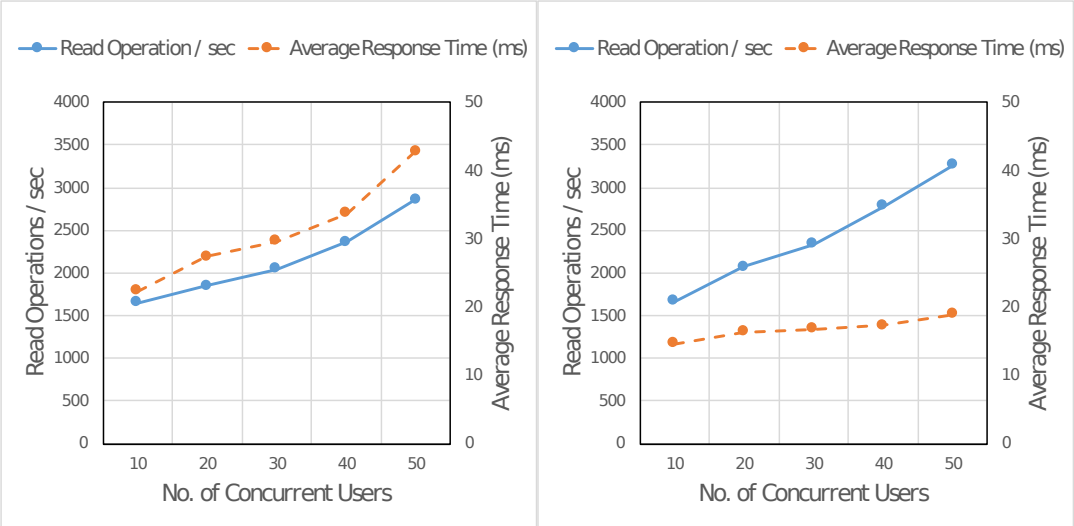
Table 4.3: Feature comparison of several metadata management systems with MetaStore (continued)

usage patterns. However, irrespective of the layers in which metadata is classified, the fundamental characteristic of metadata is the underlying model or schema to which it conforms. Thus, as MetaStore is entirely driven by the metadata schema, metadata from any layer can be handled in MetaStore. Moreover, for comprehensively aggregating heterogeneous metadata models from different layers, we adopt the METS metadata schema.



(a) One MetaStore instance (b) Two Metastore instances

Figure 4.9: MetaStore write performance (ArangoDB)

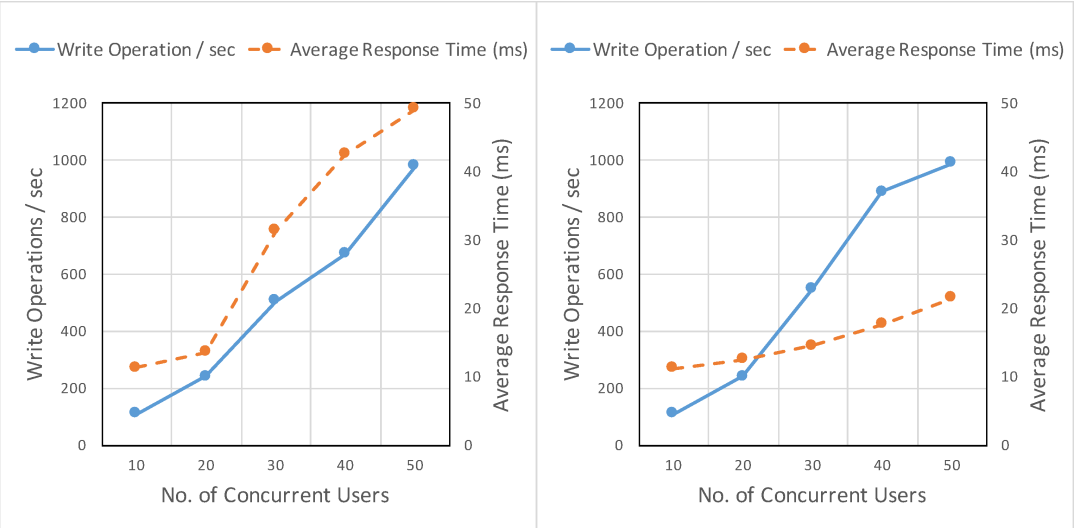


(a) One MetaStore instance (b) Two Metastore instances

Figure 4.10: MetaStore read performance (ArangoDB)

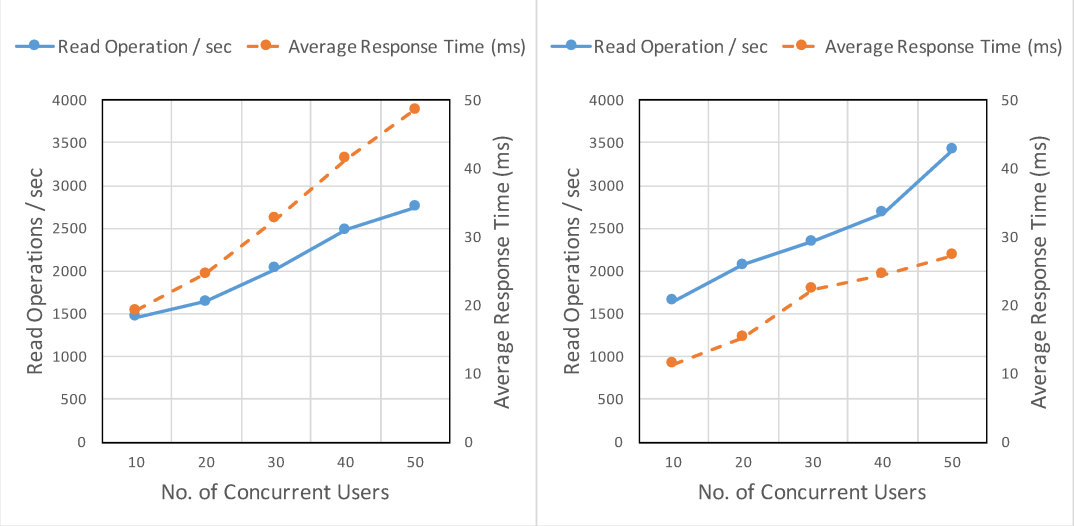
4.4.2 Performance Evaluation

In this section, we present the performance evaluation of MetaStore and compare it with existing metadata management systems. For demonstrating the extensibility



(a) One MetaStore instance (b) Two Metastore instances

Figure 4.11: MetaStore write performance (MongoDB)



(a) One MetaStore instance (b) Two Metastore instances

Figure 4.12: MetaStore read performance (MongoDB)

of MetaStore, we integrated MetaStore with both ArangoDB and MongoDB, and the read and write performance are evaluated. MetaStore is not integrated with an SQL database due to the primary requirement of having a flexible data model in modeling heterogeneous metadata models. Moreover, there are multiple studies that illustrate the advantages of NoSQL over SQL, especially considering the query read/write performance between these two kind of databases [22, 29, 97, 117, 163].

The MetaStore framework is evaluated considering a typical research community usage pattern (more read and fewer write operations) and the performance is measured in terms of the average response time for each operation. For evaluating the performance, various configurations of MetaStore were assessed. Each instance of MetaStore is deployed on a 32 GB RAM server with a 2.66GHz (8 cores) processor,

and each configuration is setup with one Apache2 load balancer on 32 GB RAM server with a 2.66GHz (8 cores) processor.

Following cluster configuration is deployed for ArangoDB. A single instance of ArangoDB coordinator server is deployed on a 128 GB RAM server with a 3.5GHz (6 cores) processor, and four Arango database servers (shards) were deployed on 16 GB RAM server with a 2.3GHz (2 cores) processor each. In the case of the MongoDB cluster, three MongoDB query routers and config servers were deployed on a 8 GB RAM server with a 2.3GHz (2 cores) processor. For storing the data, four MongoDB data nodes (shards) were deployed on 16 GB servers with a 2.3GHz (2 cores) processor each.

For efficiently handling large numbers of concurrent requests, we used the Apache server with the *mod_jk* module for load balancing the MetaStore instances deployed on independent tomcat servers. The `maxThreads` parameter in the *AJP 1.3* configuration for each tomcat server is increased to 1024 for handling a larger number of concurrent requests. We also optimized the load balancer to handle a higher number of requests per second by enabling the Apache Multi-Processing Module (MPM)[16] worker. It is possible to further tune the load balancer performance by increasing the `maxRequestWorkers` parameter's default value of 256, but an inappropriately large value might bring the server to a standstill due to an influx of requests. Thus, for a production environment we set this parameter to its default value. Each test is executed three times. Due to negligible variations in each result, the error bars are not shown, and only the average results are depicted. Figures 4.9a to 4.12b show the write and read results.

Write Performance: Figures 4.9 to 4.12 illustrate the results of the write performance for one and two instances of MetaStore integrated with AranogDB and MongoDB, respectively. Each write operation carries a payload ranging between 20-30 KB and follows the process described in Figure 4.6. For a single instance of MetaStore deployed on ArangoDB and MongoDB, we noticed that with increasing concurrent users, each generating multiple write operations, the average response time increased drastically, and some write operations failed to complete. This increase in average response time is due to the concurrent request handling limitation of each tomcat server. Moreover, each write operation comprises the quality control process, where an additional request is sent to retrieve the corresponding schema, followed by well-formedness and schema conformance checks.

With the introduction of a second instance of MetaStore either deployed on ArangoDB or MongoDB, we observe that due to the systematic distribution of the write operations between the two MetaStore instances, the average response time is retained below 22ms, and not a single write operation failed.

For comparing the write performance of MetaStore with the MCS query interface, first we need to understand how metadata is stored in the MCS systems. Each add operation in the MCS system creates a logical file with multiple user-defined attributes

[16] `https://httpd.apache.org/docs/2.4/mpm.html`

in the database (maximum of ten attributes). The add operation is implemented as a native Java API and as a web service interface. Using the native Java API connected to a MySQL database, a maximum of 360 writes per second is achieved with 25 concurrent threads. For the web interface, a maximum of 70 add operations is possible for 25 concurrent threads. For multiple clients with four concurrent threads each, approximately 380 add operations is achieved, whereas for the web interface, a maximum of 80 add operations is achieved.

The LEAD infrastructure's XMC Cat metadata catalog based on hybrid XML or relational approach is evaluated with a native XML database, Oracle's Berkley DB XML[17]. Jensen et al. [86] argue that the size of a metadata file or the metadata content of an experiment affect the insert time, and the hybrid XML/relational approach performs better than an XML database. The insert performance for a scaled workload ranged between 80ms and 120ms.

MetaStore exposes all its functionalities including the metadata insert operation through REST services. Comparing the write performance of MetaStore with the MCS web interface and the XMC Cat metadata catalog, it is clear that MetaStore is easily able to handle more than 800 requests with 50 concurrent users (threads) per second, and the performance improved with the addition of a second MetaStore instance.

Read Performance: Figures 4.10a to 4.12b show the results of the read performance. The number of read operations on the MetaStore framework are expected to be higher than the write operations and hence the reads/sec factor is approximately five times more than write/sec. As the documents in ArangoDB and MongoDB are fully indexed, all read operations are arbitrary full-text searches.

In the case of Metacat, the XML tree structure represented by a Document Object Model (DOM) [173] is decomposed into nodes, where the root node is the document entity and the children nodes are elements and attributes. These nodes are stored in a relational database schema, for allowing querying of the metadata at different depths. Nested SQL queries are defined with a specific depth or full-text search queries are defined with depth zero. For evaluating the read performance, the full-text query is executed on a node size of 2.03×10^6 and the nested query on a node size of 23,000/25,000 with a depth equal to five. For full-text search queries, it took 30sec when using indexes and 29sec when using a nested query. For the queries using indexes with a node depth varying between one and six, the query execution time ranged between 1sec and 50sec, whereas for SQL nested queries, the execution time is between 1sec and 10sec.

The MCS query interface provides simple queries that do a value match for a single static attribute in the file and complex queries that do value matching for all the attributes in the file. For simple queries with a single client executing multiple concurrent threads on a native Java API, a peak of approximately 2300 queries per second is reached, whereas for the web interface approximately 120 queries per second

[17] `https://www.oracle.com/database/berkeley-db/xml.html`

is achieved. For the complex queries, approximately 560 queries per second is achieved and for the web interface, approximately 100 queries per seconds were possible.

Compared to these results the full-text read performance of MetaStore (considering both ArangoDB and MongoDB) varied between 19ms to 49ms for 10 to 50 concurrent users with 1650 reads/sec to 3200 reads/sec, thus, showing significantly better performance compared to that of Metacat, the LEAD XMC Cat metadata catalog, or the MCS query interface.

Analyzing the performance comparison between MetaStore and existing metadata management systems, it necessary to understand how metadata is modeled, stored and queried in these systems. Following are the reasons why the performance of MetaStore is better than that of existing systems: (1) In MetaStore the XML containing the metadata is never decomposed into atomic entities and stored in a relational database but transformed to JSON and stored as a JSON file in the NoSQL database. The JSON-based storage of metadata in NoSQL databases avoids the overhead of transforming XML either as a CLOB or shredding it using the inlining approach. (2) As the metadata schema is fully indexed, the I/O intensive nested queries for retrieving specific paths from the XML structure are not required. (3) The overhead of reconstructing XML documents from the relational schema is completely avoided because the queries return JSON objects that are serialized to XML based on the registered metadata schema. (4) In the case of NoSQL databases, the write operations have a better throughput because they are primarily performed in the main memory, instead of writing to the disk, as in the case of an SQL based database. For example, in ArangoDB the memory-mapped files are used to handle write operations and are regularly synced to disk via the `fsync` command.

4.5 Application Use Cases

In this section, we briefly explain three use cases (one from bio-medical research and two from digital humanities) for which MetaStore is adopted.

Use case 1: Nanoscopy

For illustrating the applicability of MetaStore in managing heterogeneous metadata models, first, we consider the light super-resolution microscopy (nanoscopy) use case. Nanoscopy is a novel imaging technique that aims at bridging the resolution gap between conventional light microscopy and electron microscopy [35]. A typical nanoscopy investigation begins with the acquisition of raw image datasets in HDF5, KDF, or TIFF format from a high-resolution microscope. A complete series of measurements for a given specimen can easily amount to 100-150 TB in size. For handling such large volumes of data, NORDR offers efficient handling of massive data volumes collected from various geolocated high-resolution microscopes, backed by a large-scale data storage and high-performance computing cluster [123]. Various nanoscopy scientific workflows are executed for processing these raw datasets. These workflows

(a) An angel with bagpipes.
Photo: Andrea Gössel, CVMA Germany/Freiburg, CC BY-NC 4.0, Link: http://id.corpusvitrearum.de/images/4486.html

(b) The holy family.
Photo: Andrea Gössel, CVMA Germany / Freiburg, CC BY-NC 4.0 Link: http://id.corpusvitrearum.de/images/3431.html

Figure 4.13: Digitized medieval stained glasses.

are described using the Business Process Execution Language (BPEL) and the administrative and descriptive metadata is described based on the community-specified metadata models[18].

For automating the metadata extraction from raw datasets, we implemented a staging processor that extracts the metadata from HDF5 and TIFF file during the ingestion of the raw datasets in NORDR. This extracted metadata is submitted to MetaStore via the REST interface, where it is verified with the registered localization microscopy metadata schema, and stored in the document store of ArangoDB. Using the Prov2ONE algorithm, the workflow defined in BPEL is translated into the ProvONE prospective provenance, which is enriched with the retrospective provenance collected during the workflow execution. The entire ProvONE graph is serialized in RDF and stored in Apache Jena. As the functionality (services) for handling the extracted metadata is automatically created during the schema registration stage, we do not need to create any additional services in MetaStore explicitly. Moreover, this extracted metadata is wrapped in the METS format that complies to a registered METS-profile and exposed to the researcher communities for metadata harvesting through OAI-PMH.

Use case 2: Corpus Vitrearum Deutschland

The Corpus Vitrearum Deutschland (CVD) is a part of the Corpus Vitrearum Medii Aevi (CVMA), which is long-term international research project in digital humanities. The CVMA aims at digitizing, cataloging and analyzing the medieval stained glasses

[18] `http://datamanager.kit.edu/masi/localizationmicroscopy/2016-03/LocalizationMicroscopy.xsd`

Figure 4.14: Digitized parchment page of manuscript 1108 from city library and city archives Trier, dating to 15th century

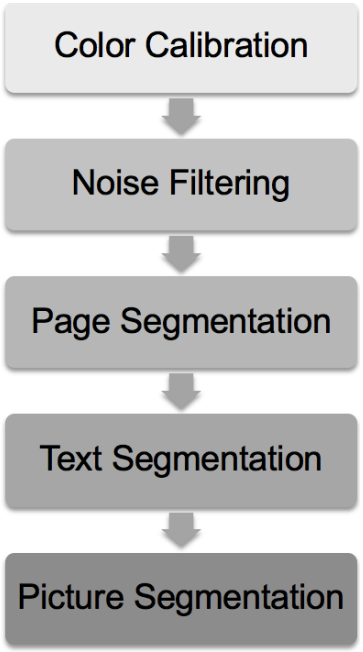


Figure 4.15: Layout feature extraction workflow

that are preserved in churches, museums, and galleries all over Europe. Currently, CVD research community has digitized and published 5086 images of stained glasses, out of which two examples are shown in Figure 4.13. On the one hand, the CVD image repository is implemented for storing and accessing these images, but on the contrary discovery and analysis of the images based on metadata is lacking. Each image is embedded with an extensive set of metadata attributes that conform to the XMP metadata standard. Most of these attributes are filled during the digitization process, while few attributes that describe the painted figures or depicted scenes are referred from a predefined ICONCLASS vocabulary.

Similar to the nanoscopy use case, the first step is to extract the metadata that is embedded in the image (TIFF). For this, we reused the TIFF metadata extractor staging processor of the nanoscopy use case. The extracted metadata is submitted to MetaStore through the REST interface, where the metadata is validated before inserting in the document store of ArangoDB. For automatically verifying the correctness of the terms describing the stained glasses, the corresponding ICONCLASS vocabulary is linked to the XMP metadata schema during the metadata registration stage.

Use case 3: eCodicology

The eCodicology is a research project in digital humanities that aims at designing, evaluating and optimizing algorithms for identifying the layout features of medieval

manuscripts [31]. First, the manuscripts from the "Virtual Scriptorium St.Matthias" were digitized, and a total of 170,000 images were acquired. An example image is shown in Figure 4.14. These images are stored in CodiStore, which is a scientific data repository supporting long-term archival and reuse of digitized medieval manuscripts [32]. Currently, CodiStore is not capable of handling the metadata embedded within the datasets (static metadata) and the metadata generated during the workflow execution (dynamic metadata). Hence, the entire MetaStore is integrated with CodiStore.

In the following, we describe our experience and the challenges faced during the integration of MetaStore with CodiStore. During the digitization process, each manuscript (each containing few hundreds of pages) is enriched with bibliographical metadata describing the following attributes: *Signature, Century, Material, Format, Leaf Count, Library* [32]. For systematically structuring the metadata for an entire manuscript, the eCodicology community has adopted the Text Encoding Initiative (TEI) format that is further embedded in a METS file, which conforms to an eCodicology METS-profile. This METS file also contains the administrative metadata, structural map, and structural links that describes the hierarchy of the pages within a manuscript and the logical and physical links between the pages. As the metadata in TEI and METS is not subject to change, we consider it as static metadata. Similar to the previously use cases, during the ingest of data (digitized manuscripts) in CodiStore data repository, we implemented a staging processor for extracting TEI metadata. The extracted metadata is inserted in the document store of ArangoDB with prior quality control.

The handling of dynamic metadata that is generated during execution of workflow is a challenging task. For each digitized page the Layout Feature Extraction workflow (see Figure 4.15) is executed. Similar to the nanoscopy workflow, the provenance is captured in ProvONE and stored in Apache Jena. The workflow consist of five steps; *Color Calibration, Noise Filtering, Page Segmentation, Text Segmentation, and Picture Segmentation*. Each step in the workflow generates metadata that describes the output generated by a particular step. For example, the *Color Calibration* generates a color calibration matrix, the output of *Page Segmentation* is page space measurements (page area, height, width, inclination angle, and left-corner co-ordinates), and the *Text Segmentation* generates written space measurements (main text area, height, width and inclination angle) and number of lines. This workflow-specific metadata is subject to frequent changes because modifying the workflow parameters will yield different output, or changing the algorithms will generate entirely different results. Hence, to handle such dynamic metadata, we had to extend MetaStore with an annotation framework that supports modeling of ad hoc metadata schemas conforming to the W3C recommended WADM. For this, we integrated the Anno4j library with an RDF triple store (Apache Jena).

4.6 Discussion

In this section, we explain the various architecture design decisions undertaken in the implementation of the MetaStore framework. As a core design principle, the MetaStore framework is designed based on the modular architecture design pattern. Adopting this design pattern has a multitude of advantages. First, each feature of MetaStore is developed as an independent component (high cohesion) with no inter-component dependencies (low coupling). This allows component-specific updating of features with easy maintenance of the entire framework. For example, initially, the Metadata Quality Control component offered only basic schema validation and well-formedness checks. However, with modular architecture design of MetaStore, it is easy to extend this component to support domain-specific vocabulary metadata content verification. Second, it enables the seamless integration of new features (components) while reusing the existing ones. For example, the integration of an RDF database and the support for building and reusing SKOS-based domain-specific vocabulary is seamlessly appended to the core functionality of MetaStore. Moreover, the same RDF database is used to store and query the ProvONE provenance graphs. Third, customizable integration of MetaStore with existing systems kept the complexity of the overall architecture to a minimum. For example, for the nanoscopy use case, as there is no dynamic metadata either from the workflows or the annotations, only the MetaStore core layer is integrated with NORDR. Thus, the modular design of MetaStore allowed integration of selective features from MetaStore with NORDR. However, for handling diverse metadata in eCodicology research, i.e., static metadata encoded in TEI format, dynamic annotations in the WADM, and workflow provenance in ProvONE, the entire MetaStore framework is integrated with CodiStore.

With MetaStore architecture design based on NoSQL database systems, different types of metadata, such as descriptive, administrative, provenance and structural metadata can be efficiently modeled to maximize the utilization of the data models offered by the ArangoDB and Apache Jena. This decision not only vastly reduced the complexity of the architecture (technology footprint) but also the repetitive software development effort in redesigning, implementing and updating of a relational database schema is avoided. Moreover, with the adoption of existing open-source solutions, the total cost of ownership is minimal. For example, workflow and provenance metadata comprising complex relationships are modeled in the ProvONE provenance model and serialized as RDF triples for allowing execution of graph traversal queries. Descriptive, administrative and structural metadata that can be serialized in XML is modeled in the document data model for enabling full-text search and complex analytical queries. Regarding database scalability in handling increasing data volumes, with ArangoDB it is possible to shard large volumes of metadata in multiple database instances.

The MetaStore offers different research communities an entirely automated metadata management system that is capable of handling both community-specific metadata schemas as well as existing metadata standards. The Metadata Code Generator

component automatically generates the necessary code (services) for handling the registered metadata schema and on-the-fly extends the functionality of MetaStore. Hence, significantly reducing the resources and efforts of scientific communities in writing and maintaining the software (services) that are necessary for implementing the metadata schema. For generating the code, we use the JAXB-XJC[19] library, and at runtime, the Metadata Management Component determines the appropriate metadata schema that needs to be created using Java Reflections API [56].

For enabling large-scale metadata harvesting in MetaStore, it is necessary to implement the six verbs of the OAI-PMH specification. Initially, our approach is to integrate the jOAI web application[20] that provides an implementation of the OAI-PMH specification. However, it involves setting up an additional data provider server containing the metadata in XML format that conforms to a set of rules described by the jOAI web application. This would have been an unnecessary overhead, for not only maintaining an additional metadata file-server just for metadata harvesting purpose but also in transforming the metadata stored in ArangoDB to the jOAI-compliant format. Instead, we implemented the six OAI-PMH verbs as AQL queries in MetaStore, allowing better query performance as compared to native file I/O operations in the case of jOAI. Hence, our architectural design approach for implementing the OAI-PMH specification within MetaStore and directly across different NoSQL databases significantly reduced our cost and effort in using and maintaining additional metadata harvesting tools and infrastructures.

With the availability of numerous metadata schemas, each specific for a given application, it is evident that research communities will adopt multiple schemas simultaneously to describe their data. For example, the nanoscopy research community adopted the Core Scientific Metadata (CSMD) model for describing the administrative metadata, a community-specific metadata schema for the descriptive metadata, and the PREMIS and ProvONE for the provenance metadata. On the one hand, supporting heterogeneous metadata schemas is realized by MetaStore, but on the contrary, the aggregate modeling of metadata for allowing comprehensive metadata harvesting required for metadata publishing, or during data and metadata migration is a challenging aspect. For this, we adopted the METS schema and provided a default METS-profile that allows us to aggregate heterogeneous metadata schemas. However, as METS supports only PREMIS for modeling provenance, it is necessary to define the SKOS-based vocabulary mapping between PREMIS and ProvONE to enable provenance interoperability.

Currently, existing workflow management systems and the provenance interoperability frameworks are based on the PROV model or the OPM [6, 24, 111]. On the one hand, the OPM and PROV allow provenance interoperability through a standard provenance model, but on the other hand, a core limitation of these models is that they are capable of handling only the retrospective provenance and not the

[19] `https://jaxb.java.net/2.2.4/docs/xjc.html`
[20] https://uc.dls.ucar.edu/joai/

prospective provenance. In scientific research where complex workflows are designed, executed, repeated, and continuously evolved, the prospective provenance is of critical importance. Thus, for handling the entire provenance trace for scientific workflows in a single provenance model, we adopted the ProvONE provenance model. With the availability of workflow provenance in ProvONE, it is possible to design queries that encompass the analysis of the workflow results along with the corresponding workflow definition. In the case of communities where the research is still growing, the availability of provenance in ProvONE is useful in avoiding redundant execution of obsolete workflows, with the possibility to rapidly evolve their workflow to generate better results. Moreover, for long-term sustainability and adoption of standard querying language (SPARQL), the ProvONE graphs are serialized in the RDF model.

