FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Improving Verifiability and Repeatability of data-driven Workflows

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Business Informatics

by

Gerhard Hager
Registration Number 1027506

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao. Univ. Prof. Dipl.-Ing. Dr.techn. Andreas Rauber

Vienna, 12th April, 2016

Gerhard Hager	Andreas Rauber

Erklärung zur Verfassung der Arbeit

Gerhard Hager
Address

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 12. April 2016

Gerhard Hager

Abstract

Abstract

Using Workflow Systems for Machine Learning provides a scalable and efficient way to process data to create models which can be used for data-driven decisions and predictions. This thesis examines concepts for improving the verification and repeatability of such workflows. The covered topics include *Data Provenance* and *Dynamic Data Citation*. It investigates how provenance information can be collected during workflow execution and properly represented with a model. Furthermore it is analyzed how the citation of dynamic data can be realized in form of a workflow. To show the feasibility of the elaborated concepts some practical implementations are created in a high-performance, cloud-based machine learning environment. The final implementation is generically designed so the construction approach can be used for other workflow management systems. In addition recommendations for machine learning applications for supporting the discussed topics are proposed. The created concepts and recommendations can be used for improving the verifiability and repeatability of workflows.

Kurzfassung

Das Verwenden von Workflow Systeme für Machine Learning ermöglicht es Daten skalierbar und effizient zu verarbeiten, um daraus Modelle herzustellen, die für datengetriebene Entscheidungen und Vorhersagen verwendet werden können. Diese Masterarbeit untersucht Konzepte für die Verbesserung der Verifikation und Wiederholbarkeit von Workflows. Die behandelten Themen sind *Data Provenance* und *Dynamic Data Citation*. Es wird untersucht wie Provenance Information während der Ausführung eines Workflow gesammelt und dargestellt werden kann. Des Weiteren wird analysiert wie das Zitieren von dynamischen Daten mit Hilfe von Workflows umgesetzt werden kann. Um die Durchführbarkeit der ausgearbeiteten Konzepte zu zeigen, wurden eine praktische Implementierung in einer cloud-basierten Machine Learning Umgebung durchgeführt. Die fertige Implementierung wurde möglichst generisch konstruiert damit man die Vorgehensweise auch auf andere Workflow Management Systeme anwenden kann. Zusätzlich wurden Empfehlungen für Machine Learning Applikationen erstellt, um die behandelten Themen besser unterstützen zu können. Der Einsatz der ausgearbeiteten Konzepte und Empfehlungen dieser Masterarbeit verbessern die Verifizierbarkeit und Wiederholbarkeit von Workflows.

Contents

List of Figures

Listings

CHAPTER 1

Introduction

1.1 Motivation

With the growing amount of information available today, simple statistics and database queries alone are not enough to identify the complicated relations hidden within this data. Hence, more complex analytical methods from the field of Machine Learning are necessary to fully utilize the potential of this enormous amount of information. Because of this the popularity of workflows increased drastically in the last decade.

Using Workflow Management Systems for Machine Learning provides a scalable and efficient way to process data by creating models which can be used for data-driven decisions and predictions. These are essential tools for data-centric science because they offer a systematic and transparent way to routinely analyze complex data. With such systems workflows can be created and changed dynamically by combining standard components in new ways to get more insight in specific scientific questions. The finished workflows can be shared and reused by other data analysts [9].

Machine Learning is not just used by researchers but also found its way into industry. In the markets of today one could argue that the most valuable resource is information. In the last few years many companies collected as much data as possible from their business processes. By analyzing it they want to improve their processes and products and thus giving them an advantage over their competitors.

An example would be Amazon which is one of the most successful companies today. One big part of this success story is the usage of data generated from users to improve their recommendation models. Such recommendation models can of course be modeled with the help of workflow systems [10].

Even though the software applications for creating workflows already offer a lot of features and get easier to use, there are still issues present when using such automated analysis processes. One problematic issue of these workflows is the verification of the results generated. By using standard components which are provided through services or libraries the user loses control and insight of the internal computation. Even a small update can change the internal handling of these components and lead to different results. It is difficult to detect the systematic bias introduced by such changes and particularly to determine if the new results are more accurate or just incorrectly calculated. Locating the cause of such computation differences is tough because a workflow usually consists of several process steps, which all have their own input and output. Checking every step manually takes a lot of time, especially when a large amount of data is involved.

Another important topic is the repeatability of workflow experiments which is necessary to verify and use the knowledge gained from earlier work of other analysts. Even if the newest workflow software products are well documented and offer standard building blocks, most workflows are not built to be reusable over a long period of time. Great amounts of effort have already been raised to combat this problem of *Workflow Decay* [11] [12] [13].

1.2 Research Questions

The goal of this thesis is to create concepts and recommendations to improve the verifiability and repeatability of high-performance, cloud-based machine learning environments. To prove their feasibility the created concepts will be implemented in the *Microsoft Azure Machine Learning Studio* [1]. This platform is chosen because it is a new cloud-based service for creating workflows. In addition it offers good documentation and great usability.

To achieve this goal a couple of related topics from literature were selected. The first technique for improving repeatability is called *Dynamic Data Citation.* It is used to reproduce the results of experiments even if the input data used is constantly evolving. The following states a suitable description: "Dynamic data citation provides tools and methods which allow referencing a specific state of a subset, which was derived from an evolving data source." [14].

A concept related to data citation which is important for ensuring the repeatability of and establishing trust in process execution is the concept of data provenance. Data provenance can be defined as: "The provenance of a data item includes information about the processes and source data items that lead to its creation and current representation" [15, p. 1]. This means it comprises information about the origin of the data and every change applied to it which led to the current state of a data item. It can improve the repeatability by recording information during the execution of a workflow.

Through these selected topics the following Research Questions can be derived:

[1]https://studio.azureml.net

- How can a model for the documentation of data provenance be applied to a cloud-based ML environment?
- Which provenance information can be collected automatically during a workflow process execution in a cloud-based ML environment?
- How can dynamic data citation be integrated in workflow-driven analysis processes?

The remainder of this thesis is organized as follows. Chapter 2 discusses related literature on the topics *Data Provenance* and *Data Citation* as well as some of practical applications.

Chapter 3 investigates the first two research questions. It analyses how provenance data can be collected and modeled. It also documents the implementation of a system for recording provenance information of data-driven workflows.

Chapter 4 focuses on the third research question. This is done by the implementation of a workflow which uses the concept of dynamic data citation.

The final chapter summarizes the obtained results and created recommendations. In addition a number of directions for future work are mentioned.

The appendix contains documents which were used or created during the practical work. In addition several scripts which are part of the final implementation are listed.

CHAPTER 2

Related Work

This chapter takes a look at scientific work on the topics *Data Provenance* and *Data Citation* in general or in combination with workflows management systems. By analyzing theoretical concepts about data provenance, insights are gained for the practical part. It is also interesting how provenance information is recorded and handled in different software products. Since a *Provenance Model* has to be used for the implementation in Microsoft Azure Machine Learning the most prominent models are discussed. The most recent theoretical developments and commonly used practices for *Data Citation* are also discussed. Especially work focusing on managing the citation of dynamic data is interesting for the third research question.

2.1 Workflow Management Systems

The first challenge of this thesis is to choose a workflow management system for the practical implementation. This section provides an overview of several established workflow management system applications. Some of their features are analyzed in subsequent sections.

A prominent example is the *Taverna Workbench* which uses its own workflow language *Taverna* [1] to construct experiments for analyzing data. It is a Java based open source tool developed by the *myGrid Project*[2]. Taverna is successfully used in several scientific domains like bioinformatics, astronomy, medical research and life science[16]. The Figure 2.1 shows an example workflow created with the Taverna Workbench.

Kepler [3] is also an open source tool which is used in several multi-disciplinary fields. A

[1]http://www.taverna.org.uk/
[2]http://www.mygrid.org.uk/
[3]https://kepler-project.org

key feature is a web service extension which enables the use of workflows as a remote resource. It also provides an actor-oriented modeling method and offers several extensions [17]. Another workflow management system frequently used for scientific projects is Pegasus [18]. The special usage of distributed resources allows this system to be executed in distributed environments.

The state-of-the-art of these applications evolves fast and recently launched cloud services like Amazon Machine Learning [4] and Microsoft Azure Machine Learning [5] show that the big players of the software industry also see a high potential in automated analysis processes.

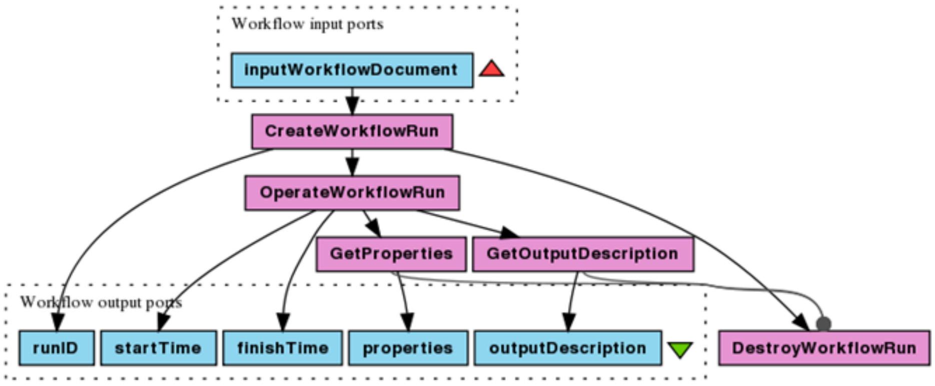


Figure 2.1: Workflow created with Taverna [1]

But these are not the only software products created for cloud environments. *SciCumulus* is a middleware for the distribution, execution and monitoring of workflows on the cloud. It uses a three-layer architecture for transferring the locally created workflows to a distributed environment consisting of virtualized machines where they are parallel executed [5].

All of these systems are legitimate choices for the implementation. However many aspects of established applications like Taverna and Kepler are already examined in a variety of publications. Focusing on a newer technology which is not already covered in depth is more interesting for this thesis. Cloud-based applications are a good choice because their flexible scalability allows the processing of variable sized of data. Even if only smaller amounts of data are used during implementation it is easily possible to increase the computational power. This way even information on the scale of big data can be managed. The Microsoft Azure Machine Learning service offers great usability and a comprehensive documentation. It also includes a feature which makes it easy to share

[4]http://aws.amazon.com/de/machine-learning
[5]https://studio.azureml.net

created workflows with other users by publishing them as web services. This allows users to use their created workflows over the internet through RESTful API calls and integrate them into their software. Because of these reasons the Microsoft Azure Machine Learning service is chosen for the practical part.

2.2 Research Objects

A major problem of workflows is their inability to produce the same results over time because of changing resources and components, which is called *Workflow Decay* [12]. Zhao et al. created an empirical study with Taverna workflows to identify the cause of this decay. The workflows were taken from the myExperiment repository which is a platform for sharing and exchanging workflows [6]. One finding was that about 50% of the workflows showed decay as a result of unavailable third party resources. A recent study for workflow re-execution by Mayer et al. [19] shows that workflow decay is even more problematic. Their results show that out of 1443 workflows only 29.2% were re-executable. As a countermeasure to workflow decay both studies proposes a minimal set of auxiliary information that should be bundled to the workflow. This set of information is defined as a *Research Object*.

Research Objects are aggregations of resources that encapsulate all the information necessary to share and reuse knowledge. The definition of a workflow-centric Research Object is proposed in [2]. The usage of these objects is designed for workflow languages and systems like Taverna, Triana [7] or Kepler. Such objects consist of the following elements:

- Workflow template for defining the workflow
- Provenance information of the results obtained
- Artifacts that are relevant (papers for description, datasets used)
- Annotations that semantically describe the other elements

Figure 2.2 shows an example of a workflow template (b) containing two processes. When this workflow is executed with a workflow engine the result is a workflow run which contains the provenance information (c). This information consists of processes which use artifacts as input through the *used* association and create artifacts with the *wasGeneratedBy* association. The abstract workflow template (a) contains all the abstract processes of the workflow, including their input and output, which are semantically labeled to specify to steps performed. The goal of this thesis is to apply concepts to a workflow system that can be integrated into their automatic execution of processes. The provenance part of the Research Objects can be generated during a workflow execution. Hence, only this

[6] http://www.myexperiment.org
[7] `http://www.trianacode.org`

part will be of interest as it is a key component for understanding the results obtained and thus can improve the repeatability of the workflow.

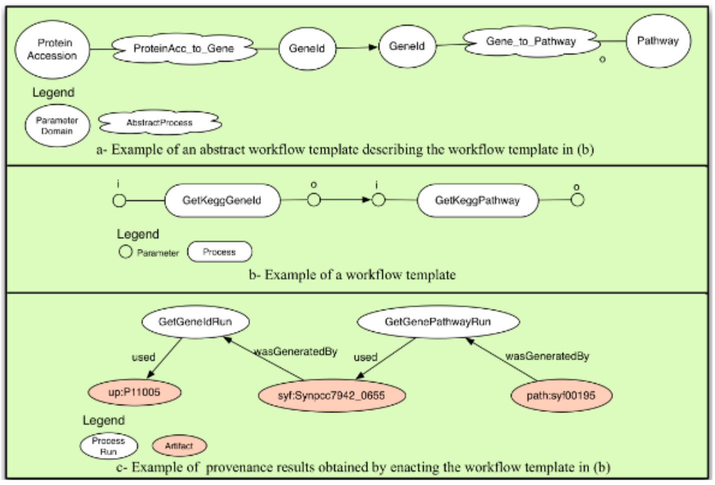


Figure 2.2: Example of abstract workflow, template and provenance information [2]

2.3 Provenance of Data

The *Wf4Ever project*[8] also proposes extensive use of provenance information and research objects to achieve workflow preservation [20]. In this paper a workflow using a database and a web service as data sources is illustrated. Since both of these sources are subject to changes the results can change as well. Hence, it is important to document the provenance of the workflow output. This provenance information can be used to backtrack the original workflow in case some steps are no longer executable. It is also mentioned that provenance can be used to verify if evolved resources are still consistent with a workflow.