Based on the software architecture design pattern adopted for implementing MetaStore, and the generic adaptability of MetaStore for diverse use cases, in the following, we summarize the benefits of MetaStore: (a) The modular architecture design pattern implicitly provided a clear separation of concerns, allowing us to realize the requirements as separate modules in the architecture. Moreover, with low inter-module coupling, the maintenance and upgrading of the components are easy. (b) With the integration of open-source solutions and the automatic adaptive architecture design, the overall total cost of ownership is minimal. Adopting the dynamic composition design pattern further eliminated the need in following the routine software development life cycle. (c) By exposing all the features through a well-defined REST interface, a seamless integration of MetaStore with existing systems is possible. For example, NORDR, CodiStore and CVD image repository were easily integrated with MetaStore. (d) Conforming to existing metadata standards in handling provenance and annotation metadata, and with an implementation of the widely accepted OAI-PMH metadata harvesting specification, we guarantee the long-term sustainability and reuse of the MetaStore framework. (e) The flexible data model of NoSQL and RDF databases enabled us not only to handle ad hoc metadata models but also provided the inherent sharding capabilities for managing increasing metadata volumes. Moreover, with the adoption of these databases, it is possible to efficiently store and query heterogeneous metadata schemas in the appropriate data model.

4.7 Summary

In this chapter, we presented MetaStore, a novel metadata management framework for handling heterogeneous metadata models. In principle, our aim is to design a generic metadata management system that can be reused by arbitrary research communities using either a standard metadata model or a community-specific metadata model.

The main idea of MetaStore is based on two aspects. First, we realized from the literature that existing metadata management systems are limited in the functionality they provide and cannot be adopted by research communities with varying metadata models. Second, for generic applicability of MetaStore, we collected and grouped

the requirements put forth by three independent research communities. Thus, for enabling reuse of MetaStore by arbitrary research communities and for realizing the requirements as independent functionalities, the core architecture of MetaStore is based on the principle of modular design. On the one hand, the modular design enabled the requirements to be implemented as task specific components, but on the other hand, the architecture design is still static, i.e., for handling new metadata schemas, the required functionality has to be manually implemented in MetaStore. To overcome this static nature of MetaStore, the modular design of MetaStore is enhanced with the principle of Compositional Adaption. The new functionality to handle a newly registered metadata schema in MetaStore is dynamically generated at runtime using the dynamic composition design pattern. Following is a summary of the functionalities offered by the MetaStore framework:

- a metadata schema registry for validating metadata, with an extension to integrate discipline-specific SKOS vocabularies for allowing automated metadata quality control,
- on-the-fly generation, compilation and deployment of the entire framework for creating the services necessary for handling the registered metadata schema (zero downtime upgrade of MetaStore),
- automated index creation in a NoSQL database for enabling full-text search, and integration of ElasticSearch for allowing execution of complex analytical queries, faceted search, and fuzzy search,
- providing an annotation framework for enabling researchers to add descriptive information (dynamic metadata) in the form of annotations, with support to WADM for enabling annotation interoperability,
- an OAI-PMH metadata harvester implemented on a NoSQL data provider for enabling metadata harvesting (metadata sharing).

To show the generic applicability of MetaStore, the framework is integrated into NORDR, CodiStore, and CVD image repository. For evaluating the MetaStore framework, first, we presented a feature based comparison of MetaStore with existing metadata management systems. We observe that MetaStore not only exhaustively covers the necessary functionalities but also adheres to recommended standards for long-term architecture design sustainability and metadata interoperability. In Table 4.2 and 4.3, we provide a comparison of features available in existing metadata management system to those of the MetaStore framework. Second, we presented the performance-based evaluation, where we highlighted the benefits of designing MetaStore on a NoSQL database over traditional SQL or XML databases. Additionally, for evaluating the scalability of MetaStore, various configurations of MetaStore have been assessed with different workloads, and it is observed that the performance of MetaStore is significantly better compared to existing metadata systems.

In this chapter, we briefly mentioned the importance of provenance for long-term data preservation, which is critical for scientific data repositories. For enabling automated provenance interoperability, the mapping rules between the PREMIS and the ProvONE model were established using the SKOS mapping vocabulary specification. However, in the next chapter, we extend the topic of provenance interoperability for enabling analysis of heterogeneous provenance traces from different WfMSs.

Chapter 5

Provenance Management in WfMSs

In the previous chapter, we described the importance of metadata in scientific research. For efficiently handling heterogeneous metadata models, we explained that it is necessary to design a comprehensive metadata management framework that is independent of any given metadata model. We proposed the MetaStore framework as a long-term sustainable and reusable framework. With the availability of MetaStore, we have established a metadata management platform that can help extend our research in managing provenance metadata (information) in WfMSs. Moreover, for enriching the capabilities of SDRs in handling provenance information, it is necessary to handle provenance during the preservation of digital data. Hence, as a minimum functionality, the translation rules between the PREMIS and ProvONE models were established in the previous chapter. Building on those efforts, we extend our research in provenance interoperability for WfMSs. Even though provenance in WfMSs is a well-established topic, there have not been many efforts undertaken in the handling of provenance using a comprehensive model like ProvONE. In addition, literature does not contain a single approach that addresses the handling of provenance for a BPEL-based WfMS in a provenance model. The first step in enabling provenance interoperability is the translation of heterogeneous provenance traces in a common model. For this, we present a novel method for translating provenance from BPEL-based WfMS to ProvONE. In the second step, we apply this method to other WfMSs to design a complete provenance interoperability framework that provides functionality for collecting, translating, storing and analysis of heterogeneous provenance information.

5.1 Motivation and Objectives

Research in provenance is being pursued in various domains such as Operating Systems (OS), Semantic Web, WfMSs, and standalone provenance management systems [58, 178]. This chapter focuses on provenance interoperability methods that have been developed for WfMSs [41]. Most existing WfMSs are designed considering a provenance model that captures only retrospective provenance, except for a few systems [57, 100] that are capable of capturing both prospective and retrospective provenance.

The latter WfMSs are based on autonomous provenance models and storage strategies that are closely tied to the underlying storage system [42].

Currently, existing WfMSs are designed based on proprietary provenance schemas [145] that are not compatible with standard provenance models such as PROV [112], Open Provenance Model (OPM) [114], or ProvONE [37]. Moreover, the workflow definition (prospective provenance) is serialized in an XML file that adheres to the specification established by the WfMS. In addition, the associated retrospective provenance collected during a workflow run is modeled in the proprietary provenance schema and stored in logs, files, or relational databases [58, 145]. As the XML-based workflow definition is decoupled from its associated retrospective provenance, querying and analysis of comprehensive provenance information is not possible. Furthermore, additional limitations of decoupled handling of prospective and retrospective provenance are as follows:

- reproducibility of scientific results for validation of outcomes in a different execution environment is not possible due to the absence of workflow definition,
- tracking and analysis of workflow evolution are not feasible,
- comparing heterogeneous workflow definitions and their associated provenance traces are not possible,
- sharing workflow definitions with their associated runtime provenance among different WfMSs is a complicated and laborious task.

To address these challenges, research in provenance interoperability has bifurcated into the following directions: (a) few of the WfMSs have introduced extensions for mapping their system-specific provenance models to either the PROV model[1] or OPM [59, 121, 172], (b) various frameworks based on OPM are introduced for enabling interoperability of heterogeneous provenance models [6, 24, 122]. However, these systems based on either OPM or PROV only guarantee retrospective provenance and not prospective provenance interoperability.

Considering these limitations, Oliveira et al. [115] introduced a provenance interoperability framework that uses Prolog to store and query provenance from e-Science Central [166] and SciCumulus [44] WfMSs to ProvONE. Similarly, Lim et al. [98] presented a provenance collection framework that extends OPM with prospective provenance, with a realization based on a relational database.

Although these solutions are similar to the approach presented in this chapter, there are limitations associated with these frameworks.

First, these frameworks do not employ W3C standard RDF data model, with the W3C recommended SPARQL querying capabilities. Due to this limitation, the long term sustainability, and widespread adoptability, i.e., reusability of these frameworks for other WfMSs cannot be guaranteed.

[1] https://github.com/taverna/taverna-prov

Second, as these approaches are based on the mappings between the proprietary provenance schema implemented by the frameworks and the ProvONE model, the granularity of translations cannot be determined. Moreover, translation of the execution order defined in the native workflow to the ProvONE model is not explicitly addressed by these approaches.

Third, the provenance schema implemented by these frameworks is designed only to handle data-flow languages. Thus, the extensibility of these frameworks in handling control-flow languages like BPEL that comprise of multiple control-flow constructs is not feasible.

The goal of this chapter is to address shortcomings of existing provenance interoperability frameworks, with respect to: (a) long-term sustainability and reusability of the solution and (b) support for both control-flow and data-flow workflow languages with an automated translation of the execution-order and data-flow structure from the original workflow. For this, we first explain the difference between data-flow and control-flow workflow languages. Second, we analyze the design patterns of a control-flow language to design the Prov2ONE algorithm for translating these patterns to ProvONE prospective provenance. To further automate the entire translation process, we propose the WfMS-specific adapters that collect, translate and enrich ProvONE prospective graphs with retrospective provenance. Moreover, for efficient modeling of dense network of complex relationships observed in provenance graphs, the proposed framework provides native support for RDF data model. Furthermore, this allows us to reuse the already established PROV ontology that provides a mapping of PROV data model in the RDF namespace. The decision behind choosing an RDF data model is further strengthened by several studies that describe advantages of using an RDF triple store over traditional SQL databases [43, 163].

5.2 Preliminaries

The notions and concepts surrounding workflow patterns and provenance models necessary for realizing a comprehensive provenance interoperability framework are presented in this section. The ProvONE model is split into two parts to allow the modeling of both prospective and retrospective provenance. For handling each part systematically, we first describe the classes and associations of the ProvONE model and the relation between these parts. Second, we explain the difference between control-flow and data-flow workflow languages, in reference to three languages namely BPEL, SCUFL, and MoML. Finally, we describe the various patterns in a control-flow language with respect to BPEL specification. These patterns are necessary for verifying the completeness of the translations from BPEL specification into ProvONE.

5.2.1 ProvONE model

ProvONE is an extension of the PROV model that is widely accepted in the international provenance awareness community. ProvONE is specifically designed

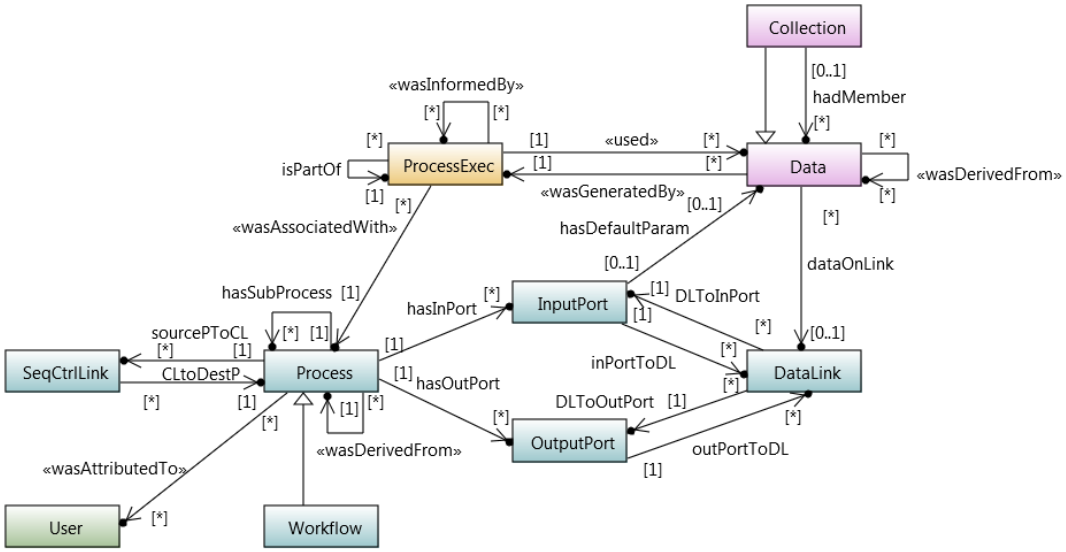


Figure 5.1: ProvONE conceptual model UML class diagram [37]

to capture both prospective and retrospective provenance in the same model. The conceptual UML diagram of ProvONE is shown in Figure 5.1. In principle, ProvONE is an abstract model that is not bound to any specific vocabulary. For modeling provenance with different levels of granularity, it is possible to extend each class and association with existing metadata vocabularies. Below, we explain prospective classes and associations available in the ProvONE model, followed by retrospective classes and associations.

ProvONE prospective provenance

Class: Process. The Process class represents a data processing task or an activity that is to be performed. The task can either be atomic or composite, typically representing a nested workflow. Each Process class is associated with at least one InputPort and OutputPort for consuming and producing the data. Using external metadata vocabulary terms the detailed attributes for a Process can be described. For example, a unique identifier for each process is specified using the dcterms:identifier term from the DC namespace, and the name of a process can be set using dcterms:title.

Class: Workflow. The Workflow class is a specialized Process and is used to represent a complete experiment in its entirety. The prov:wasDerivedFrom term is used to represent workflow evolution for handling multiple versions of a workflow.

Class: User. The User class is derived from the prov:Agent super-class that represents a person, an organization, or a software agent. Using the User class, the ownership and the accountability of a Process or a Workflow can be captured.

Class: InputPort. The InputPort is necessary for consuming data produced by other Processes, from an external source or as default parameters. The details about an input such as variable name, data type and value can be described in details using dcterms, xsd, rdf, or rdfs namespaces.

Class: OutputPort. The OutputPort class attached to a Process represents the outcome generated after execution of a process. Similar to the InputPort class, the detailed description of output can be described using the existing namespaces.

Class: DataLink. It is necessary to allow Processes to share data for a workflow to execute successfully. For this, the DataLink class enables data to be sent between processes, wherein the OutputPort of a process is connected to an InputPort in another process. The DataLink class provides a one-to-many and a many-to-one cardinality between InputPort to DataLink and DataLink to OutputPort respectively for connecting multiple source and destination ports. Moreover, the DataLink also allows connection between InputPorts and OutputPorts of nested processes.

Class: SeqCtrlLink. The SeqCtrlLink represents the execution flow between processes. The source process connected through the SeqCtrlLink has to be finished for a destination process to start. In case of a destination process with multiple SeqCtrlLinks, it is necessary for all the source processes to terminate, i.e., the SeqCtrlLink can be used to infer execution order defined in the original workflow.

Association: hasSubProcess. The hasSubProcess association is used to capture the relationship between parent and child processes. This association represents nested composition of processes in ProvONE that are defined as nested workflow in the original specification.

Association: sourcePToCL. The sourcePToCL association links a source process to its SeqCtrlLink. Through a sourcePToCL association, termination or completion of a source process is propagated for initiating the execution of subsequent processes.

Association: CLtoDestP. To complete the connection between a source and a target process, CLtoDestP connects the SeqCtrlLink to its subsequent Process. Through the CLtoDestP association, the subsequent process is initiated for execution.

Association: hasInPort. The connection between InputPorts and a Process is specified by the hasInPort association.

Association: hasOutPort. The connection between OutputPorts and a Process is specified by the hasOutPort association.

Association: hasDefaultParam. For specifying default input Data to a Process, the hasDefaultParam is directly linked between a Data class and an InputPort. The hasDefaultParam association allows workflows with special configurations to be defined.

Association: outPortToDL. For establishing data sharing among Processes, the OutputPort of a Process needs to be connected to a DataLink class. For this, the association outPortToDL is used.

Association: DLToInPort. For indicating that the output of a process via a DataLink is consumed by another process, the DLToInPort association between the corresponding DataLink and an InputPort needs to be specified.

Association: inPortToDL. For representing data sharing between the OutputPort and InputPort of independent processes, the DLToInPort and the outPortToDL associations are used. However, for representing sharing of data between nested processes,

wherein data is passed via the InputPorts, the inPortToDL association is used.
Association: DLToOutPort. Similar to inPortToDL, for exposing the output generated by a nested process through the OutputPort of its parent process, the DLToOutPort association is used. The DLToOutPort connects the OutputPort of a nested process to the OutputPort of its parent process via DataLink.
Association: wasAttributedTo. The wasAttributedTo is an association adopted from the PROV vocabulary. It is used to relate a Process to a User that is responsible for its creation.
Association: wasDerivedFrom. The wasDerviedFrom associations is adopted from the PROV vocabulary for representing the evolution of a Process or a Workflow.

ProvONE retrospective provenance
Class: ProcessExec. For a Process defined in a workflow, the ProcessExec class represents its runtime execution. If a Process represents an entire workflow, then the ProcessExec represents the entire execution trace. There can be multiple ProcessExec classes for a given Process class. Similar to other ProvONE classes, the details of an execution can be described using the PROV, DC, or WfMS-specific vocabularies.
Class: Data. The Data class represents the basic unit of information that is either consumed or produced by a Process. For example, if a process produces images as output, then using appropriate vocabularies the Data class will describe the details of these images such as location of the image in a repository, size and format of the image, a PID that uniquely identifies this image, and/or a brief description about the image.
Class: Collection. The Collection class represents a generalization of the Data class. With the Collection class, similar Data items can be organized together. For example, in eCodicology, the digitized images of a manuscript are structured using the Collection class, where each image represents a Data entity, and the entire manuscript is a Collection.
Association: wasAssociatedWith. For connecting the prospective provenance Process class with its execution details in the retrospective provenance ProcessExec class, the wasAssociatedWith associations is used.
Association:dataOnLink. The dataOnLink association connects execution trace details by connecting the Data class in retrospective provenance to the DataLink prospective provenance. It effectively specifies the Data item that was shared between processes.
Association:used. The used association represents the relationship between a ProcessExec and Data it consumed during its execution.
Association:wasGeneratedBy. Contrary to the used association, the wasGeneratedBy represents the Data that was produced by a ProcessExec class.
Association:wasInformedBy. The equivalent representation of data sharing between processes via the SeqCtrlLink and the output-input ports is the wasInformedBy association. The wasInformedBy captures the execution order of processes. However,

for a deterministic workflow, this association can also be derived from the prospective provenance, i.e., either from the SeqCtrlLink or from the DataLink between Processes.
Association:isPartOf. The isPartOf association enables to specify a structure to the ProcessExec instance. A parent ProcessExec representing a workflow has child ProcessExec representing either individual processes or nested workflows.
Association:wasDerivedFrom. The wasDerivedFrom represents a relation between data structures that are produced during the execution of a workflow. The wasDerivedFrom is useful for independently tracking the data derivation beginning from the raw input data to the result.
Association:hasMember. To represent the relation between Data items and a Collection, the hasMember association is used.

5.2.2 Control driven vs. Data driven workflow languages

Workflow languages are primarily classified as control-driven, control-flow languages or data-driven, data-flow languages, while few workflows are classified as hybrid-workflows that adopt a combination of both control-flow and data-flow. The prime difference between these languages is how the activities are invoked. In case of control-flow languages, an explicit connection representing the transfer of control needs to be specified between activities. On the other hand, in data-flow workflows, the activities are invoked with the availability of data on their input ports.
Control-flow languages. The control-flow languages have originated from the scripting language, wherein applications are systematically chained together using a shell script to construct a larger complex application. Similarly, control-driven workflow languages are based on a specification that provides a set of control structures for assembling activities that are to be executed in a pre-determined order. Not only, do these languages allow defining a control-flow among activities, but also a data-flow between activities can be defined.

Typically, a control-flow language offers more than just sequential or concurrent execution of activities. Referring to the BPEL specification, for branching the execution of workflow based on some condition, these languages provide *switch, if-else, and pick* control structures. In case of repeated execution of a sub-section of a workflow, looping control structures such as *while, foreach, and repeat-until* are available. These control-structure enable workflow designers to define workflow with maximal granularity. Moreover, control-flow languages like BPEL offer the possibility to handle runtime exceptions using *faultHandlers* control structure safely.

Considering BPEL as concrete example of a control-flow language, we explain the characteristics of a control-flow language. BPEL is the most widely adopted workflow language for business workflows as well for designing scientific workflows for Grid infrastructures. BPEL allows description of both behavioral service interface and executable service processes. Considering the focus of this chapter, i.e., the translation of execution order of workflow defined in BPEL to ProvONE, we consider the executable

service processes in BPEL. In principle, the execution order of activities defines (structured activities) partners involved during communication (partnerlinks), and messages exchanged between processes. In BPEL, there are two types of activities, Primitive activities and Structure activities.

Primitive activities. The primitive activities are to perform the actual operations in a BPEL workflow, i.e., they provide the structure to communicate with external partner services defined in the WSDL. The important primitive activities are invoke, receive, and reply. The invoke activity is used to call an external web-service specified by the partner link, the receive activity is a channel for sending data to a workflow from an external partner service, while the reply activity is used to send a response to an external partner service. For systematically organizing the primitive activities, in order to accomplish a given task, the primitive activities need to be declared within structure activities.

Structure activities. Each structure activity available in BPEL imposes an execution constraint over the primitive activities defined within it. This restriction is necessary for designing efficient workflows that require a precise execution order. Structure activities can be nested within each other. However, it should be noted that the constraint of a parent activity is exercised on the immediate children activities. For example, multiple independent sequence activities nested within a flow activity follow the constraint of parallel execution enforced by the flow activity and all sequence activities are triggered for concurrent execution. However, the invoke activities within each sequence activity are executed sequentially. The following structure activities are available in BPEL: (a) sequence activity is the control structure for sequential execution of tasks, (b) flow activity enables parallel execution of tasks, (c) switch activity is used for conditional routing, (d) pick activity allows execution of activities based on events that are received from an external source, or time-based execution of activities based on timeout duration, and (e) while activity is a control structure for looping over activities that need to be repeated.

Data-flow languages. In a data-flow language, a graph data model is used to represent a workflow, where the nodes represent activities, and the edges represent transfer of data between the activities. The incoming edges to an activity describe the input to an activity, and the outgoing edges represent an output of activity. As the core principle of any data-driven workflow language is the execution of activities based on the availability of data on their input ports, most of the data-flow languages do not offer any control structures for implicitly defining the execution-order of activities. Thus, compared to control-flow languages the data-flow languages are simple.

Most workflow languages in the domain of scientific WfMSs are based on the principle of data flow between the input-output ports of the processes [67, 100, 172]. The prominent and the widely adopted languages are the SCUFL and the MoML language.

SCUFL is principally a data-flow language with few constructs to allow explicit

ordering of tasks. A task in SCUFL is described using the ⟨processor⟩ construct with input and output ports for consuming and producing the data. For declaring the data dependencies among processors, the ⟨datalink⟩ construct is used. The ⟨datalink⟩ contains a list of relations that define the connection between the input and output ports. The Taverna WfMS implement a greedy scheduling technique that executes processes as soon as data is available on their input ports. The creation of new execution thread is limited by the number of threads defined by the scheduler.

In the case of Kepler WfMS, the workflows are described in MoML specification. The workflow tasks are represented by actors that have input and output ports to share data between them. However, for orchestrating the execution of actors in a workflow, Kepler provides the notion of directors whereby each director in Kepler implements a specific Model of Computation (MoC) [66]. The notion of tokens determines the execution of an actor. When an actor receives a token, it is executed a given number of times, and as it executes new tokens are fired with resulting data sent to the output port. Kepler WfMS primarily supports four core directors.

- Synchronize Data Flow (SDF): For SDF directed workflow, there is fixed number of tokens consumed and produced per firing. An actor is invoked as soon as data is available on the input ports. All actors within a workflow have to declare their token production before execution. The SDF director is suitable for modeling linear or sequentially executing workflows.

- Process Network (PN): In a PN directed workflow, each actor is given an execution thread. The actors in PN modeled workflow are loosely coupled and are executed immediately after availability of data on an actor's input port. The token firing schedule of the actors is not calculated statically in a PN directed workflow. This director is suitable for managing workflows that require parallel processing on distributed systems.

- Continuous Time (CT): In a CT directed workflow, each token is affixed with a time stamp to perform system simulations. This director keeps track of time for each iteration and also the time between each iteration. The workflow is iterated for a given number of times to reach the desired stop condition.

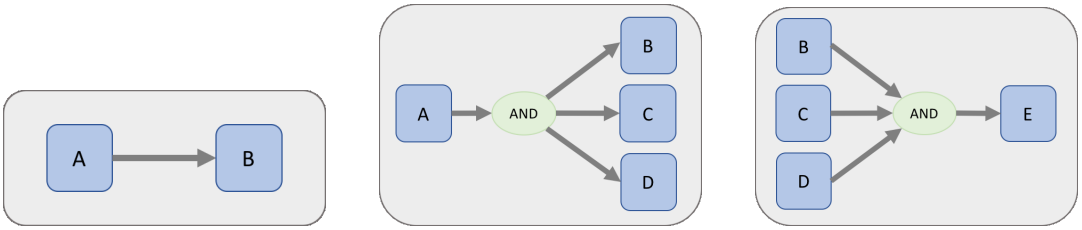
- Discrete Events (DE): The DE director uses event tokens that consist of a data token and time stamp with auxiliary information that helps the director determine the execution order of an event. These events are organized as a temporal queue on a global time scale. There are two ways in which a DE modeled workflow terminates: (a) the stop time of the earliest event is reached, or (b) all the temporally organized events on the global timescale are completed.

Based on the director specified in a workflow, the execution order of actors is determined at runtime by the Kepler WfMS. However, the definition of a workflow is independent of a director, i.e., the description of actors and their relation (potential

data dependencies) is specified during the design of the workflow, using the input and output ports.

5.2.3 Patterns in Control-flow languages

Control-flow languages such as BPEL are rich with constructs that allow designing of complex workflow patterns. An in-depth analysis of workflow design patterns for control-flow workflow languages is described by Russel et al. [133]. In this section, we present the design patterns that are supported by the BPEL specification [171]. These patterns are necessary to evaluate the completeness of the Prov2ONE algorithm in translating BPEL workflows into their equivalent representation in ProvONE prospective provenance.



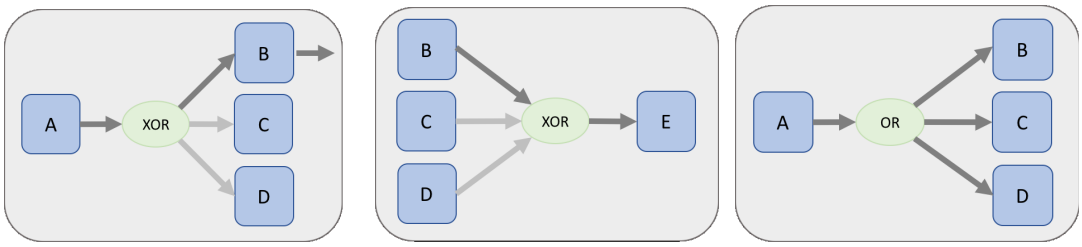
(a) Sequence (b) Parallel Split (c) Synchronization

Figure 5.2: Control-flow language patterns

Sequence: The Sequence pattern is the simplest pattern in a control-flow language. A series of tasks that are to be executed consequently are defined using the Sequence pattern. Figure 5.2a shows a sequence pattern with two tasks A and B that are connected via an edge denoting that task B will start after completion of task A. In BPEL, the ⟨sequence⟩ control structure is used to define a series of tasks that are to be executed in sequence. A ⟨sequence⟩ activity can contain primitive activities like ⟨invoke⟩, ⟨receive⟩, or ⟨reply⟩ or other structure activities.

Parallel Split: The Parallel Split pattern also called as AND-split allows for concurrent execution of multiple tasks. In a Parallel Split pattern, the thread of execution is divided into two or more branches that are executed in parallel that may or may not merge or re-synchronize, i.e., each branch may complete independent. An example of a Parallel Split pattern with three tasks that are to be concurrently executed is shown in Figure 5.2b. In BPEL specification the ⟨flow⟩ control structure is used to design a Parallel Split pattern.

Synchronization: The Synchronization pattern also called as an AND-join (see Figure 5.2c) represents the merging of parallel branches. The branches that were created by a Parallel Split control structure are merged after their completion. Figure 5.2c shows the merging of Parallel Split with three tasks. In a BPEL workflow, this is represented by the completion of a ⟨flow⟩ control structure.



(a) Exclusive Choice (b) Simple Merge (c) Multi-Choice

Figure 5.3: Control-flow language patterns

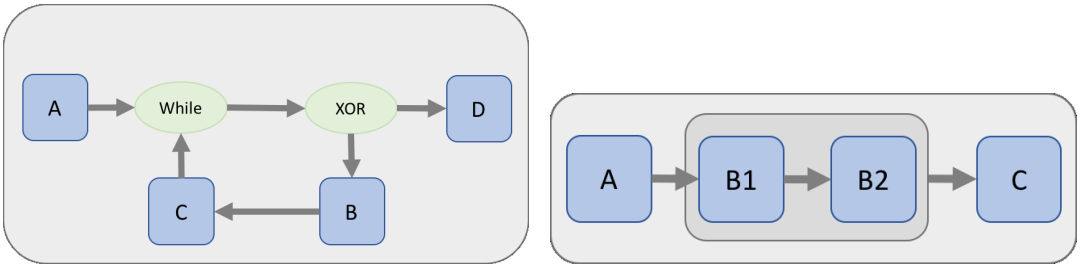
Exclusive Choice: The Exclusive Choice pattern, also knows as an XOR-join, the execution control is transferred to only a single branch, i.e., for a workflow containing two or more parallel branches, the Exclusive Choice pattern allows the execution of a specific branch based on a decision that occurs at the runtime. For example, Figure 5.3a shows a pictorial representation of the Exclusive Choice pattern, the path beginning with task B will be executed. In BPEL, this pattern is realized using a ⟨switch⟩ or ⟨flow⟩ with links control structure.

Simple Merge: Similar to the Synchronization pattern, the Simple Merge pattern allows the merging of parallel branches without the need for synchronizing the branches. The Simple Merge is also called as an XOR-join and is shown in Figure 5.3b. The Simple Merge in BPEL is represented with the completion of ⟨switch⟩ or an interlinked ⟨flow⟩ control structure.

Multi-Choice: The Multi-Choice pattern, also known as the OR-split pattern is shown in Figure 5.3c. The Multi-Choice pattern is similar to the Parallel Split pattern. However, instead of executing all the subsequent branches as in the case of Parallel Split, in a Multi-Choice design, either a single branch or multiple branches are executed. The decision of selection of the branches is made at run-time. In BPEL, this pattern is realized using an interlinked ⟨flow⟩ construct structure.

Structured Loop: For executing a task or a subset of a workflow repeatedly, the Structured Loop pattern is used. The loop pattern has a single point of entry and exit, with a precondition or a postcondition to determine the continuation of the loop. In BPEL, the Structured Loop is realized using the ⟨while⟩ control structure. An example of Structured Loop pattern is shown in Figure 5.4a.

Implicit Termination: The Implicit Termination pattern states that a workflow or a sub-workflow should terminate when no tasks are remaining. This pattern is used for determining the completion of a workflow or a sub-workflow. In BPEL, the ⟨flow⟩ provides an implicit termination of each branch, whereas for other control structures it is necessary to assign a ⟨terminate⟩ activity. The design pattern illustration for Implicit Termination is same as the Parallel Split pattern, shown in Figure 5.2b.



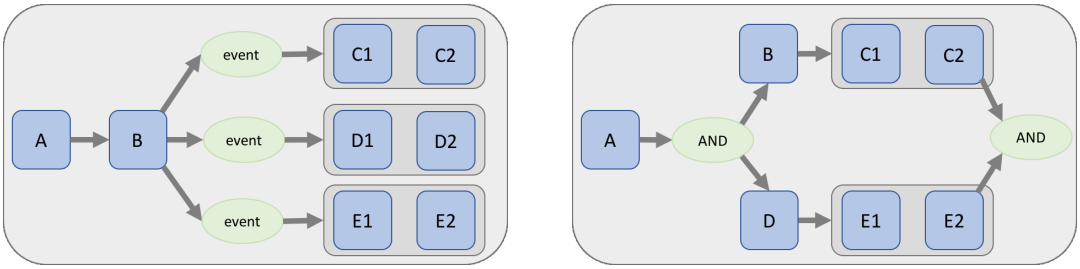
(a) Structured loop (b) Multiple instances

Figure 5.4: Control-flow language patterns

Multiple Instances with a Priori Design-Time Knowledge: For workflows designed with this pattern, the required number of instances of tasks are known at design time. The Multiple Instances pattern with a prior design-time knowledge of two predefined tasks B1 and B2 are shown in Figure 5.4b. The instances of these tasks are independent of each other and are executed simultaneously. For the subsequent tasks to be invoked, it is necessary to synchronize the completion of the task instances. In BPEL, this pattern is realized by the ⟨flowN⟩ or the ⟨foreach⟩ control structure.

Multiple Instances with a Priori Run-Time Knowledge: This pattern is similar to the Multiple Instances with a Priori Design-Time Knowledge pattern. The only difference is that the decision of creating the number of instance of a task or a set of tasks is deferred to the latest possible time during the workflow execution. This pattern is useful when runtime factors such as data and resource availability, inter-process communication, or external input from a user are to be considered. In BPEL, this pattern is realized by the ⟨flowN⟩ or the ⟨foreach⟩ control structure.

Multiple Instances without Synchronization: For creating multiple instances of a task within the same workflow, the Multiple Instances without Synchronization pattern is used. The instances of a task are independent of each other and do not need to be synchronized after completion. In BPEL, this pattern is realized by specifying a ⟨invoke⟩ activity in a ⟨while⟩ activity.



(a) Deferred Choice (b) Critical Section

Figure 5.5: Control-flow language patterns

Deferred Choice: The Deferred Choice pattern allows the execution of one or more branches of a workflow based on a predetermined condition. In a workflow with a Deferred Choice pattern, all the possible future paths of execution are predefined, and during the execution time, based on an external factor the appropriate branch or branches are executed. These factors can be external messages, timeout events, resource availability, or incoming messages. In a BPEL workflow, this pattern is realized using the ⟨pick⟩ control structure. Figure 5.5a shows an example of a Deferred Choice pattern.

Persistent Trigger: The Persistence Trigger is similar to the Deferred Choice pattern, wherein a task is triggered by a different process or from an external service. Typically, a task or a set of tasks are executed based on a message that is received from an external source. This message is persistent until the appropriate task can handle it. In BPEL, this pattern is realized using the ⟨pick⟩ structure activity with a set of ⟨onMessage⟩ activities. Each ⟨onMessage⟩ activity has a variable attribute that handles a specific type of message and based on the message the corresponding subsection of the workflow is executed.

Critical Section: The Critical Section pattern shown in Figure 5.5b allows execution of a part of a workflow in isolation. Tasks C1 and C2, and E1 and E2 are part of independent critical sections. This pattern is required when tasks in a given section require exclusive access to shared resources for their completion. Moreover, if there are multiple critical sections in a workflow, then it is necessary for one critical section to complete for the other one to commence. In BPEL, this pattern is supported by serializable scope with the attribute `variableAccessSerializable` is set to `yes`.

Interleaved Routing: The Interleave Routing pattern is similar to the Critical Section pattern, wherein each task in a section of workflow has to be executed only once. The order in which these tasks are executed is not important. However, all the tasks in a section have to be completed before subsequent tasks are executed. In BPEL, this pattern can be realized by embedding a ⟨scope⟩ within the context of a ⟨pick⟩ control structure.

From the various patterns supported by the BPEL control-flow language, it is clear that the structural design of these patterns follows either a sequential organization of activities or a parallel organization of activities. The aim of the Prov2ONE translation algorithm explained in Section 5.3.2, is to translate the workflow execution order and the data flow structure defined in a BPEL-workflow into its equivalent representation in ProvONE.

5.2.4 Provenance management in WfMSs

Considering the goal of this chapter, which is the provenance interoperability between heterogeneous WfMS. First, it is necessary to understand the provenance management capabilities of existing WfMSs. For this, we consider two different data-driven WfMSs, namely the Taverna and Kepler WfMSs and explain their provenance management architectures. In the case of control-driven WfMSs, we consider the ApacheODE WfMS. However, provenance management in a BPEL-based WfMS is a novel research topic, and literature yields no previous work on this subject. Hence, we present our first contribution in the research area of provenance handling in BPEL WfMS [125].

Provenance management in Taverna WfMS

Modeling of provenance in Taverna is based on domain aware Janus provenance model, which is an extension of the Provenir ontology. In principle, the Janus model has two parts: (a) domain-agnostic part that models the same information stored in the Taverna provenance model and (b) domain-aware part that extends the ontology to include properties and classes specific to given domain. Currently, Taverna does not support handling of workflow definition and evolution. During the execution of a workflow, runtime events from the Taverna WfMS are persisted in a Taverna-specific relational schema[2]. However, for enabling provenance interoperability, Taverna was extended with the PROV plugin to export the provenance in PROV model. For analyzing the provenance traces, SPARQL queries can be executed over the exported PROV graphs.

Provenance management in Kepler WfMS

The Kepler WfMS is enhanced with the Provenance add-on module (Provenance Framework) that provides the recording of workflow execution history [5]. This module can be further extended with with the Reporting and Workflow Run Manager modules. The overarching architecture design goal was to implement a plugin-based framework that includes an interface for configuring data models, metadata formats, and cache destinations. The separation of concern design principle modeled for the Kepler directors is also applied for the Provenance Framework. Moreover, to customize the capturing of provenance at user required level of granularity, different configuration parameters are offered. Kepler also captures the evolution of a workflow definition, i.e., the changes to a given workflow definition are also captured by the system. This allows tracking of workflow design, the corresponding executions and the parameters that yielded the required results.

The collection of complete provenance for a workflow is performed in two steps. First, the workflow definition in MoML is converted into to Kepler's internal provenance format. Second, for collecting the information generated during a workflow

[2]http://dev.mygrid.org.uk/wiki/display/developer/Provenance+schema+in+2.2.0

execution, several event listeners are implemented in Kepler. Each event listener is registered for specific concern, when a given concern occurs, the corresponding event listener is notified to take the appropriate action. For controlling the granularity of the provenance captured by the framework, event listeners can be registered or unregistered from the framework.

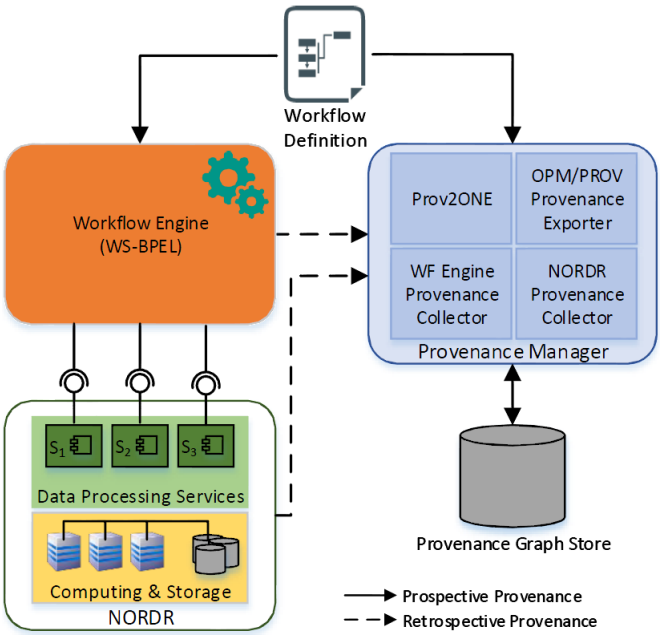


Figure 5.6: Architecture of Provenance Management in SDR [125]

Provenance management in BPEL WfMS

From literature, it is evident that BPEL-based WfMSs have widely been adopted for orchestrating complex workflows over distributed Grid infrastructure [52, 54, 105, 138, 151, 165]. Typically, *in silico* scientific experiments are not a single step process but involve multiple steps that need to be systematically coordinated to produce the desired results. The first step towards execution of workflows on a Grid infrastructure is the integration of a WfMS with the scientific data repository architecture. For this, we chose a BPEL-based WfMS and integrated it into the scientific data repository. The provenance management architecture of a BPEL-based WfMS that is integrated with a scientific data repository is explained below.

As stated before, comprehensive provenance consists of two parts, the workflow definition, and the workflow execution information. A common aspect among BPEL-based WfMSs is that the workflows are defined using the standard BPEL-specification. Depending on the architecture design, each WfMS implements the necessary functionality to capture and expose the runtime events of a workflow execution. For example, the start and end time of a process and the input and output data consumed or generated by a process.

As a prototype implementation to substantiate our concept, we chose the ApacheODE WfMS. The ApacheODE exposes the retrospective provenance in the

form of events through the dedicated Management API. The Management API provides the execution details of each process and the data shared between processes. The integration architecture the Provenance Manager with a BPEL-based workflow engine and a scientific data repository is shown in Figure 5.6. The description of the prospective and retrospective translation rules between BPEL, ApacheODE, and ProvONE are described in Section 5.3. The Provenance Manager is responsible for collecting, translating and generating the ProvONE provenance graphs. This module comprises four sub-modules. The explanation of each sub-module is as follows:

- Prov2ONE: For translating the workflows from BPEL to ProvONE prospective provenance, the Prov2ONE algorithm is implemented as a REST service that is used to submit a workflow archive containing a BPEL workflow. The ProvONE graphs are serialized in RDF data model and stored in Apache Jena.
- WF Engine Provenance Collector: To complete the creation of the ProvONE graphs, the Workflow (WF) Engine Provenance Collector is responsible for polling the events generated by the ApacheODE WfMS and translating them into ProvONE and appending them to the ProvONE prospective graph.
- NORDR Provenance Collector: For enriching the ProvONE prospective graphs with the audit trail, the NORDR provenance collector propagates these audit trail generated by the NORDR services and appends them to the ProvONE prospective graph.
- OPM/PROV Provenance Exporter: For sharing retrospective provenance interoperability, the OPM/PROV Provenance Exporter sub-module exports only the retrospective provenance in PROV ontology.

This section explained the provenance capturing capabilities of existing WfMS as well as our first contribution for enabling BPEL-based WfMS with provenance handling functionality. In the next section, we present a provenance interoperability framework that allows sharing, querying, and analyzing of both prospective and retrospective provenance from different WfMSs.

5.3 Provenance Interoperability Architecture

In this section we present the overall architecture of the P-PIF (Figure 5.7), and briefly describe the components that are necessary for collecting, modeling, storing, and querying the provenance from various WfMSs. Currently, ApacheODE, Taverna, and Kepler WfMSs are supported by the P-PIF architecture.

Provenance Adapters: Adapters are the bridge between WfMSs and the P-PIF. Each adapter collects the necessary retrospective provenance from the respective WfMS and submits it to the Workflow System Provenance Collector sub-module through the exposed REST-API, where the retrospective provenance is appended to the ProvONE prospective provenance.

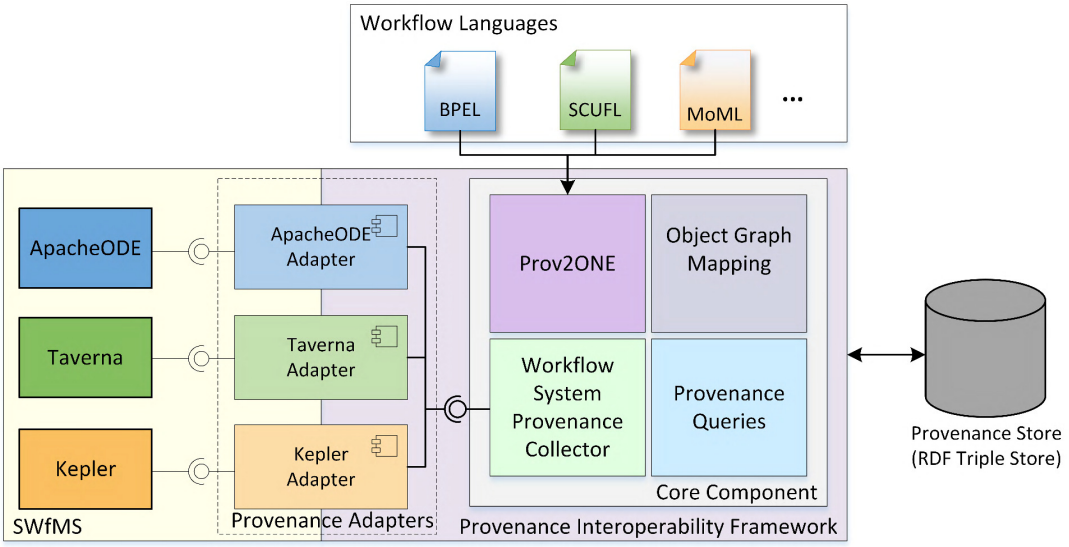


Figure 5.7: Architecture of Provenance Interoperability Framework (P-PIF)

To capture retrospective provenance, the ApacheODE adapter integrates the Management API[3] provided by ApacheODE WfMS. The Taverna adapter integrates two sources that trace retrospective provenance: (a) a REST-API[4] that provides runtime events of a workflow run, and (b) the Taverna-PROV[5] plugin that exports a provenance trace as a PROV-O RDF graph. The Kepler adapter integrates the Provenance Query API[6] exposed by the workflow system to retrieve the retrospective provenance as PROV traces in JSON format.

Core Component: The core component module comprises four sub-modules that collectively build the functionality for automating the handling of provenance information from heterogeneous WfMSs.

- Prov2ONE: A Java implementation of the Prov2ONE algorithms is implemented in this sub-module. For each execution of a workflow defined in BPEL, MoML, or SCUFL, an RDF specification complying ProvONE prospective graph is created in the Apache Jena TDB. Details of the Prov2ONE algorithms are presented in section 5.3.2.

- Workflow System Provenance Collector: This component holds the implementation of the mapping rules necessary for translating the retrospective provenance submitted by each provenance adapter to the ProvONE retrospective model. This sub-module exposes a REST-API[10] for allowing the provenance adapters to submit the retrospective provenance. The mapping between ProvONE retrospective model and the runtime provenance traces from WfMSs is described in Section 5.3.4.

[3]http://ode.apache.org/management-api.html
[4]http://dev.mygrid.org.uk/wiki/display/tav250/REST+API
[5]https://github.com/taverna/taverna-prov
[6]https://code.kepler-project.org/code/kepler/trunk/modules/provenance/docs/provenance.pdf

– Object Graph Mapping: This sub-module allows integration of ad hoc database systems or RDF triple stores with the P-PIF. Currently, Apache Jena TDB library that allows storing the ProvONE graphs as RDF triple and querying using the SPARQL is supported through this sub-module.

– Provenance Queries: The provenance retrieval queries are implemented using the query language supported by the database system and are exposed as REST services. These REST services allow easy integration in a graphical web interface for intuitive visualization of the provenance graphs. The queries from the first provenance challenge with the six additional queries that are specifically designed considering the ProvONE model are introduced in Section 5.4.2. The queries are implemented using the SPARQL query language and exposed as REST services.

Table 5.1: BPEL to ProvONE mapping rules

#	BPEL:Activity	ProvONE:Class	ProvONE:Association
1	process, scope	workflow	wasDerivedFrom
2	invoke, receive, reply, onMessage	process	—
3	invoke.inputvariable, receive.variable	inputPort	hasInPort
4	invoke.outputvariable, reply.variable	outputPort	hasOutPort
5	assign.copy.from	datalink	outPortToDL
6	assign.copy.to	datalink	DLToInPort
7	sequence, flow, pick	seqctrllink	sourcePToCL, CLToDestP

5.3.1 Vocabulary mapping rules between ProvONE and scientific workflow specifications

In this section, we present the mapping rules between the workflow specifications and the prospective classes and associations of the ProvONE model. Typically, as the workflow specifications do not change frequently, the mapping rules need to be defined only once per workflow language.

BPEL and ProvONE mapping. BPEL is built on top of XML and uses the Web Service Definition Language (WSDL) for specifying web services for external communication, making BPEL the prime candidate for defining workflows that are to be executed on a Grid environment [54]. A BPEL workflow starts with the *process* activity that holds the name and the namespace of the process, and a *variables* element that holds the data (input and output) used by the web services during the execution. The BPEL activities considered for constructing the ProvONE prospective graph are

the structure and the primitive activities. The fault-handling activities that occur during the execution of a workflow are handled as retrospective provenance.

Structure Activities: Structure activities enclose primitive activities to define a workflow composition. Structure activities are: (1) *sequence*: a sequence activity defines sequential execution of operations; (2) *flow*: a flow activity defines parallel execution of operations; (3) *pick*: a pick activity performs a specific operation when a given event occurs; (4) *scope*: a scope activity allows to define a nested workflow.

Primitive Activities: Primitive activities are used to execute tasks such as: (1) *invoke*: the invoke activity is used to call a web service; (2) *receive*: the receive activity is used to accept an input; (3) *reply*: the reply activity is used to send a response.

The mapping rules between BPEL activities and ProvONE prospective classes and associations are described in Table 5.1. The details of the rules are as follows:

Rule 1: A BPEL *process* or a *scope* activity is used to define a workflow or a nested workflow respectively. Similarly, in ProvONE the *workflow* class represents a workflow in its entirety. For *process* or *scope* activity in BPEL, the corresponding representation in ProvONE is the *workflow* class with the *wasDerivedFrom* association.

Rule 2: The BPEL primitive activities represent an execution task (i.e., it is used to perform an actual task in a workflow). Similarly, the ProvONE *process* class represents a computational task. For each primitive activity, the corresponding ProvONE representation is the *process* class. Even conditional activities such as if-else and while are also represented as an individual *process* class. Thus, allowing tracing (querying) of the conditions that led to the execution of a specific path in a workflow.

Rule 3 and 4: In BPEL, data is passed to the *invoke* and *receive* activities through the *inputvariable* and the *variable* attributes. The definition of the variables is extracted from WSDL or XSD and modeled in ProvONE *inputPort* class with the association *hasInPort*. Similarly, for the *outputvariable* and the *variable* attribute for *invoke* and *reply* activities the corresponding ProvONE representation is the *outputPort* with the association *hasOutPort*.

Rule 5 and 6: In BPEL data is shared using the *copy* element of the *assign* activity. For each *assign* activity, the corresponding ProvONE representation is the *datalink* class with associations *outPortToDL* and *DLToInPort*.

Rule 7: The execution order of the primitive activities in BPEL is governed by the structural activities such as *sequence*, *flow*, and *pick*. Based on the structural activity the corresponding sequential execution or parallel execution structure among the ProvONE processes is represented using *seqctrllink* with the associations *sourcePToCL* and *CLToDestP*.

SCUFL and ProvONE mapping. Workflows in Taverna are defined using the SCUFL specification. SCUFL is a data-flow centric workflow language that consists of a set of processes identified by the *processor* element. A *processor* element encloses an *activity* element that determines the execution type of the *processor*. For example, an *activity* can be a *dataflow* indicating a nested-workflow, *beanshell* indicating a

Table 5.2: SCUFL to ProvONE mapping rules

#	SCUFL:Element	ProvONE:Class	ProvONE:Association
1	dataflow	workflow	wasDerivedFrom
2	processor	process	—
3	inputPorts	inputPort	hasInPort
4	outputPorts	outputPort	hasOutPort
5	datalink.source.port	datalink	outPortToDL, DLToOutPort
6	datalink.sink.port	datalink	DLToInPort, inPortToDL
7	datalink.source, datalink.sink	seqctrllink	sourcePToCL, CLToDestP

shell script, or *rest* indicating an HTTP REST service. For receiving and sending data between processors, each processor is associated with a set of *input* and *output* ports, and *datalinks* that allow passing of data between processors. The mapping rules between SCUFL elements and ProvONE prospective classes and associations are described in Table 5.2, and the explanation of each rule follows below.