The *whole system snapshot exchange* (WSSE) is an approach for improving repeatability through the use of cloud based computing [21]. The general idea of this method is to create a digital image which contains the entire environment of a workflow. This image contains the operating system, software and databases used. Because the size of an image could by several terabytes the method proposes that this system is available as a service through cloud computing. The virtualized system could be shared or copied between data analysts. Cloud providers offer the technology to create large databases that can be used as a shared resource which offers a better consistency than keeping a copy of the database on a local system. The costs for such databases could be shared among all groups which are using them. It is interesting that the new cloud-based machine learning services of Microsoft and Amazon, which were not available when this work was published, already offer the possibility to realize this idea. The WSSE approach

[8] `http://www.wf4ever-project.org`

cannot be used for the practical part because it is a higher-level concept but this thesis contributes to the general idea of such systems.

Davidson and Freire provide a survey which compares how provenance is documented in different workflow system solutions [22]. They mention that for a provenance management solution there are three key components necessary: capturing, modeling and storing. The capturing of provenance information during execution is already supported by some workflow systems which will be examined in a later section. The information also has to be mapped into a suitable provenance model. The last component is a system for storing and querying the provenance information. This thesis will focus mainly on the first two components.

Deelman et al. [23] identify the necessary information for documenting the provenance of a workflow. It consists of the source data, the intermediate data products of each step, as well as the following items for each component of the process chain:

- Time Stamp
- Program Version Number
- Component Version Number
- Execution Host
- Library Version
- Data Source

Some of this information will not be available in a cloud-based environment. It is also stated that it is important to track the provenance of the workflow as it changes over time to get a better understanding how the workflow was created.

2.3.1 Provenance Models

For the first research questions a suitable provenance model has to be chosen for the provenance documentation. Some of these models with a workflow focus were already published a decade ago [24] [25] [18]. They are the result of the first and second provenance challenge commenced in June and December 2006 [26]. These challenges created the foundation for the *Open Provenance Model (OPM)* [9].

This model is technology independent and is used in many workflow system applications. The core model uses the nodes *Artifacts, Processes and Agents* and their dependencies are displayed as directed edges. The five types of edges used are called WasGeneratedBy (WGB), WasDerivedFrom (WDF), WasControlledBy (WCB) and WasTriggeredBy

[9] `http://twiki.ipaw.info/bin/view/Challenge/WebHome`

(WTB). Together they form a directed acyclic graph which represents the provenance information of a data result [27]. Figure 2.3 shows the relations between the nodes and edges of the model. The OPM also includes a XML Schema (OPMX) a Vocabulary (OPMV) and an Ontology (OPMO) for improving the representation [3]. This format is used in several workflow applications like Taverna, Kepler and Activiti [28].

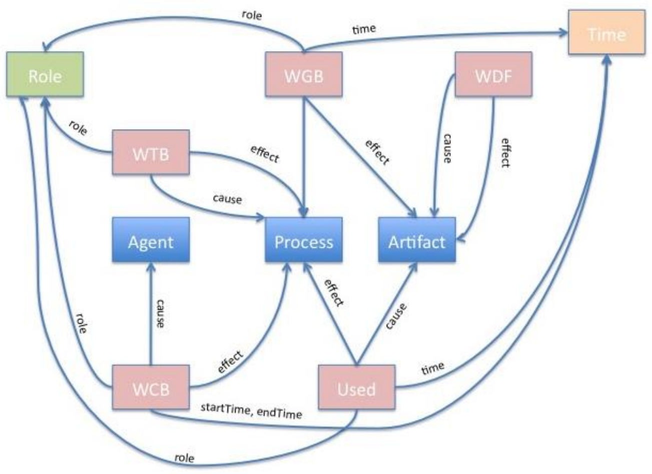


Figure 2.3: OPM Nodes and Edges Relationship [3]

Taverna offers the possibility to export provenance information in the OPM format or the more detailed *Janus* format. Janus is specifically designed to fit to the execution model of Taverna [4]. This domain-aware model extends the previous Taverna provenance model *Provenir* with semantic annotations. In combination with a query infrastructure it offers a query language for accessing the stored provenance data.

The PROV Model

The first official standard for provenance documentation is the *PROV* model. It was published in April 2013 by the World Wide Web Consortium [29]. It incorporates key requirements and design decisions identified by previous efforts [26] in addition to an elaborated documentation. Because of these features this model is chosen for the provenance documentation part of this thesis.

This standard is already supported by several applications [10] which allows the exchange of provenance information over the web. An interesting example is the tool *Git2PROV*. It extracts versioning information from a version control system like Github [11] and maps it to the PROV model [30].

[10] `http://www.w3.org/TR/prov-implementations`
[11] `http://www.github.com`

The PROV standard consists of twelve documents which describe this inter-operable model for interchanging provenance information [8]. The most interesting documents are the recommendations *PROV-DM, PROV-O, PROV-N and PROV-XML*. A detailed description of the data model is defined in PROV-DM. The PROV-O document describes the PROV ontology with the OWL2 language which can be used to map a model to the RDF format. PROV-N and PROV-XML define a humanly readable notation and a xml schema respectively.

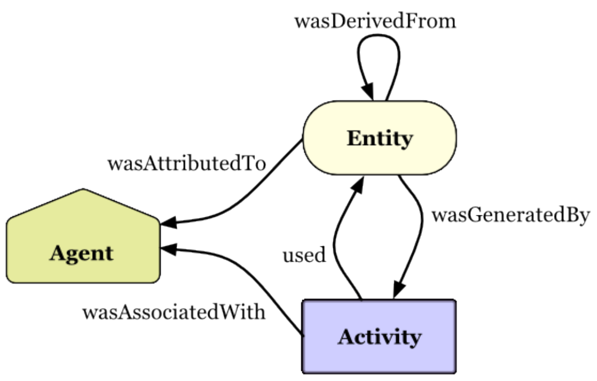


Figure 2.4: Basic PROV concept

Figure 2.4 shows the basic elements of the model and their relationships. An *Entity* represents all physical, digital or conceptual things. This element can be used for the source and result data as well as the intermediate data between the single steps of a workflow. *Activities* can create or change entities and represent the single execution steps used for a workflow. An *Agent* can be a person, software or even an organization and always has a responsibility for one or more Entities and Activities. The workflow system software will thus be an agent for the workflow because it enables its execution.

The PROV model has in many aspects a resemblance with the OPM model. Both use the insights gained during the provenance challenges for their design. The basic elements of their ontology definition are quite similar. Both use an *Agent* node, the PROV elements *Entity* and *Activity* represent the basic idea of the *Artifact* and *Process* nodes from OPM. The definition of the PROV model is however more complex as it also provides subtypes for each element and additional concepts like *Locations*, *Plans* and *Derivations*.

The OPM standard is still supported by many applications like VisTrails, Taverna or Karma [12]. It is however slowly replaced by the PROV model as it is an official standard. The W3C provides a comprehensive list of implementations using the PROV model [13].

[12] http://openprovenance.org/
[13] https://www.w3.org/TR/prov-implementations/

Provenance Capturing in other Workflow Systems

This section looks at various workflow engines and how they support the provenance of data. Not only are the details on which information is captured during the execution of a workflow of interest, but also which provenance models and formats are used. Mayer et al. compare the provenance capturing of some prominent workflow management systems [28]. The authors designed a scientific workflow, implemented it in each workflow system and analyzed which provenance data is recorded during the execution.

The *Taverna Workbench*[14] can capture every individual step executed during a workflow. This also includes the exact execution time and the data exchanged between each step. All the information captured is stored in a database. The Open Provenance Model format [27] can be used to export the recorded information but excludes the input data that is processed by the workflow. This data is included when exported in the *Janus* format which is specifically designed for Taverna workflows [28]. Janus extends its predecessor *Provenir* with subclasses in order to describe a workflow run in greater detail [4].

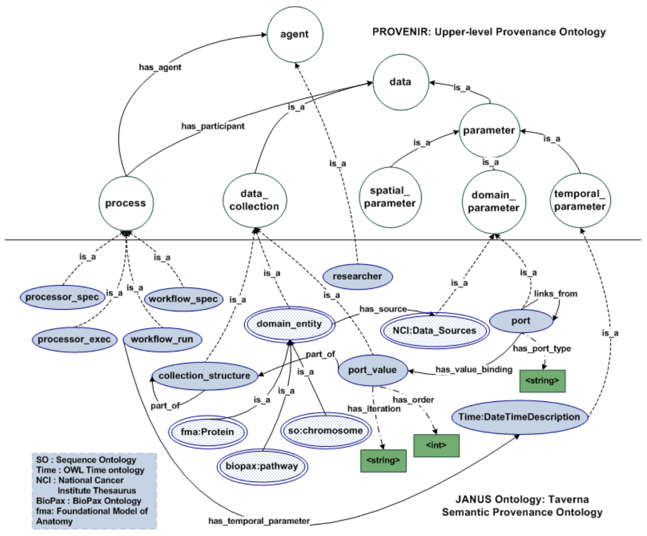


Figure 2.5: Janus Ontology [4]

Capturing the intermediate results of a workflow can lead to an extensive use of storage space. It is reasonable if only small amounts of data are used but not for bigger data sizes. The PROV model also is not intended to save the data itself. Hence, only metadata of the intermediate results will be recorded in the practical part.

The *Kepler Scientific Workflow System*[15] uses a special module called *Provenance Recorder* to record provenance data. The recorded information is stored in a relational database

[14]http://www.taverna.org.uk/

[15]http://www.kepler-project.org/

which can be queried by using the Database Manager of Kepler. Like for Taverna an export option is available to get the data into a XML file by using the OPM format. Unlike for Taverna this output also includes time stamps which help understanding the execution sequence of the workflow [28].

When looking at the table specification of the provenance database a similarity to the PROV model can be identified. The agent, activity and entity elements are called director, actor and entity respectively. For each actor the input and output ports are stored describing the relation between the single elements. In the PROV model relations like *used* and *wasGeneratedBy* serve this purpose. The storing of context information of a workflows execution is also a reasonable feature. In addition the changes of a workflow by users are also stored [31]. Recording the evolution of a workflow is also interesting for cloud-based workflow management systems and would definitely increase their repeatability.

Pegasus tracks provenance with its own lightweight job monitoring program called *Kickstarter*. Since this workflow system is used for disturbed environments it needs to record additional information:

- intial arguments and exitcode of a task
- start time and duration
- hostname of the host on which the task ran and the directory in which it executed
- stdout and stderr
- environment that was set up for the job on the remote node
- machine information about the node that the job ran on (architecture, operating system, number of cores, memory)

The kickstarter process writes this information directly into an xml with a Pegasus specific format. Another process maps and transfers the data into a relational database. The Pegasus dashboard uses this information to display reports about a workflow. An option for exporting this information using the OPM format is currently in development [32].

SciCumulus has the additional challenge of capturing the provenance of a distributed environment. Each of its three architecture layers has special components to be able to merge the whole provenance information of a single workflow run together. Figure 2.6 illustrates this architecture.

The capturing starts in the *Execution Layer* during the execution of activities in each virtualized instance. Once all are finished the Distribution Controller located in the

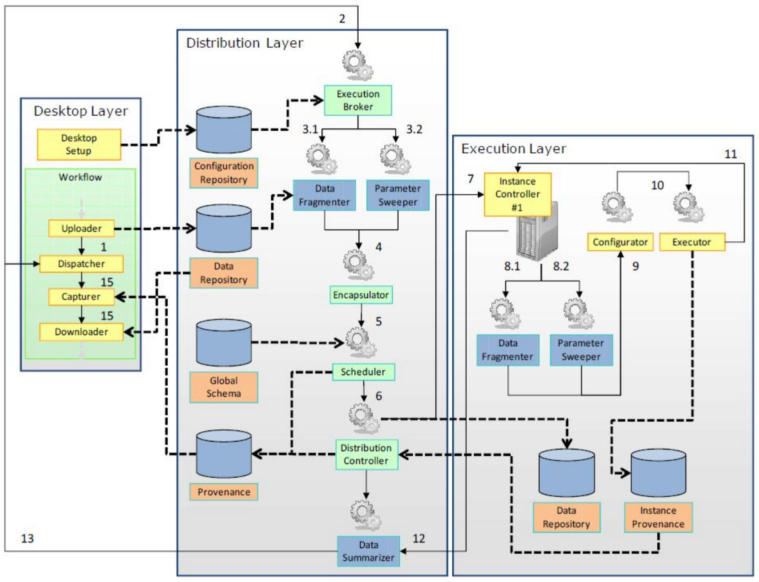


Figure 2.6: SciCumulus Architecture [5]

Distribution Layer collects and merges the data from each instance and invokes the *Data Summarizer* which creates the final results of the workflow. After the execution all the results, including the provenance information, are transferred to the *Desktop Layer* where the *Provenance Catpurerer* stores the provenance data in a local repository using the OPM model. Detailed information about the results of each virtual instance can be queried from the repository. If Azure Machine Learning wants to provide information about the execution environment additional components like in the architecture of SciCumulus will be necessary.

2.4 Data Citation

Tracing provenance during each step of an experiment helps understanding how specific results are created. However the provenance trail is completely different if another set of initial data is used for a workflow. Generating the same results while repeating a workflow, whether shared from other analysts or self created, is only possible by using the same dataset as in the original experiment. Data citation allows to verify the data used during an experiment and thus can improve its repeatability.

Altman et al. [6] explains how data citation evolved during the last decades. Four development phases shown in 2.7 were identified. Each phase facilitates different core principles.

The first three phases focus on established practices which are commonly used, like citing author, title of a work and research data used. During the third phase global persistent

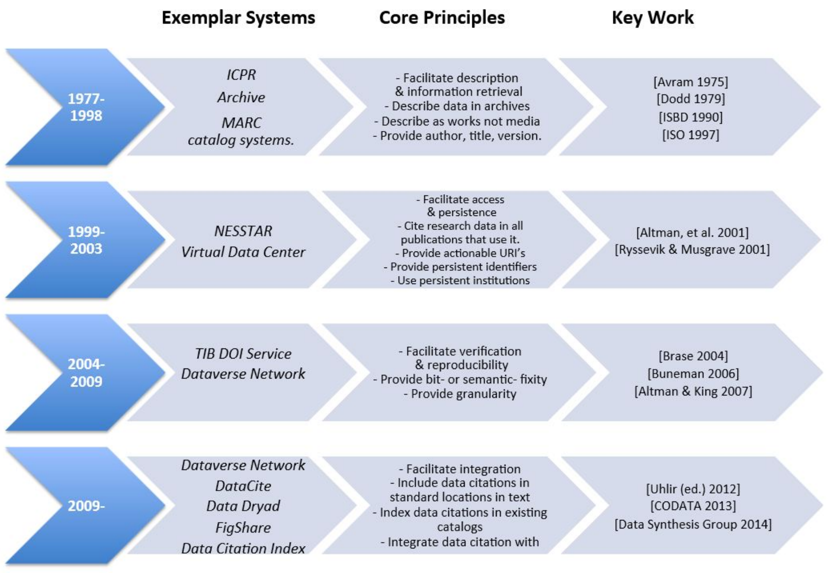

Exemplar Systems
Core Principles
Key Work
1977-1998
ICPR Archive
MARC catalog systems.
- Facilitate description & information retrieval
- Describe data in archives
- Describe as works not media
- Provide author, title, version.
[Avram 1975]
[Dodd 1979]
[ISBD 1990]
[ISO 1997]
1999-2003
NESSTAR
Virtual Data Center
- Facilitate access & persistence
- Cite research data in all publications that use it.
- Provide actionable URI's
- Provide persistent identifiers
- Use persistent institutions
[Altman, et al. 2001]
[Ryssevik & Musgrave 2001]
2004-2009
TIB DOI Service
Dataverse Network
- Facilitate verification & reproducibility
- Provide bit- or semantic- fixity
- Provide granularity
[Brase 2004]
[Buneman 2006]
[Altman & King 2007]
2009-
Dataverse Network
DataCite
Data Dryad
FigShare
Data Citation Index
- Facilitate integration
- Include data citations in standard locations in text
- Index data citations in existing catalogs
- Integrate data citation with
[Uhlir (ed.) 2012]
[CODATA 2013]
[Data Synthesis Group 2014]

Figure 2.7: Data Citation Phases [6]

identifiers like *Digital Object Identifier (DOI)* [16] or Handle emerged. These services provide a permanent link which resolves to a website and thus grant access to the desired data. If the website location changes the DOI resolves to the new source providing a long-term accessibility. For the current phase several challenges were identified. Examples are the usage of provenance for the citation text or handling big, complex and dynamic data [6]. Callaghan [33] also identifies several open questions regarding data citation. The citation of datasets which are continuously updated is still an issue. Another important topic is the attribution of all contributors to a specific set of data.

The accomplishments from each described phase shaped how data is cited today. For a data reference the following minimum information is needed: author, title, date, publisher, identifier and access information. The easiest form of depositing a dataset is through providing an URL. However websites can go offline or change their location, hence it is not a stable reference. Therefore the already mentioned persistent identifiers (PIDs) were introduced. A PID has to be requested from special Registration Authorities (RA). They manages the link between data and the identifier [34].

One of these RAs is the *DataCite* agency which was founded through a collaboration of several scientific institutes. This agency offers the services of assigning DOIs to datasets. The metamodel of DataCite as well as their recommendation for citing data is explained in [35].

[16] https://www.doi.org/

Even if services like DataCite work well there is still a platform needed for storing datasets. This task can be carried by data repositories. The open-source project *Dataverse* [17] is one of the repositories that already generate data citations automatically when a dataset is uploaded. They store the combination of data files and metadata as a so-called *dataverse*. For each of these instances the visibility, ownership and access rights can be managed separately. The platform supports Handle and DOIs as a method for citation but is not designed for dynamic data[36].

There are various groups which try to address the remaining challenges with recommendations and examples of best practice. These include the *Research Data Alliance's Working Group on Data Citation (WG-DC)* [18], the *CODATA-ICSTI Task Group* [19] and *DataCite* [20], to name a few.

In a combined effort they created the *Joint Declaration of Data Citation Principles* [37] as the *Data Citation Synthesis Group* [21]. It consists of eight principles that serve as recommendations and best practices for citing all kind of data. This declaration is widely embraced by the scholarly community and represents a great accomplishment. Especially the seventh principle has an affinity with the goals of this thesis.

1. **Importance**, Data should be considered legitimate, citable products of research. Data citations should be accorded the same importance in the scholarly record as citations of other research objects, such as publications

2. **Credit and Attribution**, Data citations should facilitate giving scholarly credit and normative and legal attribution to all contributors to the data, recognizing that a single style or mechanism of attribution may not be applicable to all data

3. **Evidence**, In scholarly literature, whenever and wherever a claim relies upon data, the corresponding data should be cited

4. **Unique Identification**, A data citation should include a persistent method for identification that is machine actionable, globally unique, and widely used by a community

5. **Access**, Data citations should facilitate access to the data themselves and to such associated metadata, documentation, code, and other materials, as are necessary for both humans and machines to make informed use of the referenced data

6. **Persistence**, Unique identifiers, and metadata describing the data, and its disposition, should persist – even beyond the lifespan of the data they describe

[17]http://dataverse.org/
[18]https://rd-alliance.org/groups/data-citation-wg.html
[19]http://www.codata.org/task-groups/data-citation-standards-and-practices
[20]https://www.datacite.org
[21]https://www.force11.org/datacitation/workinggroup

7. **Specificity and Verifiability**, Data citations should facilitate identification of, access to, and verification of the specific data that support a claim. Citations or citation metadata should include information about provenance and fixity sufficient to facilitate verfiying that the specific timeslice, version and/or granular portion of data retrieved subsequently is the same as was originally cited

8. **Interoperability and Flexibility**, Data citation methods should be sufficiently flexible to accommodate the variant practices among communities, but should not differ so much that they compromise interoperability of data citation practices across communities

2.5 Dynamic Data Citation

As already mentioned in the last section handling dynamic data is one of the imminent challenges of data citation [6] [33]. Dynamic data has the property of changing over time, whether done by adding new measures or modifying and deleting old ones. If data is cited by only referring to a current version of the dataset, it is not possible to verify or reproduce the results obtained if this data changes. To guarantee that the cited data is always in the same state each time is hard to achieve. This is especially true when using a database as source.

Most experiments do not need all the information provided by a dataset. Sometimes only a certain timeframe or specific columns are analyzed. Re-using the exact same subset is an additional challenge when it comes to dynamic data. For the final solution this important aspect also has to be considered.

An example for changing data is the updating of weather data time series with the newest measurements or the tracing of twitter tweets over time. The dynamic part of such datasets is that they are constantly enlarged with new entries. However old entries are not a subject to change which makes the citation slightly easier. An example for the modification of old entries is the *Argro Program* [22]. Its goal is the measurement of ocean temperatures with sensors planted on buoys. Such a buoy has a cycle of different float depth and can only transmit their measurements to satellites when they float on the surface. This can of course lead to transmission errors which are corrected each time there is a strong connectivity. Because their sensors have to deal with temperature changes, induced through different float depth, their measurements can contain a thermal lag error. Older measurements are constantly adjusted by the buoy and changes are transmitted every time the buoy reaches the surface. The correction of old entries and addition of new ones results in a constantly changing dataset [38].

The RDA Data Citation Working Group [23] focuses on providing solutions and reference

[22] `http://www.argo.ucsd.edu/`

[23] `https://rd-alliance.org/groups/data-citation-wg.html`

implementations for citing dynamic data. They created a document containing 14 recommendations for making evolving data citable [39]. To ensure that an implementation can be created with any technology they are generically designed. Since the goal of the third research question is to create a systems which supports dynamic data citation these recommendations can be used to realize a solution in Microsoft Azure.

The core principles revolve around the use of versioned and time stamped data and the recording of each executed query [14]. They are structured into four sections necessary for creating a system which allows the citation of dynamic data. The first section is about adapting the data store to support versioning and time stamps as well as creating a Query Store. The Query Store is necessary to save query metadata like the query itself, a unique persistent identifier (PID) and the timestamp of the execution. Saving the whole information of the result subset is no longer necessary since the query can be repeated with this information. The second section deals with how specific datasets can be persistently identified. The third section explains how a request for a specific PID has to be handled. The last section describes the steps necessary to deal with modifications to the infrastructure. Besides enabling dynamic data citation this also enables the verification of an experiment since it allows the reproduction of the exact subset used.

The foundation for these recommendations is the model introduced in by Pröll and Rauber [34]. This model is designed for citing data stored in a relational database. A relational database management system offers many requirements necessary for this model out of the box. Unique identifiable data records, a query language for referencing subsets of these record and some other requirements are necessary. It is also mentioned that this model is not limited to databases and lists all the requirements for other data models.

Their efforts to create a model for solving the issue of citing evolving data are continued in [40]. The paper proposes several implementation strategies and describes the steps necessary to extend an existing relational database system to support dynamic data citation. It also contains a use case that follows its application to a sensor network data store. This work is very detailed and is definitely usable for the implementation of a workflow in the practical part.

Pröll et al. [41] examine the privacy issues that can arise when citing sensitive data by means of a use case in eHealth. They point out how such data can affect data citation, provenance generation and experiment execution by using workflows. Methods for protecting privacy and still providing repeatability of experiments are described. As a solution they extended the dynamic data citation framework [40] with measures to ensure anonymity.

2.6 Summary

This chapter provides an overview of related work of topics which are relevant for this thesis. Out of the discussed workflow management systems the Microsoft Azure Machine Learning Studio is chosen for the practical implementation.

After looking at the origination of the provenance concept some provenance models are examined. As a result the PROV model is chosen to document the workflow executions of the cloud-based Machine Learning Studio. We also describe some cases on how provenance is recorded and used by other workflow management systems. The insights gained are important for the implementation of the provenance recording.

The subsequent section investigates the state of the art of data citation. One of the current challenges of this field is how to handle dynamically changing data. The current approaches to overcoming this challenge are discussed. A very promising solution are the recommendations published by the RDA Working Group on Data Citation. Therefore they are used as a guideline for implementing a workflow system which supports the dynamic data citation concept.

CHAPTER 3

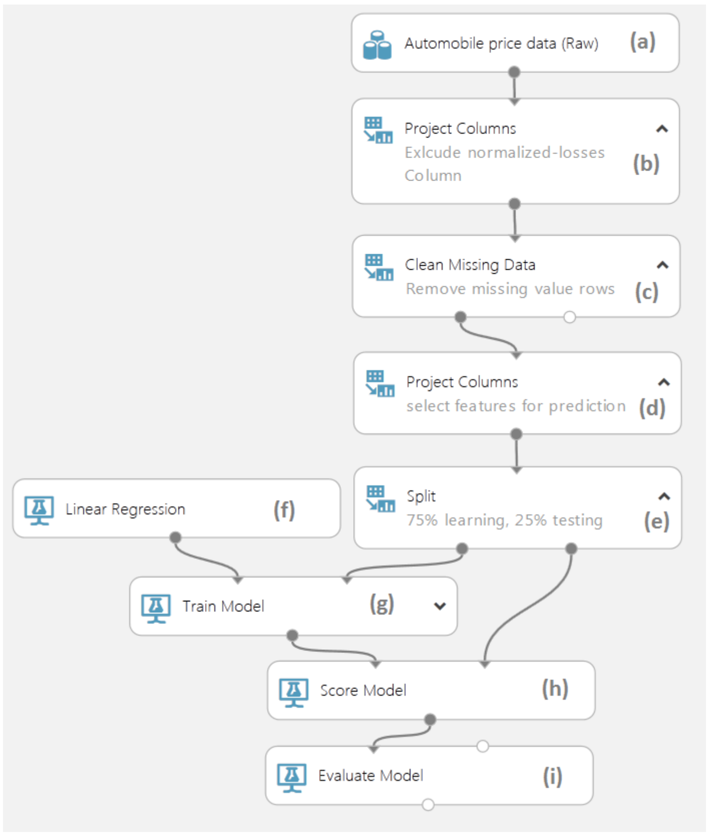
Provenance Documentation of Azure Workflows

This chapter investigates how data provenance of Microsoft Azure Machine Learning workflows can be collected, represented and documented. In the first section the possibilities of the Machine Learning Studio as well as the created method for recording provenance information are explained. The *PROV modeling* section defines how the recorded information is mapped to the PROV model. During the implementation part generic python scripts are created and used to extend a Machine Learning Studio experiment. The generic scripts are used afterwards on another experiment to test the efficiency of the method. In the final part the results are discussed.

3.1 Microsoft Azure Machine Learning

Azure Machine Learning offers the possibility to build, test and publish a predictive analysis model. To accomplish this, the first step is to create a machine learning experiment which is a blank canvas in the beginning. Workflow elements like datasets and analysis modules can be drag-and-dropped onto this interactive canvas. Each of those elements has specific settings which can be modified. By connecting certain input and output ports of the modules the dataflow is defined and the experiment becomes runnable.