Rule 1: The topmost element that defines a workflow or a nested workflow is defined by a *dataflow* element in SCUFL. For each *dataflow* element, the corresponding ProvONE representation is the *workflow* class with the *wasDerivedFrom* association.

Rule 2: Each computational task in SCUFL is defined in the *processor* element. Thus, each *processor* element is modeled as a ProvONE *process* class.

Rule 3 and 4: In SCUFL the input and output of a *processor* are defined in the *inputPorts* and *outputPorts* elements. Thus, for the ports defined for a *processor*, the corresponding representations in ProvONE are the *inputPorts* and *outputPorts* of a *process*, with the associations *hasInPort* or *hasOutPort*.

Rule 5, 6 and 7: Data sharing between *processors* in SCUFL is specified through the pair *datalink.source.port* and *datalink.sink.port*. For each such source-sink pair, the equivalent representation in ProvONE is the *datalink* class. Moreover, the execution order is determined based on the sharing of data between ports, and is mapped to the *seqctrllink* class to capture and maintain the same execution order in ProvONE.

MoML and ProvONE mapping. Workflows in Kepler are defined using the MoML specification [100]. A MoML workflow is based on the principle of actor and director. MoML actors are either simple or composite and are declared in the *entity* element. A simple actor is an atomic processing unit that is responsible for performing a task. A composite actor allows multiple simple actors to be bundled together to define a nested workflow. Actors have *input* and *output* ports for receiving and sending data. The data sharing among *actors* is realized through the *link* element using the reference of a *relation* element. The mapping rules between MoML elements and ProvONE prospective classes and associations are listed in Table 5.3, and the explanation of each rule follows below.

Table 5.3: MoML to ProvONE mapping rules

#	MoML:Element	ProvONE:Class	ProvONE:Association
1	compositeActor	workflow	wasDerivedFrom
2	actor	process	—
3	port.input	inputPort	hasInPort
4	port.output	outputPort	hasOutPort
5	link.port and relation	datalink	DLToInPort, inPortToDL, outPortToDL, DLToOutPort
6	link.port and relation	seqctrllink	sourcePToCL, CLToDestP

Rule 1: In a MoML workflow the *compositeActor* represents an entire workflow or a nested workflow. For each *compositeActor*, the equivalent representation in ProvONE is the *workflow* class with the *wasDerivedFrom* association.

Rule 2: An *actor* in MoML defines a processing task. For each *actor* defined in a MoML workflow, the equivalent representation in ProvONE is the *process* class.

Rule 3 and 4: Data exchange among MoML *actors* happen through the *port.input* and *port.output* elements, the equivalent representation in ProvONE that captures data exchange among *processes* is through the *inputPorts* and *outputPorts* classes.

Rule 5 and 6: In MoML, to share data among *actors*, *relations* are defined with their reference in the *link* element. The equivalent representation for a MoML relation is the ProvONE *datalink* class. Additionally, the execution order is determined based on the *link* element, and is mapped to the *seqctrllink* class to maintain it in ProvONE.

5.3.2 Prov2ONE Algorithm

In this section, we present the Prov2ONE algorithms. Each algorithm accepts a valid workflow defined in either BPEL, SCUFL, or MoML as input and generates the corresponding ProvONE prospective graph. In all of the three algorithms, the following set of ProvONE classes (Σ) and associations (Ω) are used for creating the labeled graph:

$\Sigma = \{$*Workflow, Process, InputPort, OutputPort, DataLink, SeqCtrlLink*$\}$

$\Omega = \{$*wasDerivedFrom, sourcePToCL, CLtoDestP, hasInPort, hasOutPort, inPortToDL, DLToInPort, outPortToDL, DLToOutPort*$\}$

Definition 1. A *ProvONE labeled graph* is defined as a graph $G = (V, E, \lambda, \psi)$, with the following elements:

- a set of vertices $V = \{v_1, v_2, v_3, ..., v_n\}$
- a set of edges $E \subseteq V \times V$
- a vertex labeling function $\lambda : V \to \Sigma$
- an edge labeling function $\psi : E \to \Omega$

Algorithm 5.1. Prov2ONE:BPEL

Input: Workflow W in BPEL specification ▷ $W = \{w_1, w_2, ...w_n\}$ is a vector of nested BPEL activities and each c_i has: $\{.input, .output, .children\}$
Output: Graph G conforming to ProvONE prospective classes and associations

List A:= {*sequence,process,while,scope*} ▷ List of BPEL structure activities that produce a sequential pattern
List B:= {*flow,pick,if,switch,assign*} ▷ List of BPEL structure activities that produce a parallel pattern
List P:= {*invoke,receive,reply,...,*∅ } ▷ List of primitive activities
Stack S := {[*process*, {*R*}, {*R*}]} ▷ R is headset or tailset of primitive activities
function PROV2ONE(W, C)
    for c_i in C **do**
        if $c_i \in$ P **then**
            GENERATEPROVONE(c_i) ▷ Create prospective provenance for c_i
        end if
        if $c_i \in$ A ∪ B **then** ▷ Get the "top" item from stack
            top = PEEK(S)
            $current = [c_i, \emptyset, \emptyset]$
            if $top[0] \in$ A **then**
                if $top[1] == top[2]$ **then**
                    if $c_i \in A$ **then** ▷ Inherit sequential tailset from previous item
                        $current[2] = top[1]$
                    end if
                    $top[1] = \emptyset$
                end if
                $current[1] = top[2]$
                $top[2] = \emptyset$
            else ▷ Inherit parallel headset from previous item
                $current[1] = top[1]$
                $top[1] = \emptyset$
            end if
            if $c_i \in A$ **and** $current[2] = \emptyset$ **then**
                $current[2]$=COPY($current[1]$)
                ▷ Copy primitive activities from headset to tailset
            end if
            PUSH(S, $current$)
            Prov2ONE($W, c_i.children$)
            $current$ = POP(S)
            top = PEEK(S)
            if $top[1] = \emptyset$ **then**
                $top[1] = current[1]$
            end if
            if $top[0] \in$ A **then**
                $top[2] = current[2]$ ▷ Replace sequential tailset
            else
                $top[2] = top[2] \cup current[2]$ ▷ Update parallel tailset
            end if
        end if
    end for
end function

Algorithm 5.2. Prov2ONE:BPEL (continued)

```
1: function GeneratePROVONE(c_i)
2:     V = V ∪ {c_i}                          ▷ Add the primitive activity in vertex set V
3:     λ(c_i) = Process                       ▷ Create Process labeled class
4:     top = PEEK(S)
5:     ATTACHPORTS(c_i)                       ▷ Attach input and output ports for the Process
6:     PROVONEMODEL(c_i, top)
7: end function
8: function PROVONEMODEL(c_i, top)
9:     λ(S) = SeqCtrlLink                     ▷ Create SeqCtrlLink activity S
10:    V = V ∪ {s}
11:    if top[0] ∈ A then                     ▷ Create sequential pattern in ProvONE
12:        ψ(E) = sourcePToCL                 ▷ Create sourcePToCl labeled edge E
13:        CONNECT(top[2], S, E)
14:        ψ(E) = CLtoDestP                   ▷ Create CLtoDestP labeled edge E
15:        CONNECT(S, c_i, E)
16:        top[2] = {c_i}
17:    else                                   ▷ Create parallel structure in ProvONE
18:        λ(S) = SeqCtrlLink
19:        ψ(E) = sourcePToCL
20:        CONNECT(top[1], S, E)
21:        ψ(E) = CLtoDestP
22:        CONNECT(S, c_i, E)
23:        top[2] = top[2] ∪ {c_i}
24:    end if
25: end function
26: function ATTACHPORTS(c_i)
27:    λ(c_i.input) = InputPort
28:    ψ(E) = hasInPort                       ▷ Create hasInPort labeled edge E
29:    if in ∈ D then
30:        L = GET(D, in)                     ▷ Get the datalink
31:        ψ(E) = DLToInPort
32:    end if
33:    ψ(E) = hasOutPort
34:    λ(c_i.output) = OutputPort
35:    λ(L) = DataLink
36:    ψ(E) = outPortToDL                     ▷ Create input,output,datalink edges
37:    E = E ∪ {(c_i.input, c_i)
38:    E = E ∪ {(c_i.output, c_i)}
39:    E = E ∪ {(c_i.output, L)}
40: end function
41: function CONNECT(N, c_i, E)               ▷ Create edges between processes and
    seqctrllinks
42:    for n ∈ N do
43:        E = E ∪ {(n.c_i)}
44:    end for
45: end function
```

Algorithm 5.3. Prov2ONE:SCUFL

Input: Workflow W in SCUFL specification ▷ $W = \{dataflow_1, ..., dataflow_n\}$ is a vector of nested SCUFL data activities
Output: Graph G conforming to ProvONE prospective classes and associations

P := $\{p_1, ..., p_n\}$ set of SCUFL processor ▷ Each p_i is a vector consisting: $\{.activity, .inputPorts, .outputPorts\}$
List T := $\{[processor, \{IP\}, \{OP\}]\}$ ▷ IP, OP are vectors of input and output ports associated with each processor
List A := $\{dataflow, beanshell, rest, ...\}$ ▷ List A consists of activities supported by SCUFL
List D := $\{d_1, ..., d_n\}$ vector of datalinks
function PROV2ONESCUFL($W.dataflow$)
    for $p_i \in P$ **do**
        if $p_i.activity = A.dataflow$ **then**
            $\lambda(p_i) = Process$
            $\lambda(w) = Workflow$ ▷ Distinguish the nested workflow
            CONNECT(p_i, w, $\emptyset$)
            ATTACHPORTS(p_i) ▷ Attach input and output ports to dataflow
            PROV2ONESCUFL(p_i) ▷ Recursive call for nested dataflow
        end if
        $\lambda(p_i) = Process$
        ATTACHPORTS(p_i) ▷ Attach input and output ports to process
        ADD(T, $[p_i, IP, OP]$) ▷ Add process, inputPorts, outPorts in T
    end for
    for $d_i \in D$ **do**
        $\lambda(L) = DataLink$ ▷ Create DataLink labeled class L
        P_s = READ (T, $d_i.source.processor$)
        $\psi(E) = outPortToDL$
        CONNECT($P_s.OutputPort$,L,E)
        P_d = READ (T, $d_i.sink.processor$)
        $\psi(E) = DLToInport$
        CONNECT(L,$P_d.InputPort$,E) ▷ Create datalink between the ports
        $\lambda(S) = SEQCTRLLINK$
        $\psi(E) = sourcePToCL$
        CONNECT(P_s,S,E) ▷ Create seqctrllink between the processes
        $\psi(E) = CLToDestP$
        CONNECT(S,P_d,E)
        DELETE (T,P_s,P_d)
    end for
end function
function ATTACHPORTS(p_i)
    for $p_i.inputPort$ **and** $p_i.outputPort$ **do**
        ADD(IP, $port$)
        $\psi(E) = hasInPort$
        $\lambda(port) = InputPort$
        CONNECT(p_i,$port$,E) ▷ Attach InputPort to process p_i
        ADD(OP, $port$)
        $\psi(E) = hasOutPort$
        $\lambda(port) = OutputPort$
        CONNECT(p_i,$port$,E) ▷ Attach OutputPort to process p_i
    end for
end function

Algorithm 5.4. Prov2ONE:MoML

Input: Workflow W in MoML specification
Output: Graph G conforming to ProvONE prospective classes and associations

List S := $\{[actor, \{IP\}, \{OP\}]\}$
List T := $\{[relation, \{links\}]\}$
List R := $\{r_1, r_2, ..., r_n\}$, List L := $\{l_1, l_2, ..., l_n\}$ is a set of relation and links
function PROV2ONEMOML($W.e_i$)
  for $e_i \in W$ **do**
    if $e_i = CompositeActor$ **then**
      $\lambda(e_i) = Process$
      $\lambda(w) = Workflow$ ▷ Create a Workflow labeled Process in ProvONE
      CONNECT(e_i,w,$\emptyset$)
      ATTACHPORTS(e_i)
      PROV2ONEMoML(e_i)
    end if
    $\lambda(e_i) = Process$
    ATTACHPORTS(e_i)
    ADD(S, $[e_i.name, IP, OP]$) ▷ Add actor,input,output ports in S
  end for
  for $r_i \in R$ **and** $l_i \in L$ **do** ▷ Add all links for a relation in T
    ADD($T, [r_i, \{l_i\}]$)
    GENERATEPROVONE(T)
  end for
end function
function GENERATEPROVONE(T)
  for $l_i \in T[i]$ **do** ▷ Create DataLink between ports
    $\lambda(L) = DataLink$
    P_s = READ (S,l_i) ▷ Get the source Process
    P_d = READ (S,l_i) ▷ Get the sink Process
    $\psi(E) = outPortToDL$
    CONNECT($P_s.OutputPort$,L,E)
    $\psi(E) = DLToInPort$
    CONNECT(L,$P_d.InputPort$, E)
    $\lambda(S) = SeqCtrlLink$
    $\psi(E) = sourcePToCL$
    CONNECT (P_s,S,E)
    $\psi(E) = CLToDestP$
    CONNECT (S,P_d,E) ▷ Create SeqCtrlLink connection
    DELETE(T,P_s,P_d)
  end for
  DELETE(S,e_i)
end function
function ATTACHPORTS(e_i) ▷ Attach all the ports for a Process
  for $e_i.ports$ **do**
    $\psi(E) = hasInPort$
    $\lambda(port) = InputPort$
    CONNECT(p_i,$port$,E)
    $\psi(E) = hasOutPort$
    $\lambda(port) = OutputPort$
    CONNECT(p_i,$port$,E)
  end for
end function

Prov2ONE:BPEL The Prov2ONE:BPEL algorithm comprises two components. The first component, as described in Algorithm 5.1, maintains the structural definition of the workflow and the second component (Algorithm 5.2) is for constructing the ProvONE prospective graph. To identify the starting point of a workflow, a provone:workflow node corresponding to a bpel:process is created. The stack S is filled with triples of the form *[structural element, head node set, tail node set]*, and is initialized with the bpel:process activity. In Algorithm 5.1, from line 12 to line 30, the BPEL activities are distinguished according to structure activities (defined in List A and B), or primitive activities (defined in List P). Structure activities are added to the stack, with their head and tail sets determined according to the topmost activity on the stack. The algorithm recurses on children activities of the inserted structure activity, and are gradually popped upon completion (line 37). In Algorithm 5.2, the *PROVONEMODEL* function generates either a sequence or a parallel structure in ProvONE. In the *ATTACHPORT* function the input and output ports for a process are connected. The *CONNECT* function iterates over all the primitive activities, and the labeled edges between the process and seqctrlink nodes are created. Following operations for manipulating a stack data structure are used, *PUSH* for pushing an element in the stack, *POP* for removing the topmost element from the stack, *PEEK* for checking the topmost element on the stack, and *COPY* for copying the contents of a variable.

Prov2ONE:SCUFL The Prov2ONE:SCUFL algorithm comprises two functions, as shown in Algorithm 5.3. In the Prov2ONESCUFL function, from line 9 to line 40, the algorithm iterates over all the SCUFL processors and the corresponding structure in ProvONE is created. For handling nested workflow, if a scufl:processor is identified as a dataflow activity, then a provone:workflow node is attached with the provone:process node to distinguish a nested workflow, as shown from line 10 to line 19. A recursive call is made to the algorithm for iterating the elements of the nested workflow. From line 25 to line 39, based on the scufl:datalinks, the *SeqCtrlLink*, *DataLink*, *outPortToDL*, *DLToInPort*, *sourcePToCL* and *CLToDestP* are created.

Prov2ONE:MoML Each MoML workflow is attributed with a director that dictates the execution pattern that has to be followed by the Kepler WfMS. However, regardless of the director, the first step is to translate the entire workflow definition with all the relations, i.e. the potential data dependencies and execution paths that are specified in the original workflow to the ProvONE prospective provenance. Eventually, the exact execution order and the actual data values shared between actors can be determined with the enrichment of retrospective provenance. The Prov2ONE:MoML algorithm, as shown in Algorithm 5.4 comprises three functions. In the PROV2ONEMoML functions, from line 9 to line 15, the MoML entities are separated either as a composite actor or as a simple actor. As a MoML composite actor defines a nested workflow, it is mapped to a provone:process node with an associated provone:workflow node. As shown from line 28 to line 41, the *GENERATEPROVONE* function iterates over all the moml:relations that are referenced in

the moml:link element, and the corresponding *SeqCtrlLink*, *DataLink*, *outPortToDL*, *DLToInPort*, *sourcePToCL* and *CLToDestP* are created.

5.3.3 Correctness and completeness of the algorithms

In the following, we consider a BPEL, SCUFL, or MoML workflow to be valid if it conforms to the respective workflow specification, and a correct ProvONE graph as one which preserves the execution order of the given workflow.

Theorem 1. *Let W be a valid workflow defined in BPEL. Then, by the mapping Rules 1–7 defined in Table 5.1, a ProvONE prospective graph G is generated for W.*

Proof. We first guarantee termination. Note that each BPEL child pushes at most one structural activity with enclosed primitive activities onto the stack, and after each call to the Prov2ONE method, an activity is popped off the stack. Thus, the stack becomes empty after a finite number of iterations, which terminates the algorithm.

To prove the correctness of the Prov2ONE:BPEL algorithm, we proceed by induction on the number of activities defined in the workflow, with an upper bound equal to the number of structural activities. By correctness, we imply that the execution order of the primitive activities defined using structural activities for an input workflow is maintained in the ProvONE prospective graph generated by the algorithm.

For the base case, we have $W{=}\emptyset$, and the stack S is populated with the *process* activity with empty head and tail sets. Then, by *Rule 1*, the algorithm generates a singleton graph with a *provone:workflow* node. For a workflow W, let $|W|$ denote the number of activities, including an *empty* structure activity that contains no primitive activities. Now, for the induction hypothesis, we make the following assumptions:

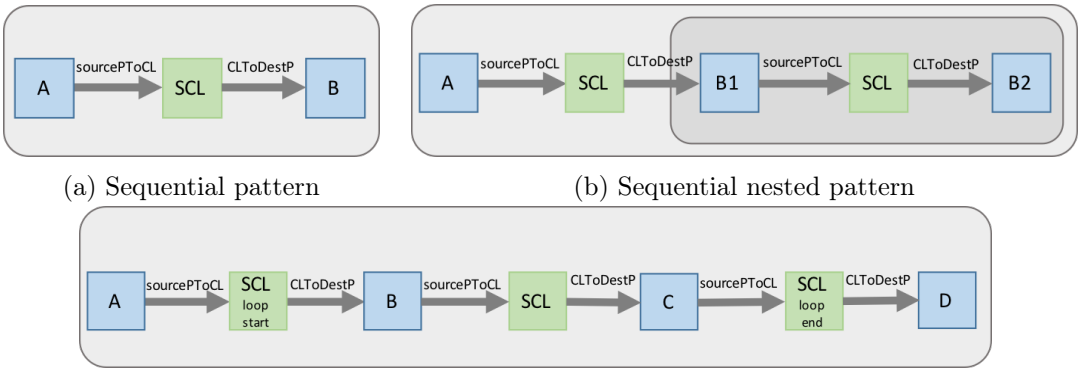
- For any workflow W such that |W| is the number of activities in a workflow. Then for $|W| \leq n$, the *PROV2ONE(W, C)* method creates a correct ProvONE prospective graph.
- Let us assume that a workflow with *n+1* activities is handled and let the last activity in W be x and its immediate parent structure activity be f (i.e., the structure activity on top of stack S). When *PROV2ONE(W, x)* is executed the following holds true:
  - S contains every nested structure around x, in the order from the root *bpel:process* to the current process *f*.
  - The primitive activities in $f[1]$ and $f[2]$ are accurate if the workflow ended without handling the structure activity x.
  - For a structure activity $s \in S$ other than f and $s[0] \in A$ implies $s[1]$ is empty and $s[2]$ contains the last primitive activity before x.
  - For a structure activity $s \in S$ other than f and $s[0] \in B$ implies $s[1]$ is empty and $s[2]$ contains every primitive activity of s excluding those of the activity x.

Now consider a workflow W with $|W| = n+1$. For $x \in A \cup B$, the completion of x implies $x.children = \emptyset$, so recursively executing *PROV2ONE(W, x)* does not change the content of the stack S. However, for a workflow W^* that has the same composition as the workflow W, but without the structure activity x will generate an equivalent ProvONE prospective graph as that of W. Thus, $|W^*| = n$, and by inductive hypothesis it implies that a correct ProvONE prospective graph is generated. When a structure activity s is popped from the stack S, a recursion is completed. After all the activities are popped and the stack S is empty, the algorithm terminates.

Suppose $x \in P$ and $|f.children| > 1$. By the induction hypothesis, W^* run in the Prov2ONE algorithm would produce a correct graph, so we can assume that $f[1]$ is the correct head and $f[2]$ would be the correct tail if $x \notin W$. Then in the *GenerateProvOne* component, as per *Rules 3–6* the function *ATTACHPORTS* will create the necessary inputports, outputports, and the labeled edges for a process.

If $f[0] \in A$, then x is connected to a primitive activity in $f[2]$ and replaces the content of $f[2]$. If $f[0] \in B$, then x is connected to primitive activities in $f[1]$ and added to $f[2]$. By the mapping *Rule 7*, the corresponding sequential or parallel structure is generated in the function *PROVONEMODEL*. As $|f.children| > 1$, f is not an empty structure before the inclusion of x, so the inductive hypothesis guarantees correctness.

Now, suppose $x \in P$ and $|f.children| = 1$, i.e., x is the only activity in f. This implies that f is empty in W^*, so the inductive hypothesis provides no information about how the head and tail sets of the completed structure activity are passed on to the structure activity that is on the top of the stack S. For item s that is immediately below f on the stack, we consider the following cases, beginning at call of *PROV2ONE(W, f)*:



(a) Sequential pattern

(b) Sequential nested pattern

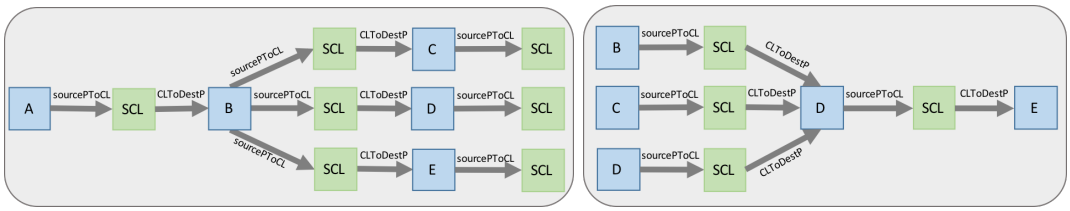
(c) While condition pattern

Figure 5.8: ProvONE sequential prospective patterns

Case 1: $s[0] \in A$, $f[0] \in A$. First we consider when a sequence structure activity follows another sequence activity. For example, a while activity follows a sequence activity, or a scope activity is embedded within a sequence activity. If s is non-empty, then $s[1] \neq s[2]$, so following the call to *PROV2ONE(W, x)*, $s[1] = s[2] = \emptyset$. By induction $f[1]$ contains the correct head set passed on from s and $f[2] = \{x\}$. Note that to satisfy the last inductive hypothesis requirement, it is necessary to copy the contents of $f[1]$ to $f[2]$ before reading x, and this copy is immediately deleted upon

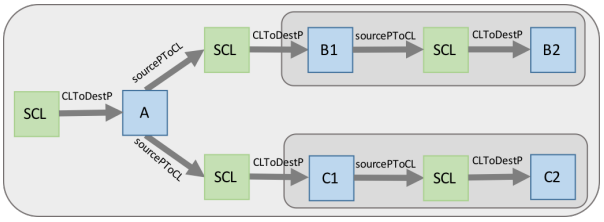
reading x. These values are restored in the topmost item of the stack S, as $s[1]$ is empty it inherits the head of the above structure. As s is empty, it follows the same procedure, except the set $s[1]$ is moved to $f\,[2]$, which enables the algorithm to avoid a redundant copy.

This case is applicable for BPEL workflows with sequentially ordered processes using the ⟨sequence⟩, repetitive execution of processes using structured loop ⟨while⟩, or multiple instances design pattern using ⟨scope⟩. In Section 5.2.3, we described the pattern for sequential control-flow structures in Figures 5.2a, 5.4a, and 5.4b. A sequential execution of arbitrary number of processes (invoke activities) in BPEL is translated to a ProvONE graph that consists of same number of processes interconnected by a SeqCtrlLink node. An example with two processes is shown in Figure 5.8a. In the case of ⟨while⟩ activity, to identify the repetitive section of a workflow, not only the entire looping condition defined in XPath is copied in the SeqCtrlLink node, but also the start of the loop is captured in the SeqCtrlLink that is before the commencing of the loop and the termination of the loop in the SeqCtrlLink that follows the last process in the loop. Figure 5.8b shows the ProvONE translation of the while design pattern comprising two tasks illustrated in Figure 5.4a.



(a) Sequential pattern follwed by a parallel pattern

(b) Parallel pattern follwed by a sequential pattern



(c) Sequential pattern followed by parallel scopes

Figure 5.9: ProvONE parallel prospective patterns

Case 2: $s[0] \in B$, $f\,[0] \in A$ This case is observed when a sequence activity follows a parallel activity. For example, a sequence activity follows a flow, switch, or a pick activity. Initially, $f\,[1]$ saves $s[1]$, and after calling *PROV2ONE(W, x)*, $f\,[2] = \{x\}$. Because f starts as an empty structure, we need to copy $s[1]$ in order to fill $f\,[2]$. After returning the topmost item of stack for *PROV2ONE(W, f)*, $s[1]$ is restored, the copy is deleted, and $s[2] := s[2] \cup f\,[2] = s[2] \cup \{x\}$ as expected.