An example of such a training experiment is illustrated in Figure 3.1. It shows the *Car Price Prediction* experiment which will be extended to capture provenance information. Every component used in this experiment is explained to show some of the features offered by Azure Machine Learning. The first element (a) is a sample dataset, which is provided by the Machine Learning Studio. It contains data of different cars and their respective prices. For creating a machine learning model the size of the dataset is rather



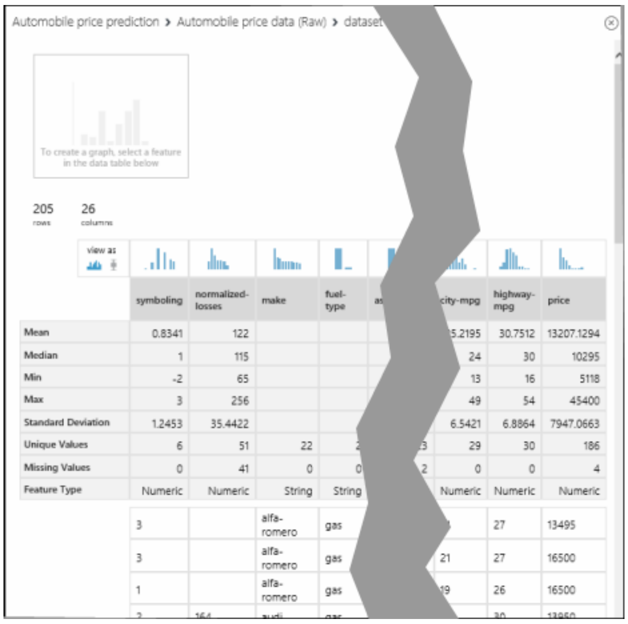
Automobile price data (Raw) (a)
Project Columns
Exlcude normalized-losses Column (b)
Clean Missing Data
Remove missing value rows (c)
Project Columns
select features for prediction (d)
Split
75% learning, 25% testing (e)
Linear Regression (f)
Train Model (g)
Score Model (h)
Evaluate Model (i)

Figure 3.1: Car Price Prediction Experiment

small but still sufficient for the purposes of this work. The Studio offers the following options for defining a data input source.

- Dataset uploaded and stored on Azure
- Url file reference
- Hive Query
- SQL Database
- Azure Table
- Azure Blob Storage
- Data feed provider

The next few steps are necessary to preprocess the data before it can be analyzed. The Project Columns module allows to select a subset of the columns. Since the 'normalized-losses' column contains a lot of empty entries it is excluded in (b). After this the whole


Automobile price prediction › Automobile price data (Raw) › dataset

205 rows   26 columns

	symboling	normalized-losses	make	fuel-type	as	city-mpg	highway-mpg	price
Mean	0.8341	122				5.2195	30.7512	13207.1294
Median	1	115				24	30	10295
Min	-2	65				13	16	5118
Max	3	256				49	54	45400
Standard Deviation	1.2453	35.4422				6.5421	6.8864	7947.0663
Unique Values	6	51	22	2	3	29	30	186
Missing Values	0	41	0	0	2	0	0	4
Feature Type	Numeric	Numeric	String	String		Numeric	Numeric	Numeric
	3		alfa-romero	gas			27	13495
	3		alfa-romero	gas		21	27	16500
	1		alfa-romero	gas		19	26	16500
	2	164	audi	gas			30	13950

Figure 3.2: Car Price Dataset

dataset left is cleaned (c) by removing all rows which contain missing data. The last step of the preprocessing is the selection of every column which should influence the price. Some features like height of the car will not add much new information to the model and can be removed. The columns chosen for predicting the price are *make, body-style, wheel-base, engine-size, horsepower, peak-rpm, highway-mpg* and are selected with the Project Columns module (d). Since a car price can be any numerical value, a linear regression (f) is used for creating the model. As any other module the Linear Regression Module offers properties which can be changed, but for this example the default properties are used. There are also other supervised machine learning techniques available like classifications, SVMs or clustering. The Split module (e) divides the dataset into a training set and a test set. The first output port of this module is used for training the model (g). The result of the Train Model component is already the complete *Car Price Prediction* model. To verify the quality of this model it is tested with the test set from (e) in the Score Model module (h) which tries to predict the prices of each test entry by using the created model. The last module (i) shows how well the model performed and offers feedback if changes to the model, like different feature selection, are an improvement. The second input port of the Evaluate Model component can be used to compare two models.

After the creation of a training model is finished, it can be converted into a *Scoring Experiment*. Through this conversion the model can be published as a web service and used with other data. Users can send their data with appropriate format to the scoring web service and get back the prediction results. By clicking the *Create Scoring Experiment*

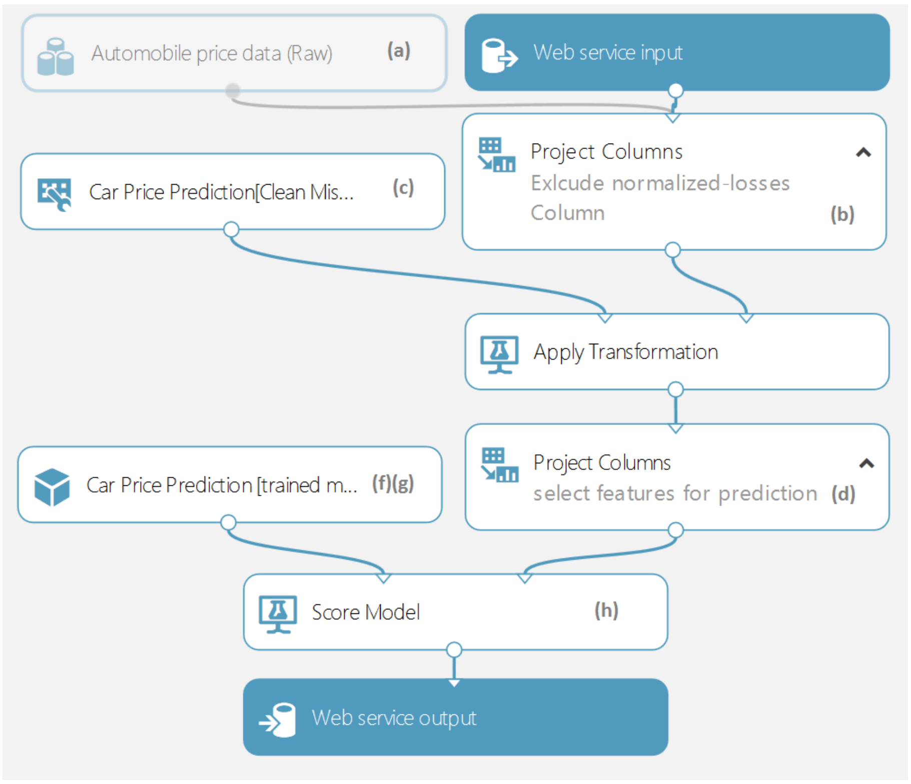

Automobile price data (Raw) (a)
Web service input
Project Columns
Exlcude normalized-losses Column (b)
Car Price Prediction[Clean Mis... (c)
Apply Transformation
Project Columns
select features for prediction (d)
Car Price Prediction [trained m... (f)(g)
Score Model (h)
Web service output

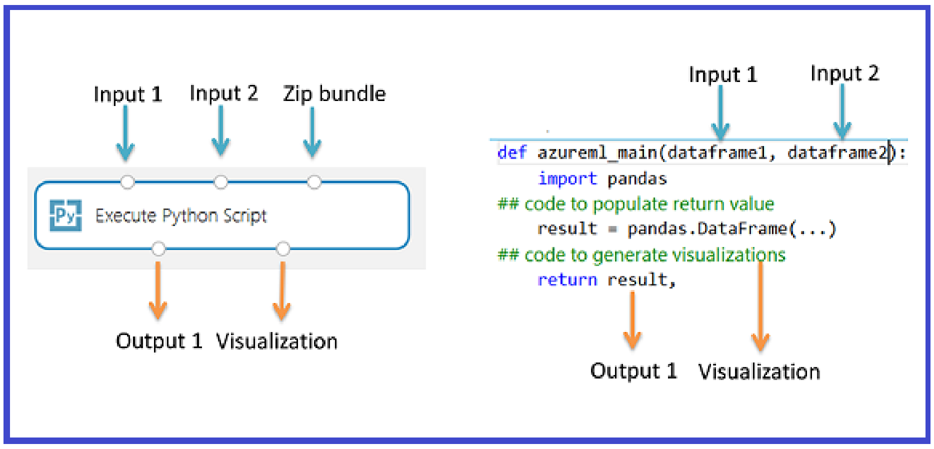
Figure 3.3: Car Price Prediction Scoring Experiment

button the following three steps get performed automatically. The created model is saved as its own *Train Model* component which can be reused for other experiments. Modules which are not necessary for a scoring run are removed. For the Car Price Prediction model this includes the Split, Train Model and Evaluate Model components because they were only needed during the creation of the model. In the last step web service input and output modules get added. Figure 3.3 shows the Car Price Prediction model after the conversion.

3.1.1 Recording Provenance Data

Most of the components offered by Azure Machine Learning are closed modules. The only customization is possible by changing some properties. But the functionality of two modules can be completely defined by the user, the *Execute Python Script* and *Execute R Script* components. The basic idea of these modules is to offer a possibility to incorporate existing code of older workflows which only consist of scripts, to the Machine Learning Studio. For this thesis this intention is changed by using the Execute Python module for other activities.

The basic design of the Execute Python module is shown in the left side of Figure 3.4.



```
def azureml_main(dataframe1, dataframe2):
    import pandas
## code to populate return value
    result = pandas.DataFrame(...)
## code to generate visualizations
    return result,
```

Figure 3.4: Execute Python Module [7]

Its properties only consist of a single text field which can be used to insert python script language. The text field offers python code highlighting but no detection of compiler errors. It must contain an *azure_ml* function so it can be executed by the Anaconda [1] based backend.

The first two input ports of this module are optional and require the *Pandas DataFrame* format. Pandas is an open source python library which provides data structures and data analysis tools [2]. A DataFrame is a two dimensionally labeled data structure which can contain columns with different types. This is a limitation to the practical part because the recorded provenance information must be inserted into a DataFrame structure. The python module only allows to return one data frame as output which is the return value of the azure_ml function. The second output provides the console output of the python interpreter. It also can be used to display PNG images.

Another limitation for the implementation is that each module is a closed unit. This means that the settings and parameter cannot be extracted or changed by the python script since there is no port available for this task. These settings can only be changed manually in the canvas of the Machine Learning Studio. As a result a workaround is necessary to get provenance information from the modules. For this purpose the *Zip bundle* port can be used. This port offers the possibility to use a compressed file which can contain a custom python library. The library created for the implementation contains functions for each module. The purpose of these functions is to provide the python module with setting information of a specific module. Of course this requires some manual effort because the information for every module has to be inserted into the library.

For every module used in the workflow at least one python script has to be present to

[1] https://store.continuum.io/cshop/anaconda/
[2] http://pandas.pydata.org/

create provenance information. Since the Python Script module has only one output port the whole information has to be forwarded from one module to the next. The second input port is therefore receiving the created information of the previous module. After the python script module has appended an additional PROV element the whole data will be forwarded via the output port as a pandas DataFrame to the next module. The first port will mainly be used to get information like the size of a dataset or the time of a specific execution. All the python scripts created are designed to be as generic as possible to minimize the customization needed. The python script modules encase the whole workflow and are a framework for recording the provenance data of an experiment.

3.2 PROV Modeling

The first task of the modeling is to choose a format for the provenance data. The available formats are used in an example documented in 3.2.2 to get a better understanding of the elements of PROV. The following section 3.2.3 defines how each PROV element is mapped to an Machine Learning Studio module.

3.2.1 PROV Annotations

As already mentioned there are three options to notate the PROV data model shown in Listing 3.1, 3.2 and 3.3. Since the most readable option is the PROV-N notation it is chosen for the representation of the provenance information generated in the practical part.

Listing 3.1: RDF using TURTLE notation

```
exg:dataset1     a prov:Entity .
exc:regionList   a prov:Entity .
exc:composition1 a prov:Entity .
exc:chart1       a prov:Entity .
```

Listing 3.2: PROV-N expression

```
entity(exg:dataset1)
entity(exc:regionList)
entity(exc:composition1)
entity(exc:chart1)
```

Listing 3.3: PROV-XML

```
<prov:document>
  <prov:entity prov:id="exg:dataset1"/>
  <prov:entity prov:id="exc:regionList1"/>
  <prov:entity prov:id="exc:composition1"/>
  <prov:entity prov:id="exc:chart1"/>
</prov:document>
```

3.2.2 PROV Example

Listing 3.2 defines several generic elements which originate from the PROV Model Primer document [3] . They serve as the initial objects of this example. These include an entities data set (exg:dataset1), a list of regions (exc:regionList), data aggregated by region (exc:composition1) and a chart (exc:chart1). The generic namespace prefixes are used to label where the element is created. The next step is the definition of some activities and some *used* and *wasGeneratedBy* relations shown in Listing 3.4.

Listing 3.4: PROV Element Definition

```
activity(exc:compose1)
activity(exc:illustrate1)
used(exc:compose1, exg:dataset1, -)
used(exc:compose1, exc:regionList1, -)
wasGeneratedBy(exc:composition1, exc:compose1, -)
used(exc:illustrate1, exc:composition1, -)
wasGeneratedBy(exc:chart1, exc:illustrate1, -)
```

The activity *compose1* uses the *dataset1 and regionList1* to generate a *composition1*. This composition is used by the activity *illustrate1* to generate a *chart1*. This simple PROV data is visualized in Figure 3.5. The usage and generation are relations between entities and activities and point from the future to the past.