This case is applicable for workflows where a parallel execution pattern defined by ⟨flow⟩, ⟨pick⟩, or ⟨switch⟩ activity are succeeded by sequential patterns like ⟨sequence⟩ or ⟨while⟩. Initially, the completed set of parallel ⟨invoke⟩ activities are

copied to the tailset *f*[2] of the current activity. This allows the systematic merge of previously completed parallel ⟨invoke⟩ activities. In case of ⟨switch⟩ or ⟨pick⟩ activity, the last ⟨invoke⟩ activity from each branch is copied to the tailset *f*[2] of the current activity. It should be noted that for translating the exact prospective provenance, it is not necessary to know whether it is an AND-join, XOR-join, or an OR-join. For the XOR and OR join, the execution path taken by the workflow is obtained from the retrospective provenance. Hence, all the paths defined in the original workflow have to be translated to the ProvONE prospective provenance. This case covers the Synchronization, Simple Merge, and the Multiple instance with parallel branches control-flow patterns shown in Figures 5.2c, 5.3b, and 5.4b respectively, and the pattern realized in ProvONE is shown in Figure 5.9b.

Case 3: $s[0] \in A$, $f[0] \in B$ Assume a parallel activity follows a sequence activity. For example, a flow structure activity follows a sequence activity. If s is non-empty, then $s[1] \neq s[2]$, so following the call to *PROV2ONE(W, x)*, $s[1] = s[2] = \emptyset$. By induction, the contents of head set of s are passed on to the head set of $f[1]$, and $f[2] = \{x\}$ that is the current tail set of x. Thus, s inherits its head and tail sets as in Case 1.

In workflows wherein the control of execution is transferred from a single thread of execution to multiple threads, it is necessary to connect the last process from the prior sequential section of the workflow to each of the initial processes in the parallel section. For example, to translate the exact execution order of a ⟨sequence⟩ activity having a single ⟨invoke⟩ activity followed by a ⟨flow⟩ activity with multiple ⟨invoke⟩ activities, it is necessary to connect the ⟨invoke⟩ activity from the ⟨sequence⟩ section to all the other ⟨invoke⟩ activities in the ⟨flow⟩ section. This pattern is also applicable for ⟨switch⟩, ⟨if-else⟩, and ⟨pick⟩ activities that follow a ⟨sequence⟩, ⟨while⟩, or ⟨foreach⟩ activities. Thus, to connect the succeeding ⟨invoke⟩ activities from a parallel structure, it is necessary to copy the ⟨invoke⟩ activity from the previously completed structure activity in the headset of the current parallel activity $f[1]$. This case covers Parallel Split, Multi-choice, Deferred Choice, and Critical Section control-flow patterns shown in Figures 5.2b, 5.3a, 5.3c, 5.5a, and 5.5b respectively, and the corresponding pattern in ProvONE is shown in Figure 5.9a and 5.9c.

Case 4: $s[0] \in B$, $f[0] \in B$ The final cases occurs when a parallel activity follows another parallel activity. For example, a flow activity follows a switch activity. Initially, the content of $f[1]$ is passed on to $s[1]$, and after calling *PROV2ONE(W, x)*, $f[2] = \{x\}$. After reading the topmost item of the stack S for *PROV2ONE(W, f)*, $s[1]$ is restored and $s[2] := s[2] \cup f[2] = s[2] \cup \{x\}$, allowing a parallel structure for all the primitive activities.

Thus, for all the cases in a BPEL workflow, the inductive hypothesis guarantees a successful completion, i.e., the Prov2ONE algorithm generates a ProvONE prospective graph G that maintains the same execution order defined in the input BPEL workflow.

Theorem 2. *Let W be a valid workflow defined in SCUFL. Then by the mapping Rules 1–7 defined in Table 5.2, a ProvONE prospective graph G is generated for workflow W.*

Proof. To first prove termination, note that each call to Prov2ONESCUFL handles a single processor element that is added to the list T, even in the case of a dataflow processor element. After creating the corresponding ProvONE classes and associations, the processor element is deleted from the list T. Thus, for a SCUFL workflow defined with a finite number of processors the termination is guaranteed.

To prove the correctness of the Prov2ONE:SCUFL algorithm, we proceed by induction on the number of recursive calls made for the nested workflows. By correctness, we imply that a dataflow defined in a given SCUFL workflow is captured and maintained in the ProvONE prospective graph generated by the Prov2ONE:SCUFL algorithm.

Consider a workflow W with n processors (P) with no dataflow activities, and m datalinks (D) that produce a correct ProvONE prospective graph. If $|P| = n+1$ and $|D| = m$, then the inductive hypothesis guarantees correctness until the final processor is resolved by applying *Rules 2–4*. Similarly, if $|P| = n$ and $|D| = m+1$, the inductive hypothesis followed by *Rules 5–7* gives a correct ProvONE prospective graph until the final datalink is resolved.

This forms the base case for a larger induction on the nesting of dataflow activities. For a workflow W in which there are at most k nested dataflows, the Prov2ONE algorithm creates a correct ProvONE prospective graph. Consider a workflow W' with at most $k+1$ nested dataflows. Let $\{p_1, \ldots, p_t\}$ be the processors with dataflow activity, then it follows that for each p_i with at most k nested dataflows, a recursive call to the algorithm will produce the correct ProvONE prospective sub-graph. Thus, as per the inductive hypothesis, it shows that W' will also produce the correct ProvONE prospective graph. The algorithm terminates after all recursions are completed, this is indicated when all the *processor* elements are removed from the list T. Thus, for SCUFL workflows, the inductive hypothesis guarantees that a correct ProvONE prospective graph G is generated.

Theorem 3. *Let W be a valid workflow defined in MoML. Then by the mapping Rules 1–6 defined in Table 5.3, a ProvONE prospective graph G is generated for the workflow W.*

Proof. The formality of termination is dealt with by noting that the Prov2ONEMoML method handles one MoML entity per call, so regardless of recursive workflows defined by a composite actor, the algorithm concludes after iterating over each entity.

To prove the correctness of the Prov2ONE:MoML algorithm, we proceed by induction on the number of recursive calls made for the nested workflows. By correctness we imply that the dataflow defined in a given MoML workflow is captured and maintained in ProvONE prospective graph generated by the Prov2ONE:MoML algorithm.

For a workflow W with n actors (A), none of which is a composite actor, and m links (L), the Prov2ONE:MoML algorithm will create a correct ProvONE prospective graph. If $|A| = n+1$ and $|L| = m$, then the inductive hypothesis guarantees correctness until the final actor is resolved by applying *Rules 2–4*. Similarly, if $|W| = n$ and $|L| = m+1$, the inductive hypothesis followed by *Rules 5–6* generates a correct ProvONE graph.

This forms the base case for a larger induction on the nesting of composite actors. For any workflow W in which there are at most k composite actors, the Prov2ONE:MoML algorithm generates a correct ProvONE prospective graph. Consider a workflow W' with at most $k+1$ nested actors. Let $\{e_1, \ldots, e_t\}$ be the composite actors, then it follows that for each e_i that has at most k nested actors, a correct ProvONE prospective sub-graph is created. Then, as per inductive hypothesis, reapplying the above argument on all the composite actors of W' will also produce the correct ProvONE prospective graph. The algorithm terminates after all the recursions are completed. This is indicated when all the *actors* are removed from the list S. Thus, for MoML workflows, the inductive hypothesis guarantees that a correct ProvONE prospective graph G is generated.

From Theorems 1, 2 and 3 it follows that for a workflow defined in BPEL, SCUFL, or MoML, the Prov2ONE algorithm will generate a correct ProvONE prospective graph.

5.3.4 Retrospective mapping rules between WfMSs and ProvONE

In this section, to complete the translation of the entire provenance from WfMS to ProvONE, we present the mapping rules between the runtime provenance traces from ApacheODE, Taverna, and Kepler and the ProvONE retrospective provenance. These mapping rules are realized in the WfMS-specific provenance adapter either as SQL queries for retrieving the runtime events from the relational database of ApacheODE, or SPARQL queries in the case of provenance exported from Taverna and Kepler.

Table 5.4: ApacheODE events to ProvONE retrospective translation rules

#	Event name	Event value	ProvONE:Class	ProvONE:Association
1	ns:name, ns:process-id	NewProcessInstance	ProcessExec	wasAssociatedWith
2	ns:name, ns:activity-id	ActivityExecStart	ProcessExec	wasAssociatedWith
3	ns:instance-id	—	ProcessExec	isPartOf
4	ns:message, ns:input	payload, parameters	Data	used
5	ns:message, ns:output	payload, parameters	Data	wasGeneratedBy
6	ns:message	payload, parameters	Collection	—

ApacheODE events and ProvONE retrospective provenance

A workflow execution trace in ApacheODE is captured in the form of events that are persisted in a relational schema specific to ApacheODE. There are three options to retrieve the execution details of a workflow run: (a) The ODE Execution Events module in ApacheODE captures each event that occurs within the workflow engine, and an event listener can be registered for analyzing and extracting the relevant events. (b) SQL queries can be executed over the event database to query and retrieve the required events. (c) A Management API that provides an abstraction over the ODE Execution Events module and exposes the BPEL processes, instances and data exchange events through dedicated web services (endpoints), with the extension to query specific details about an event.

From the options mentioned above, we integrated the Management API in the ApacheODE Adapter. The reasons to choose the Management API over the other two options are twofold: (1) The necessary functionality for querying and retrieving the events from the event database schema is already implemented and exposed via this API. Hence, defining a custom set of queries for retrieving the stored events will be redundant implementation. Moreover, with updates to the event database schema, our queries would be obsolete. However, with the Management API, the changes to the schema would also be propagated to the Management API. (2) The ApacheODE developers state that enabling ODE Execution Events module introduces a non-negligible overhead while execution of a workflow, and will affect the workflow execution time. Thus, the option of ODE Execution Events is not considered for the implementation of the adapter.

In the first step, the workflows are translated to ProvONE prospective provenance graphs. In the second step, these graphs are enriched with the translation of the runtime events from ApacheODE to ProvONE retrospective classes and associations, shown in Table 5.4. The events are defined using Apache ODE namespace (ns) and exposed through two core endpoints namely *listEvents* and *getCommunication*. Following is the explanation of each rule.

Rule 1: For an execution of a workflow deployed in ApacheODE WfMS, the system generates a ns:name with a ns:process-id event. The event value NewProcessInstanceEvent indicates that the entire workflow is triggered for execution and for each execution a unique process-id is created. Thus, for this event, a ProcessExec class with the link wasAssociatedWith to the ProvONE prospective provenance is created.

Rule 2: Similar to the first rule, for execution of each activity within a given workflow, the ns:name event with the value ActivityExecStartEvent is generated. The activity-id uniquely identifies each instance of an active activity. Thus, for an event, a ProcessExec class with the wasAssociatedWith link connecting to the Process class of the same activity in the ProvONE prospective provenance is created. In the case of a failed activity, an ActivityFailureEvent is generated by the workflow engine. For tracing the failed activities, this event is added to the ProcessExec. Additional events necessary for creating retrospective provenance in ProvONE are appropriately handled

in the ApacheODE Adapter. The complete list of events is available on ApacheODE website[7].

Rule 3: The ns:instance-id together with the ns:process-id describes the association of activity with the execution instance of a workflow. Thus, using this event, the association isPartOf is created for each ProcessExec.

Rule 4 and 5: In BPEL, data is shared in the form of messages. The event ns:message and ns:input event with the value payload or parameters indicate the input for a given activity, whereas the ns:output indicates the output generated by an activity. Thus, to describe the input and output of a process in ProvONE, these events are mapped to the Data class with the used or wasGeneratedBy association.

Rule 6: The ns:message in its entirety represents a complex data structure in BPEL. Similarly, the Collection class in ProvONE represents a collection of atomic data elements. Thus, the ns:message is mapped to the Collection class.

With the implementation of these six rules in the ApacheODE adapter, the retrospective provenance from ApacheODE WfMS is translated to ProvONE. However, few associations have to be derived from the prospective provenance. The association wasInformedBy is created based on the prospective provenance SeqCtrlLink and the DataLink associations between a given pair of processes. The association wasDerivedFrom and dataOnLink associations are created based on the DataLink associations between the ports for a given pair of processes.

Table 5.5: Taverna to ProvONE retrospective translation rules

#	namespace:Class	namespace:Association	ProvONE:Class	ProvONE:Association
1	WorkflowRun	describedByWorkflow	ProcessExec	wasAssociatedWith
2	wfprov:ProcessRun	describedByProcess	ProcessExec	wasAssociatedWith
3	prov:Entity	prov:used or wfprov:usedInput	Data	used
4	prov:Entity	prov:wasGeneratedBy or wfprov:wasOutputFrom	Data	wasGeneratedBy
5	wfprov:ProcessRun	wasPartOfWorkflowRun	ProcessExec	isPartOf
6	prov:Dictionary, prov:Collection, prov:Entity	prov:hasMember	Collection	hadMember
7	prov:Entity	prov:wasDerivedFrom	Data	wasDerivedFrom

Taverna provenance and ProvONE retrospective provenance

The provenance traces in Taverna WfMS are stored in a Taverna-specific provenance data model. However, for sharing and interoperability reasons, the Taverna WfMS is extended with the PROV plugin that translates and exports the provenance traces to the PROV model. The mappings in Table 5.5 show that the PROV plugin uses multiple namespaces (prov, wfprov, rdf) to expose a detailed level of provenance. Moreover, as ProvONE retrospective model inherits a large number of classes and associations

[7]http://ode.apache.org/ode-execution-events.html

from the PROV model, it allowed us to maintain the same level of granularity during the construction of the ProvONE retrospective provenance.

Rule 1: The PROV plugin of Taverna uses the wfprov:WorkflowRun and the wfprov:describedByWorkflow terms from the wfprov namespace to represent an execution of workflow with it corresponding workflow definition. Hence, this representation from Taverna is mapped to ProvONE ProcessExec class with the wasAssociatedWith association that represents an execution instance of a workflow with its associated definition.

Rule 2: For tracing the execution of individual processes with its corresponding definition, the PROV plugin uses the wfprov:ProcessRun class with association wfprov:describedByProcess. The same representation in ProvONE is captured by the ProcessExec class with the wasAssociatedWith association. Hence, the terms from wfprov namespace are translated to the ProvONE namespace.

Rule 3: To map the runtime consumption of data by a process, the prov:Entity with association wfprov:usedInput or prov:used are translated to ProvONE Data class with the used association.

Rule 4: Similar to Rule 3, to represent the data produced as a result of execution of a process, the prov:Entity with association prov:wasGeneratedFrom or wfprov:wasOutputFrom are translated to ProvONE Data and wasGeneratedBy association.

Rule 5: Typically, a workflow is composed of one or more processes, and this composition in PROV plugin is represented by the wfprov:ProcessRun with the association wfprov:wasPartOfWorkflowRun. To translate this composition in ProvONE, these terms from the wfprov namespace are translated to ProvONE ProcessExec with isPartOf association.

Rule 6: For describing the composition of a complex data object, in PROV plugin the class Dictionary, Collection, or Entity with the association hasMember is used. The ProvONE vocabulary directly inherits this class and association from the prov namespace and hence are a direct translation to the ProvONE Collection class with the hadMember association.

Rule 7: A data-flow trace in PROV plugin is represented using the prov:Entity Class with the prov:wasDerivedFrom association. In ProvoNE the wasDerivedFrom association is inherited from the prov namespace and thus is a direct translation to ProvONE Data class and wasDerivedFrom association.

The association dataOnLink has to be derived from the prospective provenance. For this, we retrieve the DataLink class between the ports of a given pair of processes and the association dataOnLink is created. Thus, the connection between the prospective and retrospective provenance is completed.

Kepler provenance and ProvONE retrospective provenance

The Kepler WfMS provides an add-on module for recording workflow execution history. The retrospective provenance is primarily recorded in a relational database

Table 5.6: Kepler to ProvONE retrospective mapping rules

#	namespace:Class	namespace:Association	ProvONE:Class	ProvONE:Association
1	prov:plan (kepler:workflow)	wasAssociatedWith	ProcessExec	wasAssociatedWith
2	prov:activity (kepler:actorName)	wasAssociatedWith	ProcessExec	wasAssociatedWith
3	prov:Entity	wasGeneratedBy	Data	wasGeneratedBy
4	prov:Entity	used	Data	used
5	prov:agent (kepler:user)	—	User	wasAttributedTo

schema, and there are two approaches to retrieve this provenance. First, SQL queries can be executed to retrieve the provenance trace from the database. Second, the provenance add-on module implements a mapping between the SQL schema and the PROV model and exports the provenance in JSON format. For a comprehensive translation, multiple namespaces are applied in translating the entire trace in PROV model. In the Kepler Adapter, we extend this mapping for the ProvONE model. The translation rules are shown in Table 5.6.

Rule 1: For maintaining the correspondence between a workflow execution and its definition, the Kepler module uses the prov:plan or the kepler:workflow with the wasAssociatedWith association from the prov namespace. Hence, the prov:plan and kepler:workflow classes are mapped to the ProcessExec class, whereas the same wasAssociatedWith association is inherited from the prov namespace in ProvONE.

Rule 2: For translating the runtime details of each process, the Kepler module uses the activity or the actorName class from the prov and kepler namespace. These classes are mapped to ProvONE ProcessExec that represents the execution details of a single process. The wasAssociatedWith association is directly inherited from the prov namespace and thus is a straight forward translation in ProvONE.

Rule 3 and 4: The Entity represents the consumption and production of data by a process with the used and wasGeneratedBy association. The ProvONE model inherits both the class and the association and hence is a direct translation between PROV and ProvONE.

Rule 5: The retrospective provenance exported by the provenance module contains the reference to the user or agent that is responsible for the execution of the workflow. Hence, the prov:agent or the kepler:user term is mapped to the ProvONE User attribute with the association wasAttributedTo.

Similar to the Taverna adapter, the dataOnLink association is derived from the prospective provenance DataLink class between the output and input ports of given pair of processes. Thus, the connection between the prospective and the retrospective provenance is completed.

5.3.5 Prov2ONE Algorithm Analysis

In this section we discuss the time and space complexity of the Prov2ONE algorithms. The Prov2ONE:SCUFL algorithm maintains only the ProvONE prospective graph and a list structure T, as shown in line 2 of Algorithm 5.3. To calculate the time-complexity we need to consider the for-loop defined from line 9 to line 23. The for-loop has an upper-bound that is equal to the number of processor elements $|P|$. Associated with each processor $|P|$ are $|IP|$ input ports, $|OP|$ output ports and $|D|$ datalinks. Thus, the runtime requirements are $O(|P|+|IP|+|OP|+|D|)$. The memory requirements are just the size of the graph, so space complexity is also $O(|P|+|IP|+|OP|+|D|)$.

Similarly, the Prov2ONE:MoML algorithm maintains only the graph and a list structure S, as shown in line 2 of Algorithm 5.4. The time-complexity is bound by the number of entities $|E|$ that are iterated in the for-loop defined from line 8 to line 20. Each entity can have a link $|L|$ to an another entity defined by the relations $|R|$. Thus, for a MoML workflow the runtime of the algorithm will be $O(|E|+|R|+|L|)$. Since the only space needed is that to store the graph, the space complexity is also $O(|E|+|R|+|L|)$.

Now we consider the Prov2ONE:BPEL algorithm complexity. Let $|S|$ denote the number of structure activities, $|I|$ the number of primitive activities, and $|D|$ the number of data elements for a BPEL workflow.

With regard to space complexity, the algorithm ensures that if an item in the stack is copied then it will be destroyed before another copy takes place, and therefore, every item can appear at most twice in the stack. Thus, the total space is $O(|S|+|I|)$. Additional space requirement that needs to be calculated is for generating the output, i.e., the ProvONE graph itself, explained as follows.

The runtime complexity is at least the time complexity of reading the workflow and assembling a graph of appropriate size, which is $O(|S|+|D|+|I|+|V|+|E|)$, where V and E are the vertex and edge sets defined by Σ and Ω, respectively. Besides these two tasks, the Prov2ONE:BPEL algorithm additionally maintains the stack, which requires $O(|S|+|I|)$ time. To get the runtime complexity, it is necessary to calculate the upper-bound for the $|V|$ and $|E|$ in terms of the input.

If the total number of *DataLinks* in the graph is $|L|$, then clearly $|V| \leq |D|+|I|+|L|+|S|$, as each *Process*, *InputPort*, *OutputPort*, *DataLink* and *SeqCtrlLink* is a vertex in the ProvONE graph. Although the number of *DataLinks* can be quadratic in terms of $|D|$, in practice this does not occur, and is limited by the number of *InputPorts* and *OutputPorts*, thus, it implies that $|L| = O(|D|)$.

Regarding $|E|$ we consider all types of associations defined by the edge set Ω. Assume all *DataLink*s have a fixed number of intermediate uses that are treated as a constant, then the cardinality of the *DLToInPort* and *outPortToDL* sets are $O(|L|) = O(|D|)$.

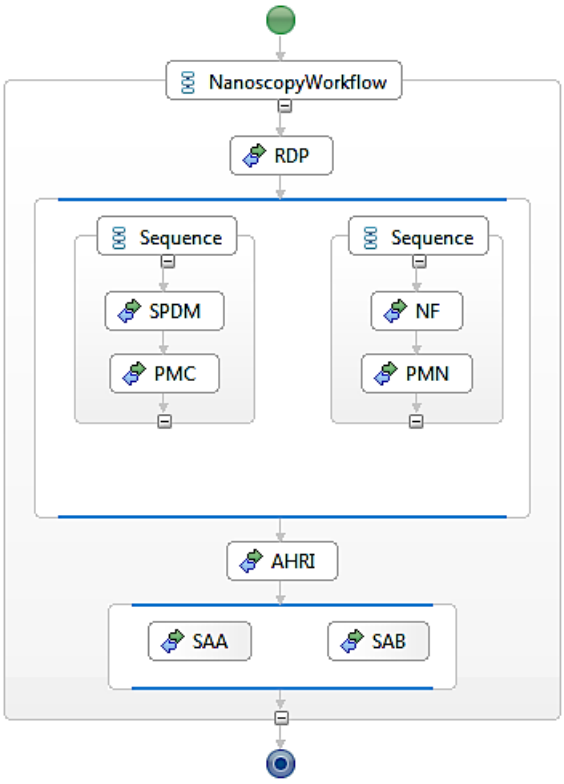


Figure 5.10: Nanoscopy workflow in BPEL

In general, it is uncommon although possible for the remaining four edge sets to be of quadratically sized. Restricting to workflows with only a linear number of such edges guarantees that the runtime is $O(|S|+|D|+|I|)$.

5.4 Evaluation

We evaluate P-PIF on the following two criteria. First, to assess the practical usage of the framework in handling heterogeneous provenance information, we present a set of queries that can assist the researchers in analyzing the workflows and provenance in their routine activities. Second, we present a feature-based evaluation of P-PIF with the existing provenance interoperability frameworks. This assessment is necessary to evaluate that the P-PIF addresses the gaps in existing literature.

5.4.1 Use cases

We briefly describe three workflows defined in BPEL, SCUFL, and MoML, and present their corresponding ProvONE graphs. These use cases show the applicability of P-PIF in handling heterogeneous workflow specifications.

The **Nanoscopy** workflow (Figure 5.10) defined in BPEL begins with the invocation of the `Raw Data Processing (RDP)` activity, followed by two parallel sub-activities `Normal Fit (NF)` and `Spectral Precision Distance Microscopy (SPDM)`. Each activity, `NF` and `SPDM` are followed by sequential activities, `Position`

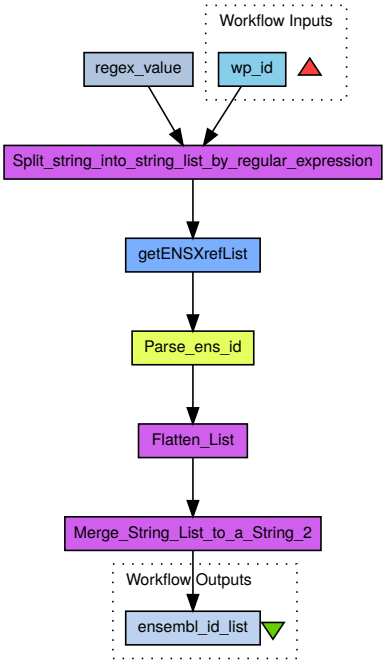


Figure 5.11: PW2ENS workflow in SCUFL

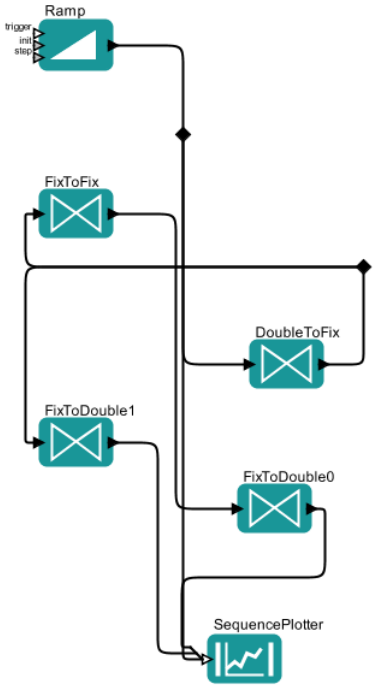


Figure 5.12: FixPoint workflow in MoML

`Matrix Nearest Neighbor (PMN)` and `Position Matrix Cluster (PMC)` respectively. The results of these activities are consumed by the `Annotate High-Resolution Images (AHRI)` activity to produce the annotated image set. Finally, based on the annotation criteria, activities `Segment Annotation-A (SAA)` and `Segment Annotation-B (SAB)` are invoked to produce the result segmented images.

The **PW2ENS**[8] workflow (Figure 5.11) defined in SCUFL start with the process `regex_value (RV)` that generates a regular expression value, followed by processes `split_string_into_string_list (SSSL)`. The result of `SSSL` is consumed by the `get_ENSX_ref_list (ENSX)` process, which is followed by sequential execution of `parse_ens_id (PENS)` and the `flatten_list (FL)` processes. Finally, the workflow ends with the `merge_string_list (MSL)` process.

For testing the Prov2ONE:MoML algorithm, the **FixPoint**[9] workflow defined in MoML was selected, see Figure 5.12. The workflow begins with the invocation of the `RAMP` actor, which is followed by `DoubleToFix (DTF)`. The output of `DTF` invokes `FixToFix (FTF)` and `FixToDouble1(FTD1)` actors. The `FTF` is followed by `FixToDouble0 (FTD0)` actor. The workflow ends with the `SequencePlotter (SP)`.

The ProvONE graphs generated by the Prov2ONE algorithms that are enriched with the retrospective provenance from the respective WfMS-specific adapters for the workflows described in BPEL, SCUFL, and MoML (Figures 5.10, 5.11, and 5.12) are shown in Figures 5.13 to 5.15 respectively.

[8] http://www.myexperiment.org/workflows/4736.html

[9] http://ptolemy.eecs.berkeley.edu/ptolemyII/ptII10.0/ptII10.0.1/ptolemy/domains/sdf/demo/FixPoint/FixPoint/

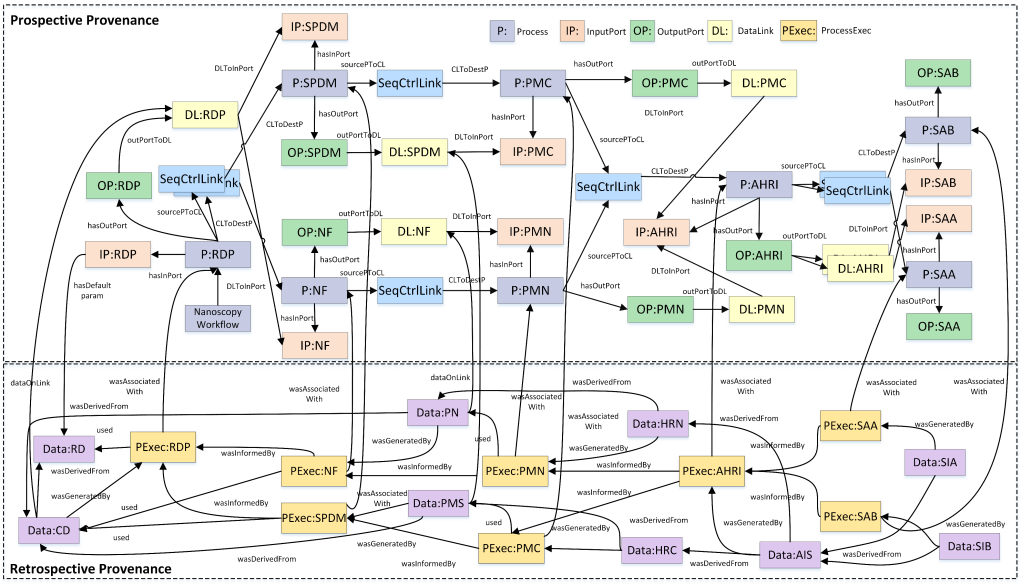


Figure 5.13: ProvONE provenance graph for nanoscopy workflow

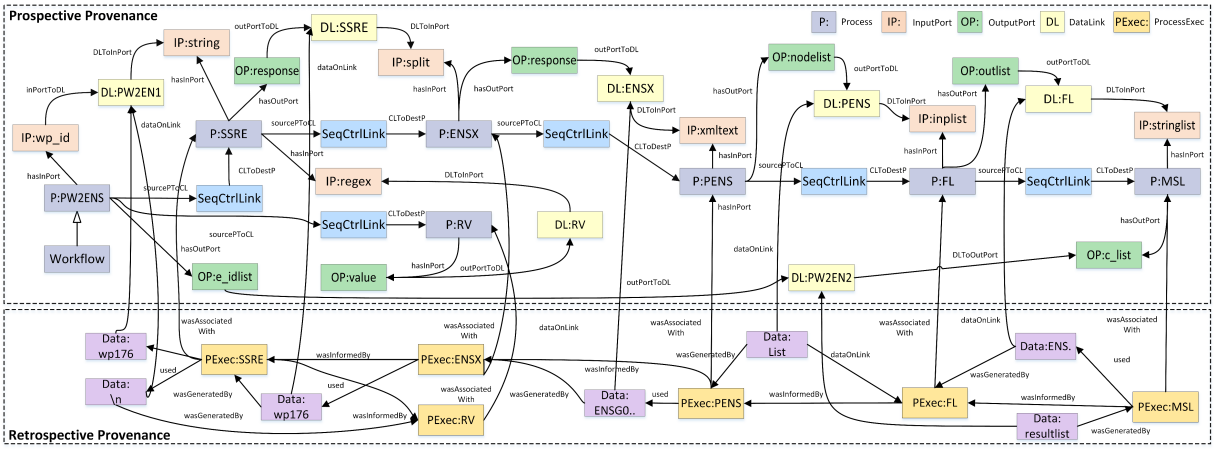


Figure 5.14: ProvONE provenance graph for PW2ENS workflow

5.4.2 Provenance Challenge Queries

We argue that for enabling provenance interoperability, i.e., allow querying and analysis of provenance from different WfMSs, first, it is necessary to translate the workflows from their original representation into a common model like ProvONE. Second, to complete the translation in ProvONE, the runtime provenance (retrospective provenance) needs to be linked to the prospective provenance. Thus, with the translation of heterogeneous provenance information in a common model, it is now possible to express graph traversal queries for analyzing the provenance information.

Following are the benefits of having heterogeneous workflows and their runtime traces as ProvONE graphs: (1) Pattern matching queries can be executed for comparing and analysing workflow definitions regardless of the original representation of the workflows; (2) As each version of workflow is modeled in ProvONE, it is possible to trace the evolution of a given workflow over a period. (3) As the execution order

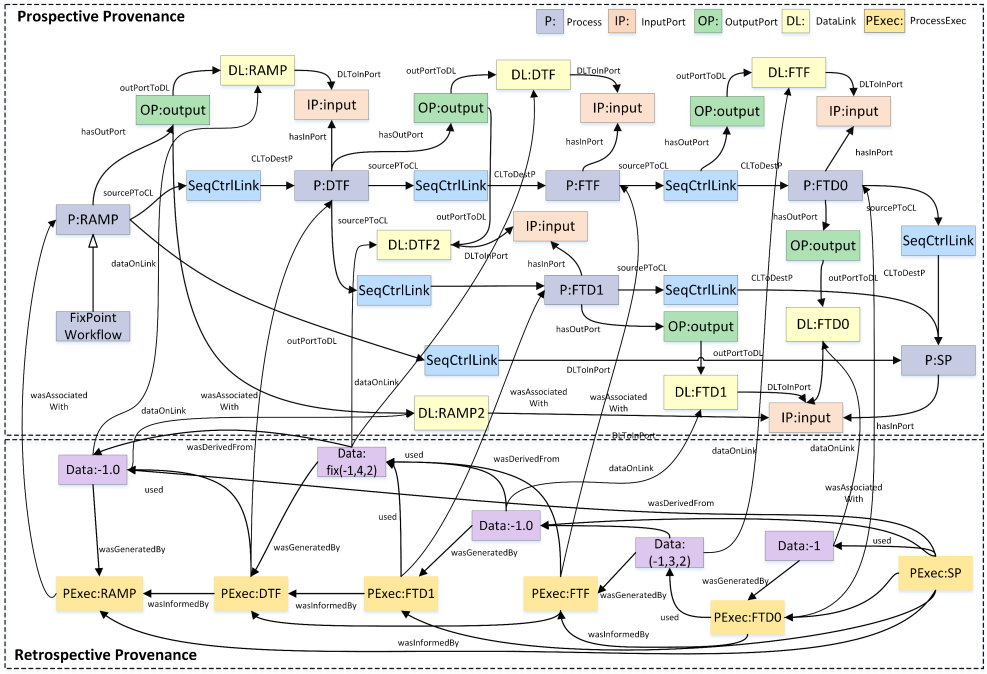


Figure 5.15: ProvONE provenance graph for FixPoint workflow

and the data-flow structure of the original workflow are maintained in the ProvONE prospective graph, it helps the researchers to reconstruct the workflow definition in another specification for executing it in a different execution environment and validating the result. It should be noted that the ProvONE prospective provenance is only a reference for the researcher when assembling a workflow definition in a different specification. The actual services, software, or the algorithm implementations (functions) that build the actual workflow has to be also migrated to the target workflow system.

To realize the benefits mentioned above as usable services for the research communities, we present six graph traversal queries implemented in SPARQL. For allowing a clear distinction between the ProvONE prospective and the retrospective provenance, we have attached *prospective* or *retrospective* keywords to the resource URI. For example, the ProvONE prospective resources URI for the PMC process in BPEL is in the format: http://nanoscopy.edu/**prospective**/process-PMC1. The complete list of queries, including the provenance challenge queries [113] are exposed through a dedicated REST-API[10].

Query 1: Retrieve the prospective provenance (workflow-plan) for the workflow with *workflow-id=nanoscopy-imageprocessingWF*. The query aims at retrieving the complete workflow definition (ProvONE prospective provenance) as described in the original workflow definitions. For this query, the result is the ProvONE prospective graph shown in Figure 5.13.

[10]http://datamanager.kit.edu/masi/localizationmicroscopy/swagger-ui/

```
SELECT {?subject ?predicate ?object}
 WHERE {
GRAPH <http://nanoscopy.edu/nanoscopy-imageprocessingWF> {
   ?subject ?predicate ?object.
   FILTER regex(str(?subject), "prospective")
       }
   }
```

Query 2: Retrieve all the *inputPorts* and *outputPorts* for a workflow or a process with *workflow-id=nanoscopy-data-curationX2* or *process_ name = SPDM*. There are two parts to this query: (a) The first query returns all the inputPorts and outputPorts of a given workflow. (b) The second query returns all the inputPorts and outputPorts of a given process.

Query 2 (a):

```
SELECT ?object ?predicate ?value
WHERE { GRAPH <http://nanoscopy.edu/nanoscopy-data-curationX2> {
    { ?subject provone:hasOutPort ?object.}
    UNION
    { ?subject provone:hasInPort ?object.}
    ?object ?predicate ?value }
}
```

Query 2 (b):

```
SELECT  ?object ?property ?value ?graph
WHERE { GRAPH ?graphName {
        ?subject rdf:type provone:Process.
        ?subject dc:title "SPDM".
      { SELECT * WHERE { GRAPH ?graphName{
            { ?subject provone:hasOutPort ?object.}
            UNION
            { ?subject provone:hasInPort ?object.}
          ?object ?property ?value. } }
    }
  }
}
```

Query 3: Retrieve the processes which has an *inputPort* with parameter *minDensity = 0.3*. The result of this query is all the inputPorts that have the given parameter and value.

```
SELECT  ?processName ?inputPort ?minDensity
WHERE { GRAPH ?graphName {
    ?subject a provone:Process .
    ?subject provone:hasInPort ?inport.
    ?inport dc:title ?inputPort.
    ?inport provone:inPortToDL ?DL.
    ?subject dc:title ?processName.
    {
      SELECT  * WHERE { GRAPH ?graphName {
          ?data a provone:Data.
          ?data prov:value ?value.
          ?data provone:dataOnLink ?DL
          FILTER (?minDensity = 0.3) } }
    }
  }
}
```

Query 4: Retrieve the processes which has an *outputPort* with parameter *datasize* $\geq$ *30GB*. The result of this query is all the outputPorts that have the given parameter and value.

```
SELECT  ?processName ?outputPort ?datasize
WHERE {
  GRAPH ?graphName {
    ?subject a provone:Process .
    ?subject provone:hasOutPort ?outport.
    ?outport dc:title ?outputPort.
    ?outport provone:outPortToDL ?DL.
    ?subject dc:title ?processName. {
      SELECT *
      WHERE { GRAPH ?graphName {
          ?data a provone:Data.
          ?data prov:value ?value.
          ?data provone:dataOnLink ?DL
          FILTER (?datasize >= 30) } }
    }
  }
}
```

Query 5: Retrieve all the workflow plans (prospective provenance) which consist of a *Process* with *process_name = Cluster Analysis*. The result of this query is the complete ProvONE prospective graph as shown in Figure 5.13. This query will retrieve all the prospective provenance traces that contains the *Process* with attribute *process_name = Cluster Analysis*.

However, this query is also extensible for tracing a workflow evolution, i.e., if a given *process_name* is encountered in multiple versions of workflows then the evolution of the workflow can be traced. The only change that needs to be done in this query is to specify the namespace of the graph for which the workflow evolution is expected. For example, in the inner-most nested GRAPH ?graphName declaration, replace the "?graphName" by "http://nanoscopy.edu/nanoscopy-data-curation" for retrieving the workflow evolution of the nanoscopy-data-curation workflow.

```
SELECT ?subject ?predicate ?object where{
  GRAPH ?graphName {
  ?subject ?predicate ?object.
   FILTER regex(str(?s), "prospective") }
   {
    SELECT *
     WHERE { GRAPH ?graphName {
    ?subject a provone:Process .
    ?subject dc:title "Cluster Analysis".
    ?subject dc:title ?processName.
        {
        SELECT *
        WHERE {
        GRAPH ?graphName {
          ?subject1 dc:title ?workflowTitle .
          BIND(URI(?workflowTitle) as ?graphName) }}}}
        }
   }
}
```

Query 6: Retrieve all the workflows that failed to complete successfully, and also return the details of the process that failed, including the error message. The result of this query is the list of all workflows that were not completed and the name of the failed-process with the details of the error message from the ProvONE *ProcessExec* retrospective provenance.

```
SELECT ?subject ?error
    WHERE{ GRAPH ?graphName {
     ?subject a provone:ProcessExec.
     ?subject prov:wasAssociatedWith ?object.
     ?object a provone:Workflow.
     ?subject wfms:completed ?completed.
     ?subject wfms:error ?error.
     FILTER(?completed ="false")
    }
}
```

The above-stated queries can be adapted to user specified parameters for retrieving the provenance modeled as ProvONE graphs. *Query 1* retrieves the full prospective provenance (workflow definition), which can be used to reconstruct the workflow for executing it in a different execution environment (WfMS). *Query 2,3,4* focus on comparing and analyzing workflow definitions (prospective provenances). *Query 5* with its extension supports retrieving of workflows that are similar and also the evolution traces of a given workflow. *Query 6* is necessary for detecting the incomplete or failed workflow runs (both prospective and retrospective provenance) and precisely determining the step at which the workflow failed.

5.4.3 Features of P-PIF

The need to share, query, and analyze provenance information from heterogeneous sources has led to the development of multiple provenance interoperability frameworks. From the literature, it can be seen that the existing frameworks do not offer comprehensive provenance interoperability. Hence, for considering a provenance interoperability framework as a complete solution, a few essential properties need to be fulfilled by the framework. These properties are described in the following, followed by a systematic comparison of the existing solution with the P-PIF based on these properties.

- Provenance granularity: An important property for enabling provenance interoperability is the adoption of a suitable provenance model. A provenance model should be flexible in modeling varying levels of detail and also handle both prospective and retrospective provenance information. For example, OPM, PROV, Collaborative Data Model (CDM), and Common Provenance Model (CPM) are capable of modeling only the retrospective provenance, whereas IWIR is designed for modeling only the prospective provenance. D-PROV and its descendent, namely the ProvONE model are capable of modeling both prospective and retrospective provenance. Thus, for allowing comprehensive provenance interoperability, it is necessary to adopt either the D-PROV with PROV model or the ProvONE model.

- Long-term sustainability and adherence to standards: A framework can be considered sustainable if it is designed on tools and technologies that are standardized and widely accepted by the research communities. RDF and SPARQL that are W3C standard data model and recommended querying language are adopted by multiple provenance research groups. Moreover, PROV-O provides an OWL2 ontology for mapping the PROV data model to RDF namespace. Supporting these standards and recommendation, the Taverna PROV plugin and the Kepler provenance module export provenance in RDF serialized graphs.

- Extensibility: Currently, there exist more than thirty WfMS that are employed by various scientific communities. Each WfMS utilizes a proprietary workflow

specification for expressing the workflows and its associated runtime provenance. For enabling provenance interoperability among the WfMS-specific provenance models, it is necessary to have an interoperability framework build on an extensible data model. For example, using the RDF data model, it is possible to add WfMS-specific vocabularies into the existing RDF namespace without requiring architecture-level modifications or changes to the database schema.

– Querying interface: Expressing provenance retrieval queries in a standard querying language like SQL or SPARQL is important for analyzing the collected provenance. Since provenance information represents a causal relationship among processes and resources, ideally a graph data model should be used. Moreover, a typical characteristic of provenance is that over a period these provenance graphs tend to form large and dense network structures that can easily contain millions of nodes and relations. Thus, query languages that are designed especially for performing efficient graph traversal should be supported by the framework. For example, SQL queries are not an efficient solution because they involve multiple join operations for retrieving graph relations. On the contrary, RDF databases or graph databases are designed to support efficient graph traversal queries. Furthermore, to support this property, there exist multiple studies that evaluate the query performance of graph databases against the traditional relational databases, and state that for complex and dense data containing a large number of relations, a graph database is an appropriate choice [43, 163].

Based on the above-stated properties, we provide a systematic comparison of the existing provenance interoperability frameworks with the P-PIF.

First, we consider the provenance interoperability frameworks that are based on OPM or its variant. A common limitation of the frameworks proposed by Braun et al. [24], Ding et al. [51], and Altintas et al. [6] is that these frameworks are build on OPM, which is a provenance model that inherently allows only retrospective provenance modeling. Hence, prospective provenance interoperability, i.e., sharing, querying and analyzing workflow definitions including their evolution traces are not supported by these systems. Evaluating each framework separately, the framework proposed by Altintas et al. [6] is based on the Collaborative Data Model (CDM), with the Query Language for Provenance (QLP). The aim of the QLP is to provide simplicity of query expressiveness over the existing querying techniques such as SPARQL, SQL, and XPath. However, neither the QLP is a standard querying language nor the CDM is standard provenance model. Under the SHIWA framework, Plankensteiner et al. [122] introduced the syntax and specification of IWIR for translating heterogeneous workflow definitions into a common workflow specification. On the on hand, the IWIR supports translation of a wide range of workflow languages, but on the contrary, the IWIR is not a standard provenance model and is not compatible to P-PLAN, D-PROV, or ProvONE. Moreover, the SHIWA framework provides no support for storing and querying the IWIR modeled prospective provenance.

For enabling comprehensive provenance interoperability, Dey et al. [49], Pimentel et al. [120], Oliveira et al. [115], and Lim et al. [98] introduced frameworks that are based on provenance models capable of handling both the prospective and retrospective provenance. We evaluate these approaches individually. First, Dey et al. [49] uses a combination of D-PROV for modeling the prospective provenance and a proprietary Datalog schema for modeling the retrospective provenance. The limitation of this approach is the overhead of maintaining two provenance models and the mapping relation between them. The approach from Lim et al. [98] and Oliveira et al. [115] are similar to our approach. However, in terms of long-term sustainability, our approach employs the RDF data model for serializing the ProvONE graphs with support to SPARQL for querying and retrieval of provenance. The prime difference between our approach and the one proposed by Oliveira et al. [115] is the technique in which the prospective provenance is captured. In our approach, we automate the translation of workflow specifications to ProvONE using the Prov2ONE algorithm, whereas in the approach presented by Oliveira et al. [115], the provenance database schema is mapped to ProvONE. The limitation in this approach is the level of details captured by the mapping of prospective provenance from the provenance stored in PostgreSQL to ProvONE. Any changes to the PostgreSQL provenance schema will render the mappings and their implementing adapters obsolete. Moreover, as the mappings do not show how the SeqCtrlLink class of ProvONE is mapped, it is not clear whether the data flow structure and the execution order defined in the original workflow is also translated to the ProvONE prospective provenance. Finally, in their approach, it is not stated whether control-flow languages such as BPEL are also handled. In our approach, as we are entirely based on the workflow specifications, the exact workflow execution order and the potential data-flow structure with the maximum level of granularity is translated to the ProvONE prospective provenance. Following is a summary of the features of P-PIF based on the above-stated properties.

- Both prospective and retrospective provenance interoperability is supported by employing the ProvONE model.
- For long-term sustainability and adherence to standards, P-PIF is designed based on RDF data model and supports querying through SPARQL. A dedicated SPARQL endpoint is exposed for communities to executes custom queries.
- P-PIP not only supports data-flow workflow languages, but also control-flow workflow languages. For the control-flow languages, the execution order of the processes and the data-flow structure from the original workflow is maintained in the ProvONE prospective provenance.

The Table 5.7 summarizes a comparison of the features of existing provenance interoperability frameworks with the P-PIF.

References	Provenance model	Granularity of interoperability	Storage data model	Querying language	Supported WfMSs	Supported specifications
Braun et al. [24]	OPM	Retrospective	Relational schema	SQL	PASS and PLUS	—
Ding et al. [51]	OPM	Retrospective	RDF	SPARQL	—	—
Altintas et al. [6]	Collaborative Data Model and OPM	Retrospective	—	QLP interface	Taverna, Kepler, WS-VLM	SCUFL and MoML
Plankensteiner et al. [122]	IWIR	Prospective	XML	—	ASKALON, Moteur, P-GRADE, Triana, Pegasus	AGWL, CondorDAG, GWENDIA, SCUFL, TWL, XEN
Dey et al. [49] and Pimentel et al. [120]	D-PROV and Datalog schema	Prospective and Retrospective	Datalog facts	DLV Datalog	noWorkflow and YesWorkflow	Python scripting
Oliveira et al. [115]	ProvONE	Prospective and Retrospective	Prolog facts	Prolog queries	e-Science central and SciCumulus	—
Lim et al. [98]	Extended OPM	Prospective and Retrospective	Relational schema	SQL	VIEW and Workflow designer	—
Missier et al. [111]	Common Provenance Model	Prospective and Retrospective	—	—	Taverna and Kepler	SCUFL and MoML
P-PIF	**ProvONE**	**Prospective and Retrospective**	**RDF**	**SPARQL**	**ApacheODE, Taverna, Kepler**	**BPEL, SCUFL, MoML**

Table 5.7: Comparison of features of various provenance interoperability frameworks with P-PIF

5.5 Discussion

In a control-flow workflow language like BPEL, the execution-order of the primitive activities is determined by the structure activities. In order to preserve the execution order during the translation of a BPEL workflow into ProvONE, the control-flow design patterns described in Section 5.2.3 are realized in the Prov2ONE algorithm through the various translation rules. Each design pattern in BPEL has its corresponding ProvONE pattern in the ProvONE prospective graph. For example, the bpel:invoke activities contained in a bpel:sequence or a bpel:flow activity determines the sequential or parallel linking of the corresponding provone:process nodes. The conditional expression defined in the bpel:if or bpel:while activity is copied to the provone:seqctrllink node that is annotated with an explicit tag *condition-process*. The assigning of tags is beneficial for discriminating the seqctrllink nodes from the ones with a condition, and for defining queries aimed at retrieving prospective provenance consisting a specific condition. There were few challenges when translating event and timeout activities. The bpel:onMessage event activity is mapped to provone:process, however, children activities contained within it are treated as a nested workflow. Hence, the bpel:pick activity encompassing the bpel:onMessage activities is necessary for creating a parallel structure in the ProvONE graph. The bpel:scope activity represents a nested ProvONE graph, and is identified by a provone:workflow node. The BPEL system exceptions that occur during the execution of a workflow are handled as retrospective provenance, i.e., whenever an activity fails to execute, the cause of its failure is recorded in the provone:processexec node. The events from the Management API of ApacheODE are adequate to complete the ProvONE retrospective provenance. However, the two associations namely provone:dataonlink and provone:wasassociatedwith have to be derived with the help of prospective provenance. For connecting the dataOnLink association, the ns:message contents are matched with the variables in provone:outputport and provone:inputport and the connecting provone:datalink is retrieved. For this, we have implemented SPARQL queries that perform this matching operation. Similarly, for connecting the ProcessesExec node to its corresponding Process node via the wasAssociatedWith association, the ns:name is matched to the dc:title in Process node.

As SCUFL is data-driven workflow language, all the SCUFL elements were a straightforward match with ProvONE, but what needs to be highlighted is the creation of the provone:seqctrllink. In the case of Prov2ONE:SCUFL algorithm when there is a scufl:datalink between two scufl:processors, the algorithm also creates a provone:seqctrllink between the corresponding provone:process nodes, this helps to determine the execution order of the processes. For completing the ProvONE graph with retrospective provenance, the translations from PROV to ProvONE in the Taverna Adapter are also a straightforward implementation. However, the connection between the prospective and retrospective parts is a challenging task. For this, we reused the SPARQL queries from the ApacheODE adapter that creates the

provone:wasassociatedwith association based on the matching between dc:title attribute in the provone:process node and the title attribute in the retrospective provenance. Similarly, the provone:dataonlink association is created based on the matching between the provone:datalink from the prospective provenance to provone:entity from the retrospective provenance.

In the case of MoML specification, all the MoML elements are translated to ProvONE. However, as the ports (input and output) are not declared explicitly, the Prov2ONE:MoML algorithm handles the input ports and output ports for a moml:actor in two stages. First, the provone:inputports and provone:outputports are created from the ports declared within a moml:actor. Second, the algorithm determines the remaining input ports and output ports for a moml:actor from the moml:link element. The provone:seqctrllink is created based on the moml:relation observed in the moml:link. As Kepler exports provenance in PROV model, the handling of retrospective provenance in the Kepler Adapter is similar to the translation in the Taverna Adapter.