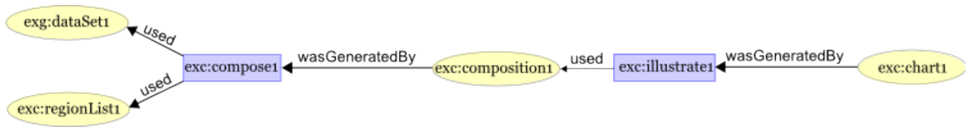


Figure 3.5: PROV Example [8]

Every element can of course contain more attributes. One of them is the time attribute which is crucial for documenting the workflow steps. The following lines are an example for the use of this attribute.

[3]https://www.w3.org/TR/prov-primer/

```
wasGeneratedBy(exc:chart1, exc:compile1,  2012-03-02T10:30:00)
wasGeneratedBy(exc:chart2, exc:compile2, 2012-04-01T15:21:00)
activity(exg:correct1, 2012-03-31T09:21:00, 2012-04-01T15:21:00)
```

3.2.3 PROV Element Mapping

The example of the last section showed the basic use of the PROV elements. The next task is to define how each Machine Learning Studio module is mapped to a PROV element. To get a starting point the Car Price Prediction Experiment explained in 3.1 is extended with python scripts from the top to the bottom. For the identifier of the used elements the namespace *ml* is used which represents the Machine Learning studio.

Entity element

The first module is a datasource which contains a matrix of information for different cars and their prices. This dataset can be represented with the *Entity* element. According to the official PROV-DM definition [42] an entity needs an identifier (id) and can have optional attributes. One of them is the *type* attribute provided by the PROV namespace. Since the dataset is a matrix the row and column count can also be used as an attribute. The resulting entity element can be used for any dataset which is used or generated during the execution of the workflow.

```
entity(ml:datasetid, prov:type="dataset", colsize=##, rowsize=##)
```

Activity element

The *Project Column* and *Clean Missing Data* modules modify the input dataset and thus can be represented as an *Activity* element. The following lines show the PROV-DM definition of this element.

An activity, written activity(id, st, et, [attr1=val1, ...]) in PROV-N, has:

- *id: an identifier for an activity;*
- *startTime: an OPTIONAL time (st) for the start of the activity;*
- *endTime: an OPTIONAL time (et) for the end of the activity;*
- *attributes: an OPTIONAL set of attribute-value pairs ((attr1, val1), ...) representing additional information about this activity.*

Once again it contains an identifier and optional attributes. A machine learning module contains several settings which can be modified. The most important non-default settings can be saved as optional parameters. As already explained in section 3.1.1 the parameter information of a module cannot be extracted directly. As a workaround a manually created zip bundle is necessary which contains the information. The python script module can access this information through the zip bundle port. A big difference to the entity element are the *startTime* and *endTime* attributes. Because the execution time of a module also cannot be read directly a workaround is needed for these attribute as well. The simple way would be to just record the system time for both time attributes when the activity element is created in the python script module. But since the python script module is executed after the activity it cannot represent the true execution time.

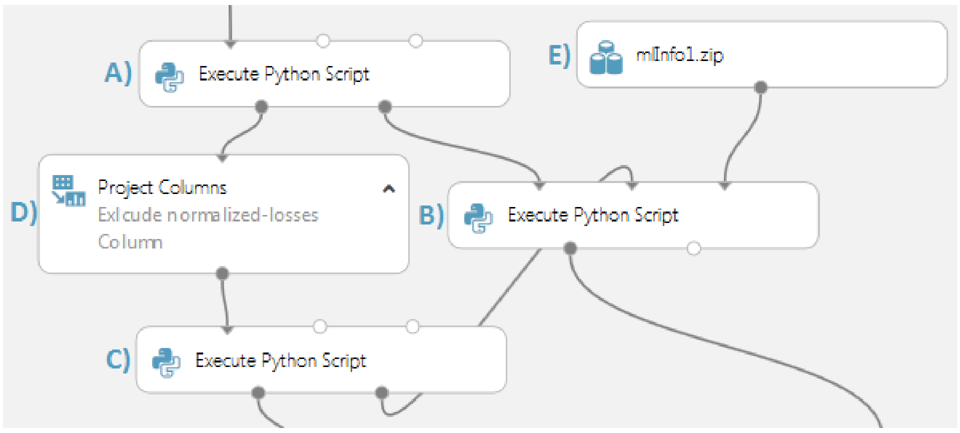


Figure 3.6: Recording execution time for activity

A more accurate option is illustrated in Figure 3.6. It shows two python script modules A and C which are placed before and after a module of the original workflow. Their only task is to record the system time during their execution. The dataset received is forwarded unchanged in the *Output 1* port on the bottom right of the module. This way the original experiment is not modified by the time extraction. Since the python script only has one return value which is send to the *Output 1* port, the time value is forwarded as a console message with the *Visualization* port on the bottom right of the python script module.

The *Visualization* port can only forward simple string values. Hence, another python script B is used to create the activity element. It receives the time value of A and C as well as the information about the project column module provided by the library in the zip bundle (E). The final activity element of such a module looks like this:

```
activity(ml:project1, prov:startedAtTime='2015-07-17 09:27:08',
         prov:endedAtTime='2015-07-17 09:27:50',
         exclude='normalized-losses')
```

The *startedAtTime* and *endedAtTime* attributes are once again taken from the PROV namespace. The *exclude* attribute is a custom attribute which states that the project column modules excluded the 'normalized-losses' column of the dataset. This information is the config parameter of the *Project Columns* module (D) and is read from the workflow config zip bundle (E).

Usage & Generation element

The *Usage* element represents the link between the activity and its entity used. The identifiers of both are saved as attributes in the element as well as the usage time (startedAtTime from activity).

The *Generation* element is very similar. It shows the connection between the activity and the entity generated as a result of the activity. The generation time is the endedAtTime from the activity. The following lines show an example of such elements.

```
used(u1, ml:project1, ml:dataset1, 2015-07-17 09:27:08)
wasGeneratedBy(g1, ml:dataset2, ml:project2, 2015-07-17 09:27:30)
```

Agent element

There are two agents involved during the execution of the workflow. The person starting the workflow and the Machine Learning Studio environment the workflow runs in. Since the studio is the software which runs on behalf of the user, a *actedOnBehalfOf* relation is needed between them. Both have to be created in the beginning of the workflow since every activity has a *wasAssociatedWith* relation with the software agent. Since there is no access to the user name this information is once again gained through the custom python library as an *init()* function. Listing 3.5 shows the agent elements and their relations. All of these elements only have to be created once, except for the wasAssociatedWith element which is needed for every activity.

Listing 3.5: PROV Definition

```
agent(user, prov:type='prov:Person', foaf:givenName='User name')
agent(mlstudio, prov:type='prov:Software Agent')
actedOnBehalfOf(act, delegate=mlstudio, responsible=user)
wasAssociatedWith(aw1, activity=projection1, agent=mlstudio)
```

Elements not used

The PROV model offers many other elements and parameters which can be useful for the right situation, but not all of them are suited to describe a data-driven workflow.

The *Role* parameter can be applied for the usage element to specify the function an entity fulfilled during an activity or how an agent was involved in an activity. These roles are application specific and thus not defined in PROV. The usage of a dataset is always the same for each module of a workflow. Including a Role parameter would only add redundant information. The software agent specified in the last section always just executes the modules. Hence, the role parameter is not used for the provenance documentation.

An agent can execute an activity according to a set of instructions. In PROV this is described as a *Plan*. The instructions which describe a plan are an entity on its own. As already mentioned above, the software agent always performs the same task and thus this element is also not used.

The *Revision* element describes the change of a single entity through time. This makes sense for documents which go through multiple revisions. In a workflow, however, the datasets are always newly generated. The only dataset which could change through user manipulation over time is the first dataset in the workflow which usually contains a large amount of data. However the initial dataset will not change during a workflow execution which makes this PROV element obsolete. The *Derivation* element is very similar because it describes if an entity is derived from another entity. An example would be if a chart is derived from a dataset. This element can be used for document entities and is not needed for workflows.

There are several other elements described in PROV-DM (Quotation, Communication) but they are only useful for specific use cases and are thus excluded from the provenance framework implementation.

3.3 Implementation

After looking at the possibilities offered by the Machine Learning Studio for recording provenance information and how it can be modeled, the implementation can be conducted. For the first implementation the Car Price prediction experiment shown in Figure 3.1 is used. The desired result is a trail of provenance data automatically generated during the execution of this workflow. This information describes the creation of the Car Price Prediction Model and also its evaluation results. In order to create repeatable results Python scripts, written as generic as possible, are created during this implementation.

For the second implementation the Scoring Experiment of the Car Price Prediction model 3.3 is chosen. The generic python scripts created during the first experiments are used. To evaluate the efficiency of the implementation with these scripts the time necessary for adapting the implementation is measured.

3.3.1 Car Price Prediction Experiment

Figure 3.7 shows the final workflow with the added python scripts. The square in the top

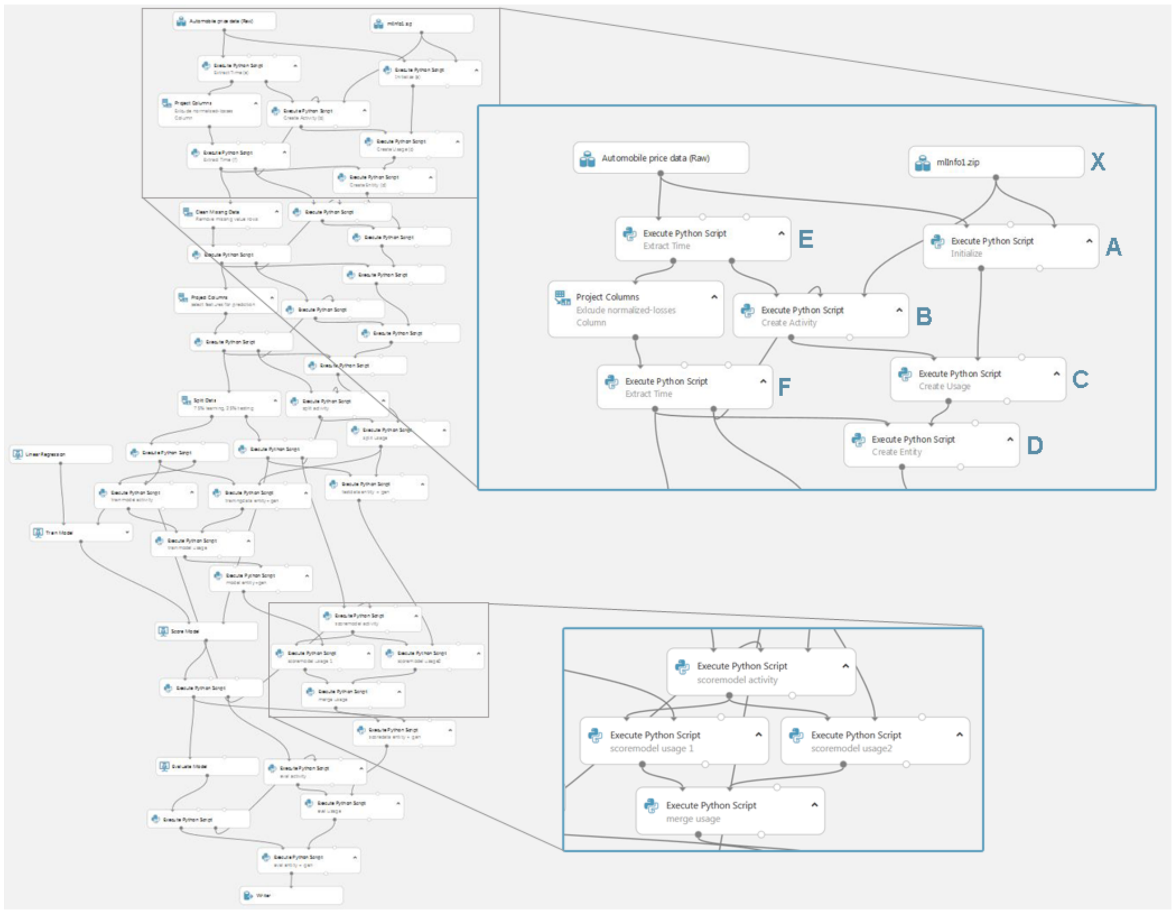


Figure 3.7: Recording Provenance Information

zooms into the the first two modules of the original workflow. The first information is generated in the (a) python module on the right side. At first it creates the person and software agents with information provided by the config zip bundle (x). After this the first entity is created with the dataset information received through the first input port.

As already explained in section 3.2.3 there are python modules used before (e) and after (f) each workflow module to record the execution time of the module. It should be noted that there can be a delay between the time recording and the execution of the module. Hence, the true time is between the recorded points of time. The (b) python module creates the *activity* element, which represents the Project Column module, and the *wasAssociatedWith* relation.

The (c) python module creates the *usage* element and forwards the information to the python module (d). This in turn creates the *entity* for the dataset, created through the Project column module, and the respective *generation* element. The three python modules to the right side already show how the provenance information is parallely passed through to the workflow. The output port of the (f) python module leads to the next workflow module and also is used as the start time for the next activity.