The automated collection of comprehensive provenance (prospective and retrospective) in a global provenance model, such as ProvONE allows querying and analysis of provenance traces generated by different workflow systems. The various queries posed by the provenance challenge and the six extension queries for retrieving the ProvONE modeled provenance enable researchers to compare workflows defined in different specifications, analyze provenance captured by heterogeneous WfMSs at different levels of granularity, and for validating researchers' claims allow reproducibility of results in a different workflow environment. Moreover, as ProvONE allows the modeling of multiple versions of a workflow, it is possible to trace the evolution of a workflow and examine the variations among the workflow versions.

Compared to the existing provenance interoperability frameworks, P-PIF offers a completely automated solution that is based on W3C standard RDF data model with support to querying using SPARQL. Adhering to the existing standards, the long-term sustainability and reuse of P-PIF is guaranteed.

5.6 Summary

In this chapter, we presented P-PIF, a provenance interoperability framework based on the ProvONE model. The P-PIF provides automated collection, modeling and storing of provenance collected from various WfMSs in the ProvONE model. To automatically model the different workflow specifications in ProvONE, Prov2ONE, a graph drawing algorithm that translates workflows specified in either BPEL, SCUFL, or MoML into the corresponding ProvONE prospective graphs was designed. The goal of these algorithms is to preserve the control-flow structure (execution order) and the data-flow structure from the original workflow in the ProvONE prospective provenance. For this, in Tables 5.1 to 5.3, we presented the translation rules between the workflow specifications and the ProvONE model. These translation rules are the

basis for the Prov2ONE algorithms. In the case of BPEL workflows, to verify that the translation rules are valid and the Prov2ONE algorithms can recursively handle nested workflows, the proof for correctness and completeness for each algorithm is presented. The four cases described in the correctness proof of the Prov2ONE: BPEL algorithm shows that all the control-flow patterns available in BPEL are considered, and are translated to their corresponding representation in ProvONE prospective provenance. For the Prov2ONE: SCUFL and Prov2ONE: MoML algorithms, the correctness proof is simple, as it is necessary only to recurse on the nested workflow structures that are identified by a dataflow activity in SCUFL, or a composite actor in MoML respectively. In the second step, to complete the translation of provenance from WfMSs to ProvONE, in Tables 5.4 to 5.6, we provide the translation rules between the retrospective provenance exported by the WfMSs and the ProvONE retrospective provenance. These rules are implemented in the WfMS-specific adapters.

With the availability of provenance from heterogeneous WfMSs in the ProvONE model, and the six queries allows researchers to (a) compare and analyze workflows defined in different specifications, (b) validate scientific results by reproducing them in a different execution environment, (c) trace workflow evolution, (d) collectively query workflow results (retrospective provenance) along with the associated workflow definition (prospective provenance).

To illustrate the applicability of the Prov2ONE algorithms, ProvONE prospective graphs for workflows defined in BPEL, SCUFL, and MoML were created. The complete provenance graph for the nanoscopy workflow is assessed with the six queries designed considering the ProvONE model.

Chapter 6

Conclusions and Future Work

SDRs have become more than a mere storage unit for storing and accessing scientific datasets. With the advent of novel data acquisition systems and their capability to produce enormous amounts of data, there is an exponential growth in the volume of data and metadata that is generated by scientific experiments. Moreover, the corresponding data processing techniques have also changed and new tools for allowing complex and efficient data processing have emerged. This rapid progress has led research communities to demand comprehensive data management solutions that can support them in handling the entire scientific data lifecycle. Hence, a significant number of research communities have deployed SDRs in their institutes. In general, a vital aspect of SDRs is its capability in handling metadata required in orchestrating the data lifecycle. Critical data management tasks such as data discovery, querying and analysis, data description enrichment with annotations, workflow and provenance for data repeatability and reproducibility, long-term preservation, and data structure description for organizing complex data objects are based entirely on metadata. This has motivated research in the field of metadata management systems for handling scientific metadata, and both open-source, as well as proprietary metadata solutions, have been developed.

In this thesis, we invested our efforts in three research areas, namely, SDR architecture, metadata management systems, and provenance handling in WfMS with provenance interoperability among heterogeneous WfMS. To systematically order the contributions of this thesis, first, we focused on the designing a generic SDR architecture. Out of the various features an SDR offers in handling scientific data lifecycle, we focused on two critical features: (a) metadata management capabilities of an SDR, (b) provenance handling and interoperability for WfMSs.

6.1 Summary

Currently, there exist multiple SDR solutions that aim at providing a complete data and metadata management solution for researchers. However, these solutions suffer from many limitations such as support for handling large datasets in the range of TBs and PBs with the possibility of efficient data transfer among different geographically distributed systems that are used within the framework of the discipline. Hence, to

overcome this limitation, we proposed and realized a flexible SDR architecture that is extensible for handling new as well as changing requirements from the research communities. To show that multiple research communities can use such an SDR for different requirements, we realized three SDR systems based on this architecture, namely the nanoscopy open reference data repository, the radiation therapy data repository, and the eCodicology data repository.

However, realizing an SDR architecture was only the first step towards efficient handling and long-term preservation of data. To realize the SDR architecture as a comprehensive solution, we devoted our efforts for the topic of scientific metadata management. In the research area of scientific metadata management, there are multiple open-source as well as proprietary solutions that aim at providing a metadata management solution for scientific metadata models. In general, these solutions have a common limitation, that is, they are designed considering few specific metadata models. Such solutions vastly limit their adoption for communities that use domain-specific metadata schema for modeling their metadata. Adopting these solutions compels the research community to either implement semantic translators for their existing metadata model to comply with the one supported by the SDR or entirely discard their metadata model and adopt the one supported by the SDR. To overcome this limitation, we proposed and implemented the MetaStore framework. The highlight feature of the MetaStore framework is its ability to support any metadata model that is serialized in XML format. The MetaStore framework eliminates the redundant software development cycle undertaken in updating the framework in supporting the new metadata model. For automating the creation of services necessary in handling the registered metadata model, the MetaStore follows the principle of Component-based design with the dynamic composition design pattern. On the one hand, by applying the dynamic composition design principle, we are able to make the MetaStore architecture adaptive, but on the contrary, it was necessary to design the MetaStore on a database system that supports flexible data storage model. For this, we used ArangoDB database that offers a flexible data storage model. Integrating a NoSQL database system with MetaStore allowed us to store and query ad hoc metadata models by modeling them in the appropriate data model.

With the availability of the MetaStore framework, we can handle arbitrary metadata models under the categories of administrative, descriptive, structural, and technical metadata. However, as we aimed to cover all the metadata categories, our study led us to the topic of provenance metadata management in WfMSs. During our research in workflow provenance, we realized that in collaborative research, where research groups have adopted multiple WfMSs that support a different provenance model, the possibility of sharing, analyzing, validating, and reproducing results is a challenging task. These limitations motivated us to extend our research in provenance in WfMS to provenance interoperability among multiple WfMSs. First, we presented the architecture of provenance management for a BPEL-based WfMS, where we showed that for a control-flow language like BPEL the provenance can be automatically translated

into the ProvONE model. To ensure that the complete provenance is captured in ProvONE, we applied a two step process: (1) we designed the Prov2ONE algorithm to translate the exact workflow execution order defined in the BPEL workflow to the ProvONE prospective provenance, (2) we enriched this ProvONE prospective provenance with retrospective provenance by translating the events generated during the enactment of a workflow. For long-term sustainability with adherence to the standard data model and query language, the ProvONE graphs are persisted in an RDF triple database. To the best of our knowledge, we are the first ones who have enabled provenance handling in ProvONE for a BPEL-based WfMS. As our approach is based on the Prov2ONE algorithm for translating the BPEL specification into ProvONE prospective provenance, it can be reused for any WfMS that supports BPEL workflows. However, the collection, translation, and enrichment of ProvONE prospective provenance with retrospective provenance are WfMS specific.

Second, we continued our efforts on the topic of provenance interoperability. Majority of the available provenance interoperability solutions are focused on enabling only retrospective provenance interoperability, with a couple of solutions, namely from Oliveira et al. [115] and Lim et al. [98], that aim at providing both prospective and retrospective provenance interoperability. Compared to these two solutions, the P-PIF framework presented in this thesis is based on the mappings between workflow specification and ProvONE that are realized in the Prov2ONE algorithm. We extended the Prov2ONE algorithm for the SCUFL and MoML specifications. Through the Prov2ONE algorithm, we guarantee the translation of a complete workflow with the same level of details defined in its native representation into ProvONE prospective provenance. Moreover, the exact execution order and the data-flow structure defined in the original workflow is maintained in ProvONE prospective provenance.

Finally, we evaluated each framework presented in this thesis with the appropriate evaluation criteria. First, we evaluated the integration of the GCS API with the exemplary nanoscopy SDR. The data transfer rates using the WebDAV protocol was tested. Here, we showed that automatically optimizing the GCS API leads to better transfer rates. Second, for the MetaStore framework, we presented two types of evaluations: (a) a feature based evaluation stating the advantages of MetaStore against existing metadata management solutions, (b) a performance study, comparing the read and write performance of MetaStore for two different database systems. Third, similar to the evaluation of the MetaStore, for the P-PIF framework we presented two types of evaluations: (a) to verify the correctness and completeness of the Prov2ONE algorithm, we explained the various workflow design patterns that can be constructed in a BPEL workflow, followed by their corresponding representation in ProvONE, (b) a feature based comparison of the P-PIF framework against existing provenance interoperability systems.

6.2 Future Work

During our research in scientific metadata and provenance management, we came up with new ideas that could extend our research. Following we briefly describe them.

- **Provenance graph mining.** Until now the purpose of capturing provenance in WfMS was to systematically document each step executed in a workflow to reproduce the scientific result. However, with the availability of provenance from various WfMSs in a common model like ProvONE, we recommend mining of these provenance graphs for extracting common workflow patterns. Based on a set of extracted patterns, researchers can not only improve their existing workflows but also detect erroneous workflows that are bound to fail.

- **Visualization of provenance graph.** In this thesis, we focused on building a complete metadata and provenance interoperability framework. The next step is to extend this framework with a visualization framework. We recommend exposing the SPARQL queries with a visualization framework like D3.js. A visualization framework will not only help researchers to navigate workflows, and their provenance traces intuitively but also assist in runtime tracking and debugging of the workflows.

- **Combining metadata, annotations with provenance.** We showed that the RDF data model is efficiently applied for storing both the ProvONE provenance graphs as well as the annotations modeled in the WADM. For future work, we recommend defining the semantic relationships between these two data models. With the semantics established, queries that would provide the researchers with domain-specific information could be formulated. There are various applications in research domain of linked data that could use annotations, provenance, and metadata in combination. For example, based on the annotations tagged to data, the workflow that is responsible for generating this data can be retrieved, or based on data that is tagged as rejected or accepted, the corresponding workflows could be marked obsolete or in vogue.

- **Workflow interoperability.** In this thesis, we focused on translating the potential data-flow and the execution order from the original workflow into ProvONE. The data types and the model of computation were not considered during the translation. For future work, we propose a framework for translation between heterogeneous workflow specifications. This framework could be an extension to the P-PIF framework. For this, we recommend the following approach. First, the syntax and semantics of the ProvONE model have to be defined using formal process algebra. Since the syntax and semantics of BPEL and SCUFL are formally defined using π-calculus and computational λ-calculus, it is necessary to define the syntax and semantics of ProvONE using π-calculus. Second, to prove workflow equivalence for translating the workflow specifications, the big-step semantics can be utilized.

Bibliography

[1] Daniel J. Abadi, Peter A. Boncz, and Stavros Harizopoulos. Column-oriented Database Systems. *Proc. VLDB Endow.*, 2(2):1664–1665, August 2009. ISSN 2150-8097.

[2] William Allcock, Joe Bester, John Bresnahan, Ann Chervenak, Lee Liming, and Steve Tuecke. GridFTP: Protocol extensions to FTP for the Grid. *Global Grid ForumGFD-RP*, 20:1–21, 2003.

[3] Julie Allinson, Sebastien François, and Stuart Lewis. SWORD: Simple Web-service offering repository deposit. *Ariadne*, (54), 2008.

[4] İlkay Altıntaş. *Collaborative provenance for workflow-driven science and engineering*. 2011.

[5] Ilkay Altintas, Oscar Barney, and Efrat Jaeger-Frank. Provenance collection support in the kepler scientific workflow system. In *International Provenance and Annotation Workshop*, pages 118–132. Springer, 2006.

[6] Ilkay Altintas, Manish Kumar Anand, Daniel Crawl, Shawn Bowers, Adam Belloum, Paolo Missier, Bertram Ludäscher, Carole A. Goble, and Peter M. A. Sloot. *Understanding Collaborative Studies through Interoperable Workflow Provenance*, pages 42–58. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[7] Ricardo Carvalho Amorim, João Aguiar Castro, João Rocha da Silva, and Cristina Ribeiro. *A Comparative Study of Platforms for Research Data Management: Interoperability, Metadata Capabilities and Integration Potential*, pages 101–111. Springer International Publishing, 2015.

[8] Manish Kumar Anand, Shawn Bowers, and Bertram Ludäscher. Techniques for efficiently querying scientific workflow provenance graphs. In *EDBT*, volume 10, pages 287–298, 2010.

[9] Renzo Angles and Claudio Gutierrez. Survey of Graph Database Models. *ACM Comput. Surv.*, 40(1):1:1–1:39, February 2008.

[10] Massimiliano Assante, Leonardo Candela, Donatella Castelli, and Alice Tani. Are scientific data repositories coping with research data publishing? *Data Science Journal*, 15, 2016.

[11] Chris Awre, Tom Cramer, Richard Green, Lynn McRae, Bess Sadler, Tim Sigmon, Thornton Staples, and Ross Wayland. Project hydra: Designing & building a reusable framework for multipurpose, multifunction, multi-institutional repository-powered solutions. Georgia Institute of Technology, 2009.

[12] Keith Baggerly. Disclose all data in publications. *Nature*, 467(7314):401–401, 2010.

[13] Charles W Bailey, Karen Coombs, Jill Emery, Anne Mitchell, Chris Morris, Spencer Simons, and Robert Wright. Spec kit 292 institutional repositories. 2006.

[14] Alexander Ball, Sean Chen, Jane Greenberg, Cristina Perez, Keith Jeffery, and Rebecca Koskela. Building a disciplinary metadata standards directory. *International Journal of Digital Curation*, 9(1):142–151, 2014.

[15] Roger S. Barga and Luciano A. Digiampietri. Automatic capture and efficient storage of e-Science experiment provenance. *Concurrency and Computation: Practice and Experience*, 20(5):419–429, 2008.

[16] Chaitanya Baru, Reagan Moore, Arcot Rajasekar, and Michael Wan. The SDSC storage resource broker. In *Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research*, page 5. IBM Press, 1998.

[17] Neil Beagrie. Digital curation for science, digital libraries, and individuals. *International Journal of Digital Curation*, 1(1):3–16, 2008.

[18] Oren Ben-Kiki, Clark Evans, and Brian Ingerson. YAML Ain't Markup Language (YAML™) Version 1.1. *yaml. org, Tech. Rep*, 2005.

[19] Chad Berkley, Matthew Jones, Jivka Bojilova, and Daniel Higgins. Metacat: a schema-independent XML database system. In *Scientific and Statistical Database Management, 2001. SSDBM 2001. Proceedings. Thirteenth International Conference on*, pages 171–179. IEEE, 2001.

[20] Deepavali Bhagwat, Laura Chiticariu, Wang-Chiew Tan, and Gaurav Vijayvargiya. An Annotation Management System for Relational Databases. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 900–911. VLDB Endowment, 2004. ISBN 0-12-088469-0.

[21] Tobias Blanke, Mark Hedges, and Stuart Dunn. Arts and humanities e-science—Current practices and future challenges. *Future Generation Computer Systems*, 25(4):474 – 480, 2009.

[22] Alexandru Boicea, Florin Radulescu, and Laura Ioana Agapin. Mongodb vs Oracle–database comparison. In *Emerging Intelligent Data and Web Technologies (EIDWT), 2012 Third International Conference on*, pages 330–335. IEEE, 2012.

[23] Rajendra Bose and James Frew. Lineage Retrieval for Scientific Data Processing: A Survey. *ACM Comput. Surv.*, 37(1):1–28, March 2005.

[24] Uri Braun, Margo I Seltzer, Adriane Chapman, Barbara T Blaustein, M David Allen, and Len Seligman. Towards Query Interoperability: PASSing PLUS. In *TaPP*, pages 1–10, 2010.

[25] Eric A Brewer. Towards robust distributed systems. In *PODC*, volume 7, 2000.

[26] Peter Buneman and Wang-Chiew Tan. Provenance in Databases. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 1171–1173, New York, NY, USA, 2007. ACM.

[27] Linda Cantara. METS: The Metadata Encoding and Transmission Standard. *Cataloging & Classification Quarterly*, 40(3-4):237–253, 2005.

[28] B. Cao, B. Plale, G. Subramanian, E. Robertson, and Y. Simmhan. Provenance Information Model of Karma Version 3. In *2009 Congress on Services - I*, pages 348–351, July 2009.

[29] Rick Cattell. Scalable SQL and NoSQL Data Stores. *SIGMOD Rec.*, 39(4): 12–27, May 2011.

[30] Digital Curation Center. Disciplinary Metadata. http://www.dcc.ac.uk/drupal/resources/metadata-standards. Accessed on: 2017-04-30.

[31] Swati Chandna, Danah Tonne, Thomas Jejkal, Rainer Stotzka, Celia Krause, Philipp Vanscheidt, Hannah Busch, and Ajinkya Prabhune. Software workflow for the automatic tagging of medieval manuscript images (SWATI). In *DRR*, page 940206, 2015.

[32] Swati Chandna, Francesca Rindone, Carsten Dachsbacher, and Rainer Stotzka. Quantitative exploration of large medieval manuscripts data for the codicological research. In *Large Data Analysis and Visualization (LDAV), 2016 IEEE 6th Symposium on*, pages 20–28. IEEE, 2016.

[33] Ben Clifford, Ian Foster, Jens-S. Voeckler, Michael Wilde, and Yong Zhao. Tracking provenance in a virtual data grid. *Concurrency and Computation: Practice and Experience*, 20(5):565–575, 2008.

[34] Leendert D Couprie. Iconclass: an iconographic classification system. *Art Libraries Journal*, 8(2):32–49, 1983.

[35] Christoph Cremer. *Optics Far Beyond the Diffraction Limit*, pages 1359–1397. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[36] Christoph Cremer, Rainer Kaufmann, Manuel Gunkel, Sebastian Pres, Yanina Weiland, Patrick Müller, Thomas Ruckelshausen, Paul Lemmer, Fania Geiger, Sven Degenhard, Christina Wege, Niels A. W. Lemmermann, Rafaela Holtappels, Hilmar Strickfaden, and Michael Hausmann. Superresolution imaging of biological nanostructures by spectral precision distance microscopy. *Biotechnology Journal*, 6(9):1037–1051, 2011.

[37] Víctor Cuevas-Vicenttín, Parisa Kianmajd, Bertram Ludäscher, Paolo Missier, Fernando Chirigati, Yaxing Wei, David Koop, and Saumen Dey. The PBase scientific workflow provenance repository. *International Journal of Digital Curation*, 9(2):28–38, 2014.

[38] Y. Cui and J. Widom. Practical lineage tracing in data warehouses. In *Proceedings of 16th International Conference on Data Engineering (Cat. No.00CB37073)*, pages 367–378, 2000.

[39] William Culhane, Kirill Kogan, Chamikara Jayalath, and Patrick Eugster. LOOM: Optimal Aggregation Overlays for In-Memory Big Data Processing. In *HotCloud*, 2014.

[40] Daniel D. Gutierrez. InsideBIGDATA Guide to Scientific Research. http://insidebigdata.com/2015/12/01/insidebigdata-guide-to-scientific-research/, 2015. Accessed on: 2017-04-30.

[41] Sérgio Manuel Serra da Cruz, Maria Luiza M Campos, and Marta Mattoso. Towards a Taxonomy of Provenance in Scientific Workflow Management Systems. In *Services-I, 2009 World Conference on*, pages 259–266. IEEE, 2009.

[42] Susan B. Davidson and Juliana Freire. Provenance and Scientific Workflows: Challenges and Opportunities. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1345–1350, New York, NY, USA, 2008. ACM.

[43] Domingo De Abreu, Alejandro Flores, Guillermo Palma, Valeria Pestana, José Pinero, Jonathan Queipo, José Sánchez, and Maria-Esther Vidal. Choosing Between Graph Databases and RDF Engines for Consuming and Mining Linked Data. In *Proceedings of the Fourth International Conference on Consuming Linked Data-Volume 1034*, pages 37–49. CEUR-WS.org, 2013.

[44] D. de Oliveira, E. Ogasawara, F. Baião, and M. Mattoso. SciCumulus: A Lightweight Cloud Middleware to Explore Many Task Computing Paradigm in Scientific Workflows. In *2010 IEEE 3rd International Conference on Cloud Computing*, pages 378–385, July 2010.

[45] Ewa Deelman, Gurmeet Singh, Malcolm P Atkinson, Ann Chervenak, NP Chue Hong, Carl Kesselman, Sonal Patil, Laura Pearlman, and Mei-Hui Su. Grid-based metadata services. In *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*, pages 393–402. IEEE, 2004.

[46] Ewa Deelman, Bruce Berriman, Ann Chervenak, Oscar Corcho, Paul Groth, and Luc Moreau. Metadata and provenance management. *In Scientific Data Management: Challenges, Technology, and Deployment, First Edition*, 2009.

[47] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J. Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira da Silva, Miron Livny, and Kent Wenger. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, 46:17 – 35, 2015.

[48] Lorcan Dempsey. Scientific, industrial, and cultural heritage: a shared approach. *Ariadne*, (22), 1999.

[49] Saumen Dey, Khalid Belhajjame, David Koop, Meghan Raul, and Bertram Ludäscher. Linking prospective and retrospective provenance in scripts. *Theory and Practice of Provenance (TaPP)*, 2015.

[50] Ivica Dimitrovski, Dragi Kocev, Suzana Loskovska, and Sašo Džeroski. Hierarchical Annotation of Medical Images. *Pattern Recognition*, 44(10–11):2436 – 2449, 2011. Semi-Supervised Learning for Visual Content Analysis and Understanding.

[51] Li Ding, James Michaelis, Jim McCusker, and Deborah L. McGuinness. Linked provenance data: A semantic Web-based approach to interoperable workflow traces. *Future Generation Computer Systems*, 27(6):797 – 805, 2011.

[52] Kelvin K. Droegemeier, Dennis Gannon, Daniel Reed, Beth Plale, Jay Alameda, Tom Baltzer, Keith Brewster, Richard Clark, Ben Domenico, Sara Graves, Everette Joseph, Donald Murray, Rahul Ramachandran, Mohan Ramamurthy, Lavanya Ramakrishnan, John A. Rushing, Daniel Weber, Robert Wilhelmson, Anne Wilson, Ming Xue, and Sepideh Yalda. Service-Oriented Environments for Dynamically Interacting with Mesoscale Weather. *Computing in Science & Engineering*, 7(6):12–29, 2005.

[53] Tommy Ellqvist, David Koop, Juliana Freire, Cláudio Silva, and Lena Strömbäck. Using Mediation to Achieve Provenance interoperability. In *Services-I, 2009 World Conference on*, pages 291–298. IEEE, 2009.

[54] Wolfgang Emmerich, Ben Butchart, Liang Chen, Bruno Wassermann, and Sarah L. Price. Grid Service Orchestration Using the Business Process Execution Language (BPEL). *Journal of Grid Computing*, 3(3):283–304, 2005.

[55] Damian Flannery, Brian Matthews, Tom Griffin, Juan Bicarregui, Michael Gleaves, Laurent Lerusse, Roger Downing, Alun Ashton, Shoaib Sufi, Glen Drinkwater, et al. ICAT: Integrating data infrastructure for facilities based science. In *e-Science, 2009. e-Science'09. Fifth IEEE International Conference on*, pages 201–207. IEEE, 2009.

[56] Ira R Forman and Nate Forman. Java Reflection in Action., 2004.

[57] Juliana Freire, Cláudio T. Silva, Steven P. Callahan, Emanuele Santos, Carlos E. Scheidegger, and Huy T. Vo. *Managing Rapidly-Evolving Scientific Workflows*, pages 10–18. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[58] Juliana Freire, David Koop, Emanuele Santos, and Cláudio T Silva. Provenance for Computational Tasks: A Survey. *Computing in Science & Engineering*, 10 (3), 2008.

[59] LM Gadelha, B Clifford, M Mattoso, M Wilde, I Foster, et al. Provenance management in Swift with implementation details. Technical report, Argonne National Laboratory (ANL), 2011.

[60] Helena Galhardas, Daniela Florescu, Dennis Shasha, Eric Simon, and Cristian-Augustin Saita. Improving Data Cleaning Quality Using a Data Lineage Facility. In *DMDW*, page 3, 2001.

[61] Daniel Garijo and Yolanda Gil. A New Approach for Publishing Workflows: Abstractions, Standards, and Linked Data. In *Proceedings of the 6th Workshop on Workflows in Support of Large-scale Science*, WORKS '11, pages 47–56, New York, NY, USA, 2011. ACM.

[62] Ashish Gehani and Dawood Tariq. SPADe: Support for Provenance Auditing in Distributed Environments. In *Proceedings of the 13th International Middleware Conference*, Middleware '12, pages 101–120, New York, NY, USA, 2012. Springer-Verlag New York, Inc.

[63] André Giesler, Myriam Czekala, Björn Hagemeier, and Richard Grunzke. Uniprov: A Flexible Provenance Tracking System for UNICORE. In *Jülich Aachen Research Alliance (JARA) High-Performance Computing Symposium*, pages 233–242. Springer, 2016.

[64] Seth Gilbert and Nancy Lynch. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services. *SIGACT News*, 33(2): 51–59, June 2002.

[65] Carole Goble. Position statement: Musings on provenance, workflow and (semantic web) annotations for bioinformatics. In *Workshop on Data Derivation and Provenance, Chicago*, volume 3, 2002.

[66] Antoon Goderis, Christopher Brooks, Ilkay Altintas, Edward A. Lee, and Carole Goble. *Composing Different Models of Computation in Kepler and Ptolemy II*, pages 182–190. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[67] Jeremy Goecks, Anton Nekrutenko, and James Taylor. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8):R86, 2010.

[68] Jennifer Golbeck and James Hendler. A Semantic Web approach to the provenance challenge. *Concurrency and Computation: Practice and Experience*, 20 (5):431–439, 2008.

[69] Trey Grainger, Timothy Potter, and Yonik Seeley. *Solr in action*. Manning Cherry Hill, 2014.

[70] Jim Gray, David T. Liu, Maria Nieto-Santisteban, Alex Szalay, David J. DeWitt, and Gerd Heber. Scientific Data Management in the Coming Decade. *SIGMOD Rec.*, 34(4):34–41, December 2005.

[71] J. Graybeal, S.P. Miller, and K. Stocks. The MMI Guides: Navigating the World of Marine Metadata. http://uop.whoi.edu/techdocs/presentations/MMI_Guides.pdf, 2010. Accessed on: 2017-07-30.

[72] Paul Groth, Sheng Jiang, Simon Miles, Steve Munroe, Victor Tan, Sofia Tsasakou, and Luc Moreau. An Architecture for Provenance Systems. February 2006. URL `https://eprints.soton.ac.uk/263216/`.

[73] Richard Grunzke, Volker Hartmann, Thomas Jejkal, Helen Kollai, Ajinkya Prabhune, et al. The MASi Repository Service - Comprehensive, Metadata-driven and Multi-community Research Data Management. *Future Generation Computer Systems*, in press.

[74] Brooks Hanson, Andrew Sugden, and Bruce Alberts. Making data maximally available. *Science*, 331(6018):649–649, 2011.

[75] Olaf Hartig and Jun Zhao. Using Web Data Provenance for Quality Assessment. In *Proceedings of the First International Conference on Semantic Web in Provenance Management - Volume 526*, SWPM'09, pages 29–34, Aachen, Germany, 2009. CEUR-WS.org.

[76] Mark Hedges, Adil Hasan, and Tobias Blanke. Management and Preservation of Research Data with iRODS. In *Proceedings of the ACM First Workshop on CyberInfrastructure: Information Management in eScience*, CIMS '07, pages 17–22, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-831-2.

[77] Maurice P. Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, July 1990.

[78] Anthony JG Hey and Anne E Trefethen. The data deluge: An e-science perspective. 2003.

[79] Tony Hey and Anne E. Trefethen. Cyberinfrastructure for e-science. *Science*, 308(5723):817–821, 2005.

[80] Tony Hey, Stewart Tansley, Kristin M Tolle, et al. *The fourth paradigm: data-intensive scientific discovery*, volume 1. Microsoft research Redmond, WA, 2009.

[81] Sarah Higgins. Using Metadata Standards. http://www.dcc.ac.uk/resources/briefing-papers/standards-watch-papers/using-metadata-standards, February 2007. Accessed on: 2017-04-30.

[82] R. N. Hook, M. Romaniello, M. Ullgrén, P. Järveläinen, S. Maisala, T. Oittinen, V. Savolainen, O. Solin, J. Tyynelä, M. Peron, C. Izzo, and T. Licha. *ESO Reflex: A Graphical Workflow Engine for Running Recipes*, pages 169–175. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[83] Iain Hrynaszkiewicz, Melissa L Norton, Andrew J Vickers, and Douglas G Altman. Preparing raw clinical data for publication: guidance for journal editors, authors, and peer reviewers. *Trials*, 11(1):9, 2010.

[84] Bo Hu, S. Dasmahapatra, P. Lewis, and N. Shadbolt. Ontology-based medical image annotation with description logics. In *Proceedings. 15th IEEE International Conference on Tools with Artificial Intelligence*, pages 77–82, Nov 2003.

[85] Thomas Jejkal, Alexander Vondrous, Andreas Kopmann, Rainer Stotzka, and Volker Hartmann. Kit data manager: The repository architecture enabling cross-disciplinary research. *Large-Scale Data Management and Analysis-Big Data in Science-*, 2014.

[86] Scott Jensen, Devarshi Ghoshal, and Beth Plale. Evaluation of two xml storage approaches for scientific metadata. *Indiana University Dept of Computer Science Tech Report*, 698, 2011.

[87] Richard Jones, Theo Andrew, and John (John A.) MacColl. *The institutional repository*. Oxford Chandos Publishing, 2006. URL `http://swb.eblib.com/patron/FullRecord.aspx?p=1640125`.

[88] L. M. R. Gadelha Jr and M. Mattoso. Kairos: An Architecture for Securing Authorship and Temporal Information of Provenance Data in Grid-Enabled Workflow Management Systems. In *2008 IEEE Fourth International Conference on eScience*, pages 597–602, Dec 2008.

[89] Jihie Kim, Ewa Deelman, Yolanda Gil, Gaurang Mehta, and Varun Ratnakar. Provenance trails in the Wings/Pegasus system. *Concurrency and Computation: Practice and Experience*, 20(5):587–597, 2008.

[90] Cory Knobel. *Understanding infrastructure: Dynamics, tensions, and design.* Academic Press, 2007.

[91] Holger Knublauch, JA Hendler, and Kingsley Idehen. SPIN-overview and motivation. W3C member submission. *World Wide Web Consortium (February 2011)*, 2011.

[92] Steven Krauwer. CLARIN: Common language resources and technology infrastructure. *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, 2008.

[93] Betty Landesman. Seeing Standards: A Visualization of the Metadata Universe. *Technical Services Quarterly*, 28(4):459–460, 2011. URL `http://www.dlib.indiana.edu/~jenlrile/metadatamap/`.

[94] Brian F Lavoie. *The open archival information system reference model: Introductory guide*, volume 1. OCLC, 2004.

[95] Damien Lecarpentier, Peter Wittenburg, Willem Elbers, Alberto Michelini, Riam Kanso, Peter Coveney, and Rob Baxter. EUDAT: A new cross-disciplinary data infrastructure for science. *International Journal of Digital Curation*, 8(1): 279–287, 2013.

[96] Edward A Lee and Steve Neuendorffer. *MoML: A Modeling Markup Language in SML: Version 0.4.* 2000.

[97] Y. Li and S. Manoharan. A performance comparison of SQL and NoSQL databases. In *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, pages 15–19, Aug 2013.

[98] C. Lim, S. Lu, A. Chebotko, and F. Fotouhi. Prospective and Retrospective Provenance Collection in Scientific Workflow Environments. In *2010 IEEE International Conference on Services Computing*, pages 449–456, July 2010.

[99] Philip Lord, Alison Macdonald, Liz Lyon, and David Giaretta. From data deluge to data curation. In *Proceedings of the UK e-science All Hands meeting*, pages 371–375, 2004.

[100] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[101] Clifford A Lynch. Institutional repositories: essential infrastructure for scholarship in the digital age. *portal: Libraries and the Academy*, 3(2):327–336, 2003.

[102] Clifford A Lynch and Joan K Lippincott. Institutional repository deployment in the united states as of early 2005. *D-lib Magazine*, 11(9):1–11, 2005.

[103] D. Maksimov, J. Hesser, C. Brockmann, S. Jochum, T. Dietz, A. Schnitzer, C. Duber, S. O. Schoenberg, and S. Diehl. Graph-matching based cta. *IEEE Transactions on Medical Imaging*, 28(12):1940–1954, Dec 2009.

[104] Laura Haak Marcial and Bradley M. Hemminger. Scientific data repositories on the Web: An initial survey. *Journal of the American Society for Information Science and Technology*, 61(10):2029–2048, 2010.

[105] Suresh Marru, Lahiru Gunathilake, Chathura Herath, Patanachai Tangchaisin, Marlon Pierce, Chris Mattmann, Raminder Singh, Thilina Gunarathne, Eran Chinthaka, Ross Gardler, Aleksander Slominski, Ate Douma, Srinath Perera, and Sanjiva Weerawarana. Apache Airavata: A Framework for Distributed Applications and Computational Workflows. In *Proceedings of the 2011 ACM Workshop on Gateway Computing Environments*, GCE '11, pages 21–28. ACM, 2011. ISBN 978-1-4503-1123-6.

[106] Chris Mattmann and Jukka Zitting. *Tika in Action*. Manning Publications Co., Greenwich, CT, USA, 2011. ISBN 1935182854, 9781935182856.

[107] Rudolf Mayer and Andreas Rauber. Towards Time-resilient MIR Processes. In *ISMIR*, pages 337–342, 2012.

[108] Michael McCandless, Erik Hatcher, and Otis Gospodnetic. *Lucene in Action: Covers Apache Lucene 3.0*. Manning Publications Co., 2010.

[109] Deborah L. McGuinness and Paulo Pinheiro da Silva. Explaining answers from the Semantic Web: the Inference Web approach. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(4):397 – 413, 2004. International Semantic Web Conference 2003.

[110] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng. Composing adaptive software. *Computer*, 37(7):56–64, July 2004.

[111] P. Missier, B. Ludäscher, S. Bowers, S. Dey, A. Sarkar, B. Shrestha, I. Altintas, M. K. Anand, and C. Goble. Linking multiple workflow provenance traces for interoperable collaborative science. In *The 5th Workshop on Workflows in Support of Large-Scale Science*, pages 1–8, Nov 2010.

[112] Luc Moreau and Paolo Missier. PROV-DM: The prov data model. 2013.

[113] Luc Moreau, Bertram Ludäscher, Ilkay Altintas, Roger S Barga, Shawn Bowers, Steven Callahan, George Chin, Ben Clifford, Shirley Cohen, Sarah Cohen-Boulakia, et al. Special issue: The first provenance challenge. *Concurrency and computation: practice and experience*, 20(5):409–418, 2008.

[114] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, et al. The Open Provenance Model Core Specification (v1. 1). *Future generation computer systems*, 27(6):743–756, 2011.

[115] Wellington Oliveira, Paolo Missier, Kary Ocaña, Daniel de Oliveira, and Vanessa Braganholo. *Analyzing Provenance Across Heterogeneous Provenance Graphs*, pages 57–70. Springer International Publishing, 2016.

[116] Heinz Pampel, Paul Vierkant, Frank Scholze, Roland Bertelmann, Maxi Kindling, Jens Klump, Hans-Jürgen Goebelbecker, Jens Gundlach, Peter Schirmbacher, and Uwe Dierolf. Making research data repositories visible: the re3data. org registry. *PloS one*, 8(11):e78080, 2013.

[117] Zachary Parker, Scott Poe, and Susan V. Vrbsky. Comparing NoSQL MongoDB to an SQL DB. In *Proceedings of the 51st ACM Southeast Conference*, ACMSE '13, pages 5:1–5:6, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1901-0.

[118] Quan Pham, Tanu Malik, Ian T Foster, Roberto Di Lauro, and Raffaele Montella. SOLE: Linking Research Papers with Science Objects. In *IPAW*, pages 203–208. Springer, 2012.

[119] Quan Pham, Tanu Malik, and Ian T Foster. Using Provenance for Repeatability. *TaPP*, 13:2, 2013.

[120] João Felipe Pimentel, Saumen Dey, Timothy McPhillips, Khalid Belhajjame, David Koop, Leonardo Murta, Vanessa Braganholo, and Bertram Ludäscher. Yin & yang: demonstrating complementary provenance from noworkflow & yesworkflow. In *International Provenance and Annotation Workshop*, pages 161–165. Springer, 2016.

[121] B Plale, B Cao, and M Aktas. Provenance Capture of Unmanaged Workflows with Karma. *Bloomington, IN, Indiana University*, 2011.

[122] Kassian Plankensteiner, Radu Prodan, Matthias Janetschek, Thomas Fahringer, Johan Montagnat, David Rogers, Ian Harvey, Ian Taylor, Ákos Balaskó, and Péter Kacsuk. Fine-Grain Interoperability of Scientific Workflows in Distributed Computing Infrastructures. *Journal of Grid Computing*, 11(3):429–455, Sep 2013.

[123] Ajinkya Prabhune, Rainer Stotzka, Thomas Jejkal, Volker Hartmann, Margund Bach, Eberhard Schmitt, Michael Hausmann, and Jürgen Hesser. An optimized

generic client service api for managing large datasets within a data repository. In *2015 IEEE First International Conference on Big Data Computing Service and Applications*, pages 44–51, March 2015.

[124] Ajinkya Prabhune, Hasebullah Ansari, Anil Keshav, Rainer Stotzka, Michael Gertz, and Jürgen Hesser. MetaStore: A metadata framework for scientific data repositories. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 3026–3035, Dec 2016.

[125] Ajinkya Prabhune, Aaron Zweig, Rainer Stotzka, Michael Gertz, and Juergen Hesser. Prov2one: An algorithm for automatically constructing provone provenance graphs. In *International Provenance and Annotation Workshop*, pages 204–208. Springer, 2016.

[126] Ajinkya Prabhune, Rainer Stotzka, Michael Gertz, Lei Zheng, and Jürgen Hesser. Managing provenance for medical datasets. *BIOSTEC 2017*, page 236, 2017.

[127] Ajinkya Prabhune, Rainer Stotzka, Sakharkar Vaibhav, Jürgen Hesser, and Michael Gertz. MetaStore: An adaptive metadata management framework for heterogeneous metadata models. *Distributed and Parallel Databases*, in press.

[128] NISO Press. Understanding metadata. *National Information Standards*, 20, 2004.

[129] Dan Pritchett. Base: An acid alternative. *Queue*, 6(3):48–55, May 2008. ISSN 1542-7730.

[130] Eric Prud, Andy Seaborne, et al. SPARQL query language for RDF. 2006.

[131] Jian Qin, Alex Ball, and Jane Greenberg. Functional and architectural requirements for metadata: Supporting discovery and management of scientific data. In *International Conference on Dublin Core and Metadata Applications*, pages 62–71, 2012.

[132] Shrija Rajbhandari and David W Walker. Support for provenance in a service-based computing grid. In *UK e-Science All Hands Meeting*, 2004.

[133] Nick Russell, Arthur HM Ter Hofstede, Wil MP Van Der Aalst, and Nataliya Mulyar. Workflow control-flow patterns: A revised view. *BPM Center Report BPM-06-22, BPMcenter. org*, pages 06–22, 2006.

[134] Satya Sahoo, Paul Groth, Olaf Hartig, Simon Miles, Sam Coppens, James Myers, Yolanda Gil, Luc Moreau, Jun Zhao, Michael Panzer, et al. Provenance vocabulary mappings. *W3C Working Draft, W3C*, 2010.

[135] Satya S Sahoo and Amit P Sheth. Provenir ontology: Towards a framework for escience provenance management. 2009.

[136] Robert Sanderson, Paolo Ciccarese, Herbert Van de Sompel, Shannon Bradshaw, Dan Brickley, Leyla Jael Garcıa Castro, Timothy Clark, Timothy Cole, Phil Desenne, Anna Gerber, et al. Open annotation data model. *W3C community draft*, 8, 2013.

[137] Thomas Schandl and Andreas Blumauer. PoolParty: SKOS thesaurus management utilizing linked data. *The Semantic Web: Research and Applications*, pages 421–425, 2010.

[138] Guido Scherp, André Höing, Stefan Gudenkauf, Wilhelm Hasselbring, and Odej Kao. Using UNICORE and WS-BPEL for Scientific Workflow Execution in Grid Environments. In *Euro-Par Workshops*, pages 335–344. Springer, 2009.

[139] Hartmut Scholz. *Die mittelalterlichen Glasmalereien in Mittelfranken und Nürnberg: extra muros*, volume 10. Deutscher Verlag Fur Kunstwissenschaft, 2002.

[140] Hartmut Scholz. *Die mittelalterlichen Glasmalereien in Nürnberg: Sebalder Stadtseite*. Deutscher Verlag für Kunstwissenschaft, 2013.

[141] Andreas Schreiber, Miriam Ney, and Heinrich Wendel. *The Provenance Store prOOst for the Open Provenance Model*, pages 240–242. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[142] Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David J. DeWitt, and Jeffrey F. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 302–314, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1-55860-615-7. URL `http://dl.acm.org/citation.cfm?id=645925.671499`.

[143] Sarah L. Shreeves, Joanne S. Kaczmarek, and Timothy W. Cole. Harvesting cultural heritage metadata using the oai protocol. *Library Hi Tech*, 21(2):159–169, 2003.

[144] Yogesh Simmhan, Paul Groth, and Luc Moreau. Special Section: The third provenance challenge on using the open provenance model for interoperability. *Future Generation Computer Systems*, 27(6):737 – 742, 2011.

[145] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. A Survey of Data Provenance in e-Science. *SIGMOD Rec.*, 34(3):31–36, September 2005.

[146] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. Query capabilities of the Karma provenance framework. *Concurrency and Computation: Practice and Experience*, 20(5):441–451, 2008.

[147] Gurmeet Singh, Shishir Bharathi, Ann Chervenak, Ewa Deelman, Carl Kesselman, Mary Manohar, Sonal Patil, and Laura Pearlman. A metadata catalog

service for data intensive applications. In *Supercomputing, 2003 ACM/IEEE Conference*, pages 33–33. IEEE, 2003.

[148] Aleksander Slomiski. On using BPEL extensibility to implement OGSI and WSRF Grid workflows. *Concurrency and Computation: Practice and Experience*, 18(10):1229–1241, 2006.

[149] MacKenzie Smith, Mary Barton, Mick Bass, Margret Branschofsky, Greg McClellan, Dave Stuve, Robert Tansley, and Julie Harford Walker. DSpace: An open source dynamic digital repository. 2003.

[150] Catherine Soehner, Catherine Steeves, and Jennifer Ward. E-Science and Data Support Services: A Study of ARL Member Institutions. *Association of Research Libraries*, 2010.

[151] Mirko Sonntag, Dimka Karastoyanova, and Ewa Deelman. *BPEL4Pegasus: Combining Business and Scientific Workflows*, pages 728–729. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[152] R. D. Stevens, H. J. Tipney, C. J. Wroe, T. M. Oinn, M. Senger, P. W. Lord, C. A. Goble, A. Brass, and M. Tassabehji. Exploring Williams–Beuren syndrome using myGrid. *Bioinformatics*, 20:i303, 2004.

[153] Robert D. Stevens, Alan J. Robinson, and Carole A. Goble. myGrid: personalised bioinformatics on the information grid. *Bioinformatics*, 19:i302, 2003.

[154] Victoria Stodden, Peixuan Guo, and Zhaokun Ma. Toward reproducible computational research: an empirical analysis of data and code policy adoption by journals. *PloS one*, 8(6):e67111, 2013.

[155] Rainer Stotzka, Volker Hartmann, Thomas Jejkal, Michael Sutter, Jos van Wezel, Marcus Hardt, Ariel Garcia, Rainer Kupsch, and Serguei Bourov. Perspective of the large scale data facility (lsdf) supporting nuclear fusion applications. In *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on*, pages 373–379. IEEE, 2011.

[156] Achim Streit, Piotr Bala, Alexander Beck-Ratzka, Krzysztof Benedyczak, Sandra Bergmann, Rebecca Breu, Jason Milad Daivandy, Bastian Demuth, Anastasia Eifer, André Giesler, et al. UNICORE 6—Recent and Future Advancements. *Annals of Telecommunications-annales des Télécommunications*, 65(11-12):757–762, 2010.

[157] Shulei Sun, Jing Chen, Weizhong Li, Ilkay Altintas, Abel Lin, Steve Peltier, Karen Stocks, Eric E. Allen, Mark Ellisman, Jeffrey Grethe, and John Wooley. Community cyberinfrastructure for Advanced Microbial Ecology Research and Analysis: the CAMERA resource. *Nucleic Acids Research*, 39(suppl_1):D546, 2011.

[158] Yiming Sun, Scott Jensen, Sangmi Pallickara, and Beth Plale. Personal Workspace for Large-Scale Data-Driven Computational Experiment. In *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, GRID '06, pages 112–119, Washington, DC, USA, 2006. IEEE Computer Society.

[159] Osma Suominen, Henri Ylikotila, Sini Pessala, Mikko Lappalainen, Matias Frosterus, Jouni Tuominen, Thomas Baker, Caterina Caracciolo, and Armin Retterath. Publishing SKOS vocabularies with Skosmos. *Manuscript submitted for review*, 2015.

[160] Wang Chiew Tan et al. Provenance in databases: past, current, and future. *IEEE Data Eng. Bull.*, 30(4):3–12, 2007.

[161] Andreas Truszkowski, Kalai Vanii Jayaseelan, Stefan Neumann, Egon L. Willighagen, Achim Zielesny, and Christoph Steinbeck. New developments on the cheminformatics open workflow environment CDK-Taverna. *Journal of Cheminformatics*, 3(1):54, 2011.

[162] Peter Van Garderen. Archivematica: Using micro-services and open-source software to deliver a comprehensive digital curation solution. In *Proceedings of the 7th International Conference on Preservation of Digital Objects, Vienna, Austria*, pages 145–149, 2010.

[163] Chad Vicknair, Michael Macias, Zhendong Zhao, Xiaofei Nan, Yixin Chen, and Dawn Wilkins. A Comparison of a Graph Database and a Relational Database: A Data Provenance Perspective. In *Proceedings of the 48th Annual Southeast Regional Conference*, ACM SE '10, pages 42:1–42:6, New York, NY, USA, 2010. ACM.

[164] Mark Ware. Pathfinder research on web-based repositories. *London: Publisher and Library/Learning Solutions*, page 3, 2004.

[165] Bruno Wassermann, Wolfgang Emmerich, Ben Butchart, Nick Cameron, Liang Chen, and Jignesh Patel. *Sedna: A BPEL-Based Environment for Visual Scientific Workflow Modeling*, pages 428–449. Springer London, London, 2007.

[166] Paul Watson, Hugo Hiden, and Simon Woodman. e-Science Central for CARMEN: science as a service. *Concurrency and computation: Practice and Experience*, 22(17):2369–2380, 2010.

[167] Stuart Weibel, John Kunze, Carl Lagoze, and Misha Wolf. Dublin core metadata for resource discovery. Technical report, 1998.

[168] E James Whitehead and Meredith Wiggins. WebDAV: IEFT standard for collaborative authoring on the web. *IEEE Internet Computing*, 2(5):34–40, 1998.

[169] Jennifer Widom. Trio: A system for integrated management of data, accuracy, and lineage. Technical report, Stanford InfoLab, 2004.

[170] Joss Winn et al. Open data and the academy: An evaluation of CKAN for research data management. 2013.

[171] Petia Wohed, Wil M. P. van der Aalst, Marlon Dumas, and Arthur H. M. ter Hofstede. *Analysis of Web Services Composition Languages: The Case of BPEL4WS*, pages 200–215. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[172] Katherine Wolstencroft, Robert Haines, Donal Fellows, Alan Williams, David Withers, Stuart Owen, Stian Soiland-Reyes, Ian Dunlop, Aleksandra Nenadic, Paul Fisher, Jiten Bhagat, Khalid Belhajjame, Finn Bacall, Alex Hardisty, Abraham Nieva de la Hidalga, Maria P Balcazar Vargas, Shoaib Sufi, and Carole Goble. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic Acids Research*, (Web Server issue):W557–W561, 07 .

[173] Lauren Wood, Arnaud Le Hors, Vidur Apparao, Steve Byrne, Mike Champion, Scott Isaacs, Ian Jacobs, Gavin Nicol, Jonathan Robie, Robert Sutor, et al. Document object model (dom) level 1 specification. *W3C Recommendation*, 1, 1998.

[174] Simon Woodman, Hugo Hiden, Paul Watson, and Paolo Missier. Achieving Reproducibility by Combining Provenance with Service and Workflow Versioning. In *Proceedings of the 6th Workshop on Workflows in Support of Large-scale Science*, WORKS '11, pages 127–136, New York, NY, USA, 2011. ACM.

[175] A. Woodruff and M. Stonebraker. Supporting fine-grained data lineage in a database visualization environment. In *Proceedings 13th International Conference on Data Engineering*, pages 91–102, Apr 1997.

[176] Bohdan S Wynar, Arlene G Taylor, and Jeanne Osborn. *Introduction to cataloging and classification*. Libraries Unlimited Englewood, CO, 1985.

[177] Ruixin Yang, Xinhua Deng, M. Kafatos, Changzhou Wang, and X. S. Wang. An xml-based distributed metadata server (dimes) supporting earth science metadata. In *Proceedings Thirteenth International Conference on Scientific and Statistical Database Management. SSDBM 2001*, pages 251–256, 2001.

[178] Jun Zhao, Chris Wroe, Carole Goble, Robert Stevens, Dennis Quan, and Mark Greenwood. Using semantic web technologies for representing e-science provenance. In *International Semantic Web Conference*, volume 3298, pages 92–106. Springer, 2004.

[179] Jun Zhao, Carole Goble, Robert Stevens, and Daniele Turi. Mining Taverna's semantic web of provenance. *Concurrency and Computation: Practice and Experience*, 20(5):463–472, 2008.

[180] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. von Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, and M. Wilde. Swift: Fast, Reliable, Loosely Coupled Parallel Computation. In *2007 IEEE Congress on Services (Services 2007)*, pages 199–206, July 2007.

[181] Yong Zhao, Michael Wilde, and Ian Foster. Applying the virtual data provenance model. In *International Provenance and Annotation Workshop*, pages 148–161. Springer, 2006.

[182] Élise Meyer, Pierre Grussenmeyer, Jean-Pierre Perrin, Anne Durand, and Pierre Drap. A web information system for the management and the dissemination of cultural heritage data. *Journal of Cultural Heritage*, 8(4):396 – 411, 2007.