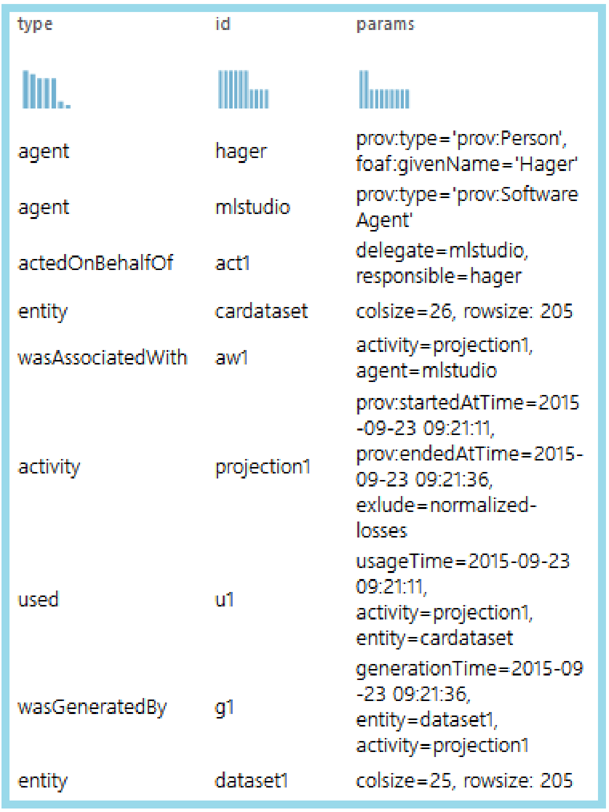
The described approach is used to modify the whole Car Price prediction experiment. Almost all the modules used in the experiment can be interpreted as an activity, expect for the *Linear Regression* module. This module can be documented as an entity element which is used by the *Train Model* activity. The specific parameters used for training the model are appended as parameters to the PROV elements, again read from the config zip bundle (x) as work-around.

The *Split* module divides the flow of the experiment by creating the training and testing dataset. Because of this the activity, representing the *Score Model* module, uses two entities. The first is the model entity created through the *Train Model* module and the second is the testset entity. Since the second port of the python script is always used for receiving the previous results, an additional python script is necessary to combine both paths after the Score Model activity. As shown in the bottom square of Figure 3.7 this script takes the PROV elements from each path and combines them into a single data frame.

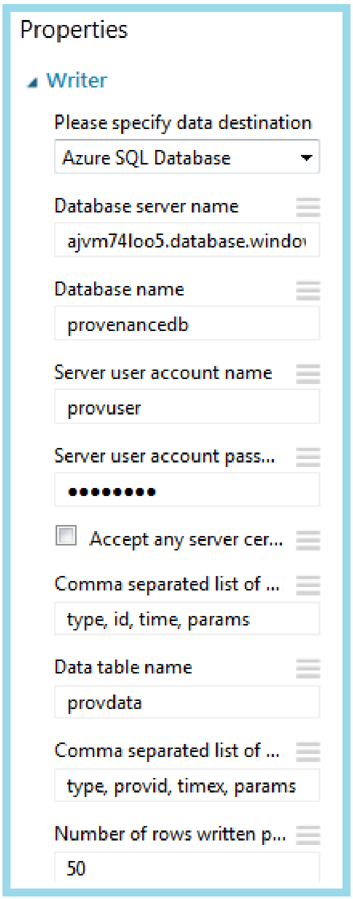
The modified experiment creates a list of PROV elements during execution. They represent the provenance trail of the model and also of its evaluation. Those elements are still encapsulated in a pandas data frame. The left side of Figure 3.8 shows the first few entries of the PROV elements. Since this data has to be still available after the execution to verify the workflow it has to be stored. To extract the information out of the Machine Learning Studio the *Writer Module* is used. This module can write the content of a data frame into an Azure SQL database. The right side of Figure 3.8 shows the used properties, consisting of information for connecting to the database and how the entries are mapped to the database schema.The database only consists of one table which contains the type, id and parameters of the PROV element.

Using the web service module is also an option for receiving the provenance trail. This can be done by publishing the workflow and calling it by using a REST request. However, this method seems to have an internal problem with the extensive use of python scripts before a web service output module. The response of the call always deviates from the provenance output displayed in the Machine Learning Studio GUI. A bug report has been filed to inform the development team of the Machine Learning Studio.

Once again a python script is used to extract the data out of the database into a viewable format. The script reads all database entries, sorts them by type and converts the information into the PROV-XML format. The output is an xml document shown in Listing A.1 in the appendix. The *ml* namespace represents an Machine Learning Studio namespace and can be adapted to any other preferred namespace policy.



type	id	params
agent	hager	prov:type='prov:Person', foaf:givenName='Hager'
agent	mlstudio	prov:type='prov:Software Agent'
actedOnBehalfOf	act1	delegate=mlstudio, responsible=hager
entity	cardataset	colsize=26, rowsize: 205
wasAssociatedWith	aw1	activity=projection1, agent=mlstudio
activity	projection1	prov:startedAtTime=2015-09-23 09:21:11, prov:endedAtTime=2015-09-23 09:21:36, exlude=normalized-losses
used	u1	usageTime=2015-09-23 09:21:11, activity=projection1, entity=cardataset
wasGeneratedBy	g1	generationTime=2015-09-23 09:21:36, entity=dataset1, activity=projection1
entity	dataset1	colsize=25, rowsize: 205



Properties

Writer

Please specify data destination	Azure SQL Database
Database server name	ajvm74loo5.database.windo
Database name	provenancedb
Server user account name	provuser
Server user account pass...	••••••••
☐ Accept any server cer...	
Comma separated list of ...	type, id, time, params
Data table name	provdata
Comma separated list of ...	type, provid, timex, params
Number of rows written p...	50

Figure 3.8: Result Data Frame, Writer Property List

Figure 3.9 shows the content of the xml file consisting of the different PROV elements and their relations. Since the purple rectangles represent the activities of the workflow their sequence is equal to the modules of the workflow. The yellow elements are entities which are either already available in the Machine Learning Studio or are generated during the workflow execution. The orange elements represent the agents respectively. Each element possesses additional information like shown for the *ml:Projection* activity or the *ml:linearReg1* entity.

3.3.2 Car Price Prediction Scoring Experiment

After finishing a model in the Azure Machine Learning Studio it can be converted into a Scoring Experiment. This allows the evaluation of new data with the previously created model. The Car Price Prediction Scoring Experiment, illustrated in 3.3, is the conversion result of the Car Price Prediction Experiment. The output of this Scoring Experiment is

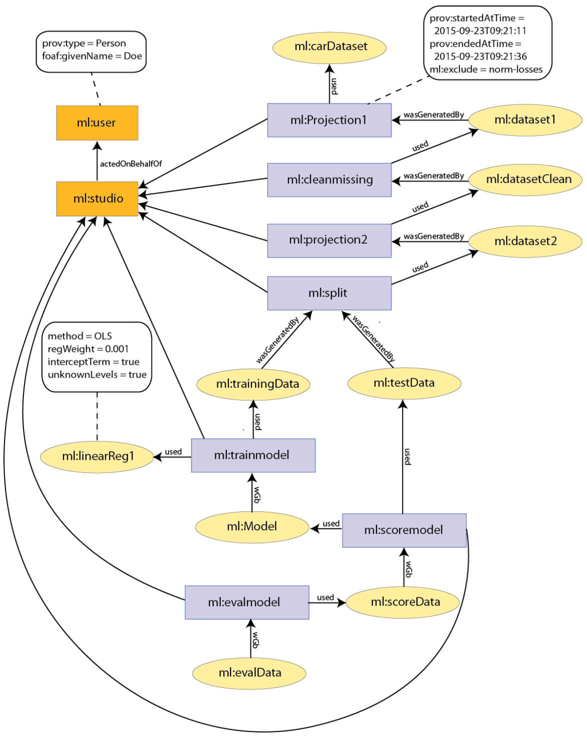
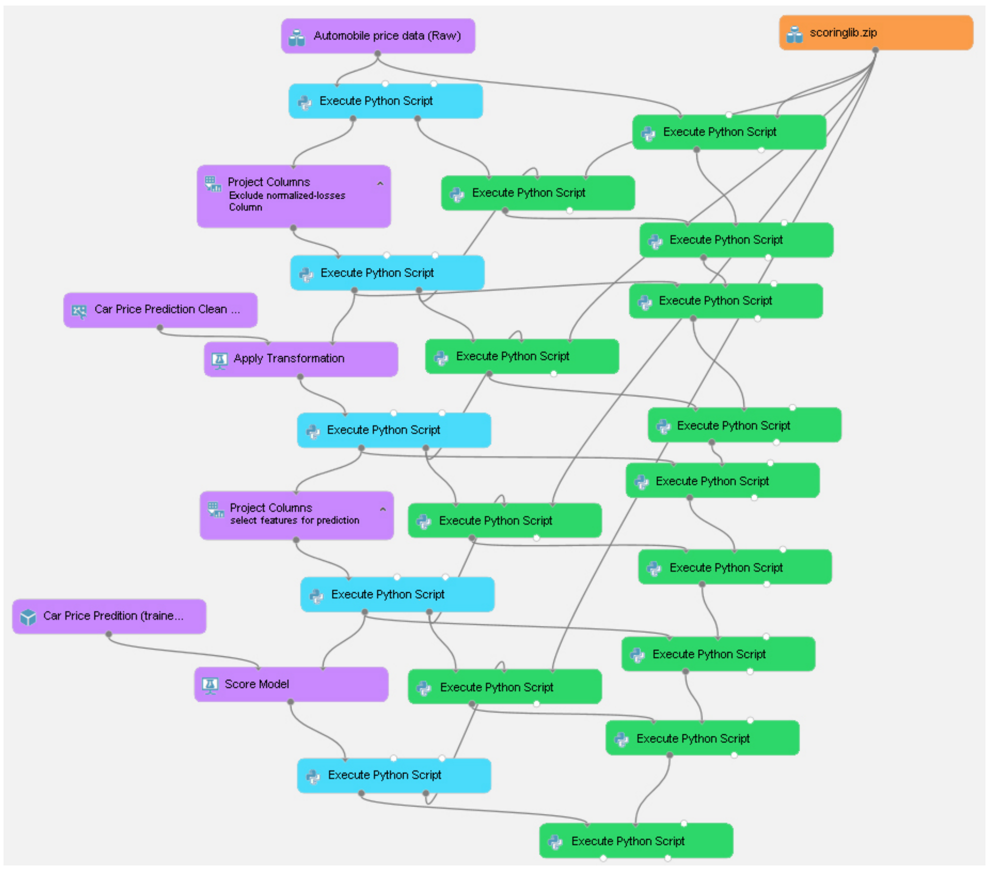


Figure 3.9: Provenance Trail of Experiment

a score value the model achieved with new data. When recording provenance information during the execution of this experiment it should be noted that it does not describe the provenance of the model itself but rather the provenance information of the scoring result created during the scoring experiment. The provenance of the model creation was already recorded during the Car Price Prediction Experiment described in the last chapter.

Since the model is already available as a module, the scoring experiment needs fewer components. For the modification of this workflow the following steps are performed. At first the zip bundle with the parameter properties of each workflow module is created. This is done manually as this information cannot be read from the closed modules. For the timestamp capturing system python script modules are added before and after each workflow module. By using the generic Python scripts additional Python Script modules are created to generate the PROV elements. To whole modification of the workflow for recording the provenance information was **finished in 35 minutes**. The web service modules were excluded since they only transport and not change the input and output data. The modified experiment is shown in Figure 3.10.

The purple modules represent the original workflow without the web service modules. The orange module to the upper right is the python library containing the information about the configuration of the modules. The blue and green modules are all python



Automobile price data (Raw)
scoringlib.zip
Execute Python Script
Project Columns
Exclude normalized-losses Column
Car Price Prediction Clean ...
Apply Transformation
Project Columns
select features for prediction
Car Price Predition (traine...
Score Model

Figure 3.10: Car Price Prediction Scoring Experiment & Provenance Recording

scripts. The blue modules record a timestamp before and after each activity. Each green module creates at least one PROV element. The green modules between the blue ones receive the timestamps and create the activity elements. The remaining modules on the right side create one of several PROV elements, append them to the received data frame and pass it to the next module.

3.4 Discussion

Recording provenance information allows to verify if a workflow with evolved resources or components is still consistent. By using a common standard like the PROV model this data can be interchanged between data analysts and results can be reproduced with workflow management systems running in different environments, thus improving the repeatability of workflows. Hence, an implementation for recording provenance contributes to the goals of this work.

To answer the first research question the possibilities of the Microsoft Azure Machine

Learning studio were examined. The used method involves the extension of a workflow with additional modules for recording provenance data. The resulting implementation approach offers a way to document the creation process of results with the help of data provenance. Another challenge was to map the available information into a suitable format. The PROV model offers a lot of possibilities for describing the execution of a workflow. By investigating the second research question the necessary information which can be collected was identified. The representation of the information collected by the chosen PROV elements describes the workflow execution in great detail.

After the generic scripts for recording the workflow provenance were created during the first implementation part, the extension of the Car Price Prediction Scoring Experiment was not much effort. However it still involves manual work increasing proportionally with the size of a workflow.

There are some limitations through the design of the Machine Learning Studio. The use of the Python Script module for recording information about the workflow is not intended by design. Python Script modules can only receive and forward DataFrames, a special format from the *pandas* library. In each script module the recorded information is combined into a newly created frame to enable forwarding. In addition the Python Script module has only one standard output port and two input ports. This limitation results in a huge number of extra modules which makes the structure of a modified workflow very confusing.

The biggest restriction is that a workflow module is a closed unit. This means that configuration information cannot be extracted automatically during runtime. As a workaround the properties of each used module were manually combined into a zip bundle. This bundle is used to provide the information to each provenance generating module during workflow execution. The execution time of the workflow components could not be extracted as well. The final solution uses additional Python Script modules before and after each workflow component to record timestamps. The real execution start and end lies between those timestamps.

When activating the workflow via API a strange behavior of the Python Script module was encountered. If several of these scripts are connected in a row only the first script module is executed. Since the created solution heavily uses the consecutively place modules the provenance does not work when the workflow is called via API. This behavior was reported in the official Azure ML Studio forum. The developers responded that they are aware of this issue and are working on it.

Recording information about the environment the workflow was executed in, like offered by Kepler and Pegasus, was also not available in this cloud-based environment. Such information is a crucial part of the provenance trail. Another aspect of the workflow which could not be observed was the evolution of the workflow itself. Such information has to be collected on a higher level than the workflow execution which is not possible without software development access.

The following points are recommended if Microsoft Azure wants to offer the possibility to record provenance information in the Machine Learning Studio:

- Enable automatic capturing of provenance information by collecting meta data internally.
- Display provenance via GUI or web service.
- Offer provenance export as XML or into a database.

A change of these modules is not necessary if the data is recorded automatically in the background of the workflow execution. It is reasonable to log information for each execution step like start and end time, module parameters and metadata of intermediate results and map them to a provenance model. The information is present in the software implementation and needs to be formatted, merged and provided to the user through the GUI or web service.

Providing information about the environment of a workflow execution, like program version, component version or library version, also enables users to verify if an error during a run was caused through the change of the environment.

A desirable use case would be if a user could, after activating provenance tracing, export the information directly out of the studio as an XML. When calling the workflow through a web service an additional parameter could to be used to also receive the provenance data as a JSON.

For complete provenance support there also should be an option for saving the data in a database. This data could either be saved directly in the cloud or in a database specified by the user.

CHAPTER 4

Workflow Support for Dynamic Data Citation

This chapter investigates the design or ways to provide dynamic data citation services as part of the ML Studio. The implementation presented in this chapter consists of a workflow and a modified database. The workflow is used as an interface for accessing data and enables the possibility to reproduce the exact results of a query which was executed when the database was in a different state. At first the basic design of the system is explained. The next section analyses which steps are necessary to realize the recommendations presented in [39]. The resulting implementation of a system consisting of a database and a workflow created in Microsoft Azure Machine Learning is documented in the subsequent section. The last part deals with the evaluation of the system and discusses the results obtained.

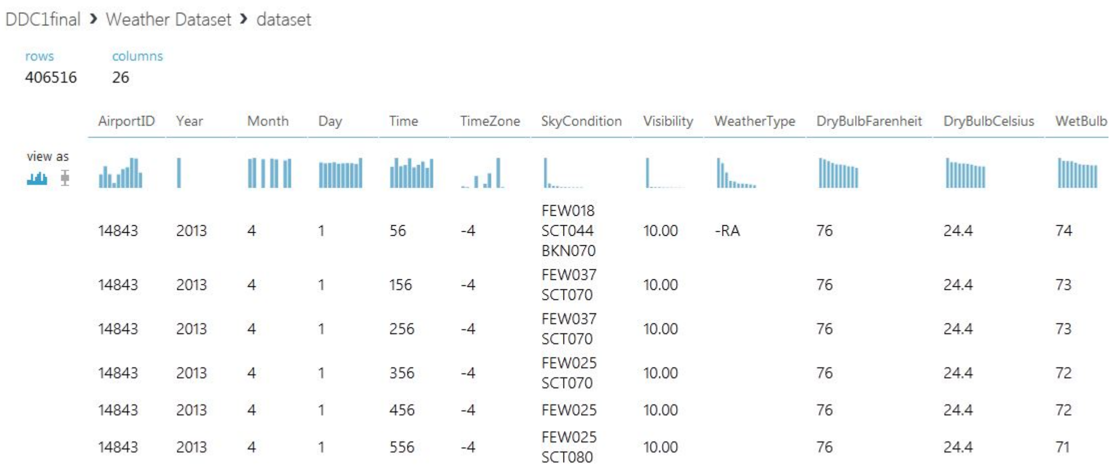
4.1 Initial System Design

The first main component of the initial system is a database which contains weather data. Since the database can change over time with each insert, update or delete command it represents the *dynamic* attribute of the system. The weather data is a sample dataset available in the Azure Machine Learning Studio. The entries were measured between April and October 2013 from airport weather stations in the United States [1]. As shown in Figure 4.1 it contains over 400.000 rows with 26 different attributes.

For the database schema the Fahrenheit values are excluded since they provide the same information as the Celsius values. The following attributes of the dataset are used for the definition of the database schema: AirportID, Year, Month, Day, Time, TimeZone, SkyCondition, Visibility, WeatherType, DryBulbCelsius, WetBulbCelsius,

[1] https://azure.microsoft.com/en-us/documentation/articles/machine-learning-use-sample-datasets/

DewPointCelsius, RelativeHumidity, WindSpeed, WindDirection, ValueForWindCharacter, StationPressure, PressureTendency, PressureChange, SeaLevelPressure, RecordType, HourlyPrecip, Altimeter



DDC1final › Weather Dataset › dataset

rows 406516

columns 26

view as

AirportID	Year	Month	Day	Time	TimeZone	SkyCondition	Visibility	WeatherType	DryBulbFarenheit	DryBulbCelsius	WetBulb
14843	2013	4	1	56	-4	FEW018 SCT044 BKN070	10.00	-RA	76	24.4	74
14843	2013	4	1	156	-4	FEW037 SCT070	10.00		76	24.4	73
14843	2013	4	1	256	-4	FEW037 SCT070	10.00		76	24.4	73
14843	2013	4	1	356	-4	FEW025 SCT070	10.00		76	24.4	72
14843	2013	4	1	456	-4	FEW025	10.00		76	24.4	72
14843	2013	4	1	556	-4	FEW025 SCT080	10.00		76	24.4	71

Figure 4.1: Weather Dataset

The primary key *wID* (weather data id) is added which is automatically incremented with each insert. The final table schema is shown in Listing A.2 in the appendix. For the DBMS the cloud-based Azure SQL Server is chosen because it can be directly accessed in a machine learning workflow with the *Reader* and *Writer module* of the ML Studio. The later is used to insert the whole dataset into the database.

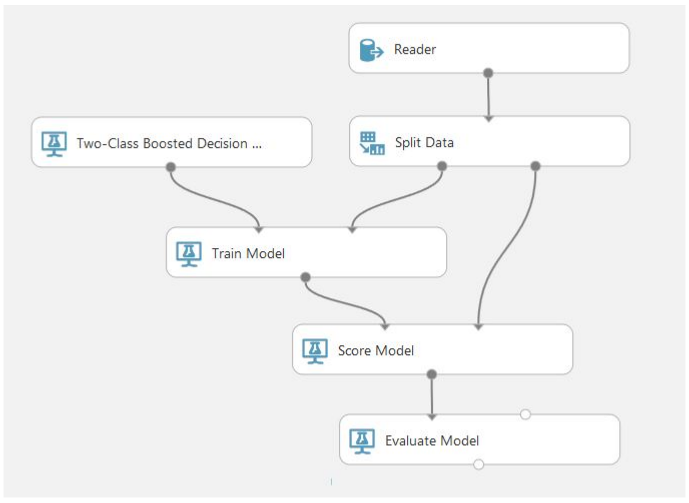


Figure 4.2: Generic Experiment using Weather Dataset

The second part of the initial system is generic Azure ML experiment shown in Figure

4.2 that accesses the database. The starting point of the workflow is the querying of the database executed in the Reader module. The query used is the only input parameter of the workflow and can be seen as the definition of a subset of the whole dataset.

This setup does, of course, not support the repeatability of experiments if the database is altered. The final workflow should enable the possibility of reusing the exact same subset of an earlier experiment even if the database has changed in the meantime. The workflow is thus extended with additional Azure ML modules which replace the Reader module in the intial workflow. The input parameter is once again the query to generate a subset from the database. The workflow's task is to process this query and to provide the data for the actual experiment. The final implementation should be useable for any experiment created in the Azure ML Studio.

4.2 Guidelines for Making Data Citable

The recommendations of the RDA Data Citation Working Group for making dynamic data citable are used as a foundation for the implementation in this chapter. The Working Group provides a detailed description as well as examples for implementation in [43]. As already mentioned in the related work chapter 2.5 the core principle is storing executed queries in a database. The system needs to be modified according to the recommendations to support dynamic data citation. After each recommendation is examined the necessary changes are explained.

4.2.1 R1: Data Versioning

R1 Recommendation: 'Apply versioning to ensure earlier states of data sets can be retrieved.'

The first task for this recommendation is to decide how the database versioning should be implemented. Pröll et al. suggests three approaches on how to change the database to support versioning [40]. One solution suggests the usage of a separate history table which is a feature included in the ISO SQL:2011 standard. However, this standard is not supported by the Microsoft SQL Server 2014 which is used for this work. It will be available for the SQL Server 2016 version in the form of *Temporal Tables* [2]. This system-versioned table would be the perfect solution for any future implementation because the database covers the versioning automatically.

A good alternative is the direct integration approach explained in [40]. This solution keeps the data storage needed and query complexity for retrieving subsets low. Since the database is not in a productive state the schema can easily be changed to meet the requirements. The database schema is extended by the version attribute **vID**. To

[2]https://msdn.microsoft.com/en-us/library/dn935015.aspx

still guarantee unique entries the Primary Key also needs to be changed to include this attribute.

The most common operations on a time series, like the weather dataset used, are *insert* statements. But with this assumption no versioning would be necessary in the first place. To completely support the recommendation the correct handling of *update* and *delete* operations still needs to be ensured.

If a database entry is updated the original row should be left unchanged. Instead a new entry with the same wID and a higher version needs to be created. The attributes should still be the same except for columns changed by the update statement. This requirement is solved by the creation of a SQL trigger *weatherdata_update*. The trigger is executed after each update of the weatherdata table. It saves all the attributes of the updated row into temporary variables and then rolls back the changes made by the update. The attributes have to be saved because they cannot be accessed after the rollback. Afterwards a new entry is created with the same wID, a higher version ID and the data saved in the variables. The *Isolation Level* of the Azure SQL server must be set to *Serializeable* to ensure atomicity during the execution of the update statement. The complete create statement of the trigger is shown in Listing A.3 in the appendix.

For delete operations a similar solution is used. The attribute *deleted*, which shows if an entry is marked as deleted, is added to the table definition with the type *bit*. The bit type is the Microsoft SQL Server's version of a Boolean and can either be 1 or 0. Once again another SQL trigger *weatherdata_delete* is executed after each delete operation. Instead of deleting the desired row it creates a new entry with the same attributes of the original row, a higher version number and the *deleted* field set to 1. The trigger definition is shown in Listing A.4 in the appendix.

Because of these triggers several versions with the same wID can exist in the database. For each request only the most recent version is of interest. Hence, each query used in the system needs to be modified as shown in Listing 4.1.

Listing 4.1: Modified Select Statement

```
SELECT * FROM weatherdata t1
WHERE windspeed = 10
AND vid = (SELECT MAX(vid)
           FROM weatherdata t2
           WHERE t2.wid = t1.wid)
AND deleted = 0
```

This query only returns the most recent entry if it is not deleted. It must to be noted that the additional sub-query does influence the performance of the database.

4.2.2 R2: Timestamping

R2 Recommendation: 'Ensure that operations on data are timestamped, i.e. any additions, deletions are marked with a timestamp.'

To fulfill this recommendation the attribute *tmstmp* is added to the table definition:

```
tmstmp DATETIME NOT NULL DEFAULT current_timestamp
```

As default the exact execution time of the creation operation is used. Every update and delete operations get transformed into an insert statement. The timestamp of the new entry always specifies the execution time of the given operation. Since this attribute is not meant to be changed afterwards, this simple modification of the schema covers all eventualities.

The new attribute allows to select only data elements available at a in a specific point in time. Each query needs to be extended by an additional *WHERE* clause like the example shown below:

Listing 4.2: Select Statement with Timestamp Clause

```
SELECT * FROM weatherdata t1
WHERE windspeed = 10
AND vid = (SELECT MAX(vid)
           FROM weatherdata t2
           WHERE t2.wid = t1.wid)
AND deleted = 0
AND tmstmp < '2012-01-25 16:51:44'
```

This query returns only entries which were available before the given timestamp. Together with the changes made for the R1 recommendation the exact reproduction of an earlier subset request is possible.

4.2.3 R3: Query Store Facilities

R3 Recommendation: 'Provide means for storing queries and the associated metadata in order to re-execute them in the future.'

Storing duplicates of data subsets would be a costly endeavor. Instead the Data Citation Working Group proposes a *Query Store* as an alternative. As mentioned earlier a query can be seen as the definition of a data subset. Hence, only this queries need to be saved in the query store. In addition it has to contain additional information to re-execute the query. The recommendation includes a list of metadata information with references to some other related recommendation:

- The original query as posed to the database
- A potentially re-written query created by the system (R4, R5)
- Hash of the query to detect duplicate queries (R4)
- Hash of the result set (R6)
- Query execution timestamp (R7)
- Persistent identifier of the data source
- Persistent identifier for the query (R8)
- Other metadata (e.g. author or creator information) required by the landing page (R11)

This query store is realized as an additional table for the database. Each item of the list is discussed in the respective recommendation section. From the above list the table schema shown in Listing 4.3 is created.

Listing 4.3: Query Store Table Schema

```
create table querystore (
  qid INTEGER IDENTITY(1,1) NOT NULL PRIMARY KEY,
  query VARCHAR(200) NOT NULL,
  qsort VARCHAR(200) NOT NULL,
  qhash VARCHAR(50) NOT NULL,
  rhash VARCHAR(50) NOT NULL,
  tmstmp DATETIME NOT NULL DEFAULT current_timestamp,
)
```

The *qid* serves as the unique primary key and is incremented each time a new query is stored. If a specific subset needs to be re-generated this key can be used as an input parameter for the workflow. The *query* attribute contains the original query string as sent to the workflow. The explanation for the other defined attributes can be found in 4.2.4 and 4.2.5. The reasons for excluding the PID information of the data source can be found in 4.2.7. The workflow should be designed to only store completely new queries. This means the workflow has to check for each query if it is already stored.

4.2.4 R4: Query Uniqueness, R5: Stable Sorting

R4 Recommendation: 'Re-write the query to a normalized from so that identical queries can be detected. Compute a check-sum of the normalized query to efficiently detect identical queries.'

Detecting identical queries can be a hard task since they can be expressed in different forms. Since complex transforming of queries is not the goal of this thesis an assumption for the input query of the workflow needs to be defined. The assumption is that a user does not directly access the workflow. Using a standard interface consisting of a form page and JavaScript would ensure the creation of standardized queries. Sorting the parameter name and value pairs alphabetically, like suggested in the recommendation, would be a simple method for standardization. Creating such a form page is not part of the implementation.

The workflow still needs to modify the input query because of the changes of the weather table to support versioning. These modifications are already defined in the sections 4.2.1 and 4.2.2. They extend the query to only select the newest version of an entry and exclude deleted rows. The final query string is saved in the query store as the *qsort* attribute.

The second part of the R4 recommendation is the checksum computation of the query. To compute a checksum the python library *hashlib* [3] is used in a *Python Script Module* which is part of the workflow. The hash function generates an md5 hash string of length 32 which can be used for duplicate detection. This value is saved as *qhash* attribute in the query store.

R5 Recommendation: 'Ensure that the sorting of the records in the data set is unambiguous and reproducible.'

The sequence of the records can have an influence on the experiment result. To make them reproducible a stable sorting of the input dataset is necessary. For this recommendation the same sorting strategy is used as described in the example of the RDA Guidelines. The result subset always is sorted by the weather id column in descending order followed by the sorting rules defined by the user. Even if the primary key also includes the version id, it can be ignored because only the highest version is selected. Since the *qsort* attribute of the query store is the extended query with all additional modifications it also contains this sorting clauses. This way a re-execution does not require any new modifications. An example for such an extended query is shown in Listing 4.4

[3] https://docs.python.org/2/library/hashlib.html

Listing 4.4: Query Transformation

Original Query:

```
SELECT * FROM weatherdata
WHERE drybulbcelsius = 15
ORDER BY drybulbcelsius ASC
```

Extended Query:

```
SELECT * FROM weatherdata t1
WHERE drybulbcelsius = 15
AND vid = (SELECT MAX(vid)
                  FROM weatherdata t2
                  WHERE t2.wid = t1.wid)
AND deleted = 0
AND tmstmp < '2012-01-25 16:51:44'
ORDER BY wid DESC, drybulbcelsius ASC
```

4.2.5 R6: Result Set Verification

R6 Recommendation: 'Compute fixity information (also referred to as checksum or hash key) of the query result set to enable verification of the correctness of a result upon re-execution.'

Creating a hash key of the result allows verifying if the re-generated subset has changed in any way. To keep the computation of this hash time efficient only the column names, weather ID and version ID of the result are used instead of the whole subset. These values are combined in a single string and used as input for an md5 hash function. The result hash is stored as the *rhash* attribute in the query store.

It is especially important to compare the resulting hash after the database has changed. We assume that an already executed query is saved in the query store. If the same query applied to the most current state of the database yields a different result two implications follow. It means the database has changed during the execution time of the two queries and that the query needs to be stored again with a new timestamp and different result hash. However if the original query is reproduced by using the query store entry with the original timestamp, the same result hash has to be obtained.

4.2.6 R7: Query Timestamping

Recommendation: 'Assign a timestamp to the query based on the last update to the entire database (or the last update to the selection of data affected by the query or the query execution time).'

In the RDA Guidelines different ways of assigning a query timestamp are presented. In order to ensure privacy the time of the last table change can be used as timestamp. Another option is to use the timestamp of the last change of only the subset. Since both methods require at least an additional query and privacy concerns can be neglected for this thesis the simplest solution is used.

The final implementation just uses the execution time of the query as timestamp. This time value is stored as the *tmstmp* attribute in the query store. Since this value should always be set during an insert the default value *current_timestamp*, defined in Listing A.2, is never used during the entry creation.

When re-executing a query this timestamp is used for an additional *WHERE* clause in the SQL statement:

```
SELECT * FROM weatherdata
WHERE windspeed = 10
AND tmstmp < querytimestamp
```

The final workflow should also offer the possibility to re-execute a stored query with the current timestamp. This is realized by an additional Boolean input parameter which is explained in detail in the implementation section. If the results are the same as for the original query no new entry has to be stored since the query hash is identical. However if the result hash is different a new entry with the same query and the new timestamp is created.

4.2.7 R8: Assigning Query PID, R9: Store the Query

R8 Recommendation: 'Assign a new PID to the query if either the query is new or if the result set returned from an earlier identical query is different due to changes in the data. Otherwise, return the existing PID of the earlier query to the user.'

Since the final system is only a prototype implementation, no real PID systems like DOIs or ARKs are used. Instead the primary key *qid* of the query store is used as unique identifier. This ID can be used as an input for the final workflow to re-execute older queries. There is also no PID used for the data source since only one dataset is used for the implementation.

To decide if a new query store entry needs to be created the same approach as described in the example of the R8 recommendation is used. When a new query is received its hash is searched in the query store. If it does not exist a new entry with all the needed information is created. If the hash does exist the result hash is checked as well. In case of a match both queries are identical and no new entry is needed.

R9 Recommendation: 'Store query and metadata (e.g. PID, original and normalized query, query and result set checksum, timestamp, superset PID, data set description, and other) in the query store.'

If a new query is identified the following information is stored in the query store of the system. Some of the values recommended in the RDA Guidelines are excluded. The reasons are described in the respective subsection of this chapter.

- Unique Identifier (qid)
- Original Query
- Normalized and sorted query
- Hash of the query
- Hash of the result set
- Query execution timestamp (R7)

The R9 recommendation also presents an isolation principle for storing the query. In order to ensure atomicity during the subset creation the data table must be locked for any other operations. This table lock is assured by setting the *Isolation Level* of the Azure SQL server to *Serializeable* which prevents any anomalies.

4.2.8 Remaining Recommendations R10-R14

R10 - Automated Citation Text Generation: 'Generate citation texts in the format prevalent in the designated community for lowering the barrier for citing and sharing the data. Include the PID into the citation text snippet. '

R11 - Landing Pages: 'Make the PIDs resolve to a human readable landing page that provides the data (via query re-execution) and metadata, including a link to the superset (PID of the data source) and citation text snippet. '

R12 - Machine Actionability: 'Provide an API / machine actionable landing page to access metadata and data via query re-execution.'

The above recommendations describe a platform that should use the created workflow and database tables to allow users to easily cite data. Creating such a platform is does not fit the focus of this thesis but is interesting for future work. For completeness the necessary steps for realizing each recommendation still will be discussed.

The R10 recommendation is about making the citation of data easier for users by generating citation texts out of the query store automatically. It is also mentioned that users should be able to specify information during the subset creation which should be stored as additional metadata in the query store. This could be done by creating an interface in form of a website for accessing the content of the query store and also uses the data citation workflow for subset creation.

This website could also include the landing pages for providing the data like described in recommendation R11. Such a page would have to display the information stored in the query store including all the metadata stated during uploading. The download button would have to trigger the machine learning workflow by a RESTful API call with the PID of the subset as attribute. To complete the functions of the website it should also provide an API to allow query re-execution like described in recommendation R12.

R13 - Technology Migration: 'When data is migrated to a new representation (e.g. new database system, a new schema or a completely different technology), migrate also the queries and associated fixity information. '

R14 - Migration Verification: 'Verify successful data and query migration, ensuring that queries can be re-executed correctly. '

The two recommendations describe the tasks necessary if the data is migrated to a new database system to ensure long term compatibility. Such a change would definitely require a modification of the workflow as well. But since such a migration will not be carried out during this thesis these recommendation require no additional implementation tasks.

4.3 Workflow Implementation

This section describes the final workflow developed according to the RDA guidelines shown in Figure 4.3. The modules in the blue rectangle present the original weather data experiment shown in 4.1.

The first module (A) provides the necessary input parameters. The first *Execute Python Script* module (C) processes them and identifies if an old subset from the query store needs to be re-generated or the request is a new query. In both cases the query string is transferred to the *Reader* module (B) which fetches the desired subset from the weather database table.

The Machine Learning Studio has only closed modules, which means that it is not possible to transmit the query to a Reader module. As a workaround the query string output from the Python module has to be copied by hand into the Reader (D) query field. This is shown in Figure 4.3 by the additional dashed line which does not exist in the real workflow. Because this is not possible during runtime this workflow needs to be executed twice. The first time the connection from module (C) to the second Python module (E) has to be removed and the query field of the reader (D) must be empty. Only this way

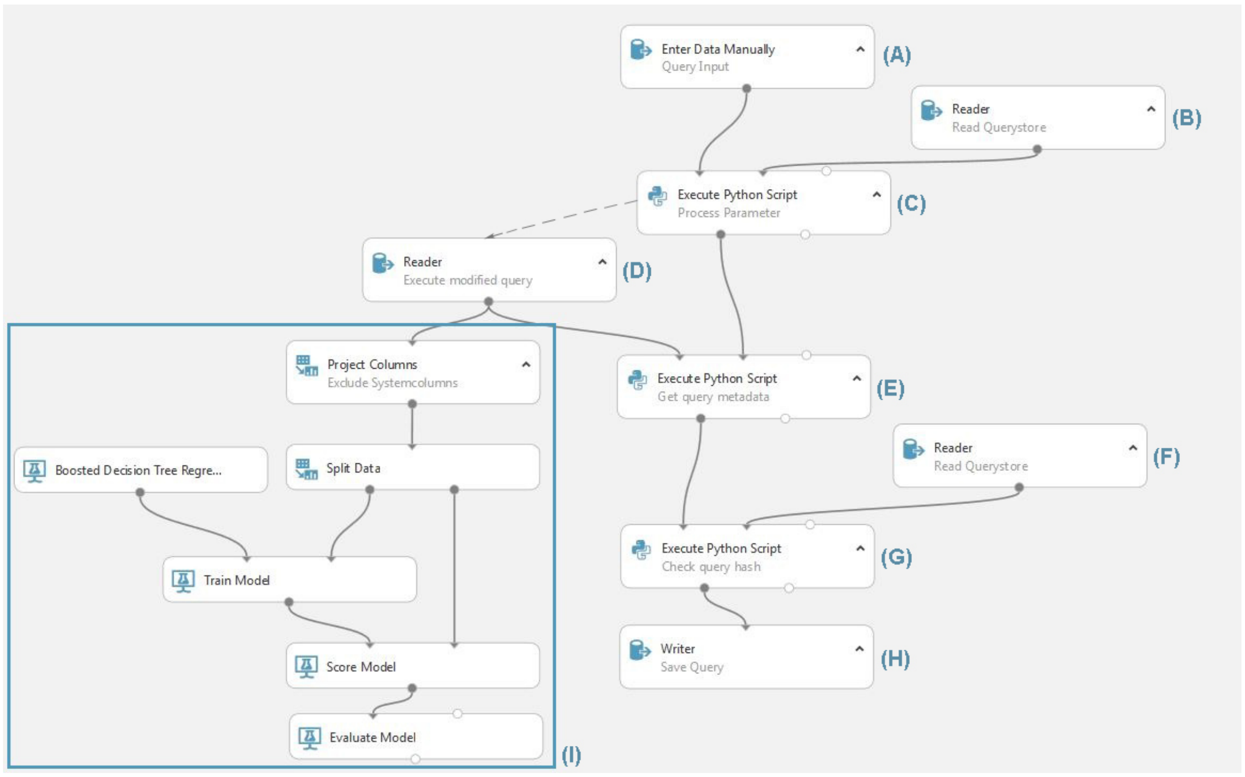


Figure 4.3: Workflow for Dynamic Data Citation

the final query is generated and then the workflow stops automatically. For the second run the query has to be copied into the Reader module (D) and the Python modules have to be reconnected.

Now the Reader (D) provides the requested subset to the actual experiment (I). The first module of the experiment removes the wid, vid and tmstmp attributes as they do not belong to the actual dataset. The experiment used for the workflow is generic and can be replaced by any other experiment. It is also possible to use a *Web Service Output* module instead. This way the subset can be received as a JSON and used for other applications like described in 4.2.8.

The remaining modules of the workflow have to check if the executed query needs to be saved in the query store. If this is the case, ambiguous information is generated and a new entry is inserted into the store. In the opposite scenario the query is already in the store and thus nothing has to be saved. The tasks of each module are described in greater detail in the respective section of this chapter.

4.3.1 Input Parameter Module (A)

The first module (A) of the workflow provides the initial parameter of the workflow. These parameters are entered manually in the property field of the module. If the workflow should be executed by a RESTful API call this module has to be replaced with a *Web Service Input* module. The input parameters of the module have the specification shown in Listing 4.5

Listing 4.5: Input Parameter Definition (A)

Input Type: DataFrame
Input Parameter:

Name	**Type**	**Description**	**Allowed Values**
query	String	query for subset generation	SQL query syntax
qid	Integer	for re-execution of existing query	>= 0
originalts	Boolean	use original timestamp	true,false

The *query* parameter represents the SQL query for the subset creation. This query is assumed to be in a standardized form like described in the R4 recommendation 4.2.4. The *qid* parameter is used for re-executing a specific query if the value is bigger than 0. If this is the case the first parameter is ignored and the workflow tries to re-execute the query with the specified id. The *originalts* parameter only is checked in case of a query re-execution. If it is 'true' the original timestamp of the stored query is used. In the other case no timestamp is used and the old query is applied to the current state of the database.

4.3.2 Process Parameter Module (C)

Listing 4.6: Input Port Definition (C)

Input DataFrame1
Description: Output of *Input Parameter Module (A)*

Input DataFrame2
Description: Entries from query store provided by Reader module (B)

Name	**Description**
qid	query ID
qsort	modified query
timestmp	query execution timestamp

The parameters are passed forward as a pandas DataFrame [4] to an *Execute Python Script* module (C). The main task of this script is to check the parameters and to identify if a completely new query is used or an old query is re-executed.

[4]http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html

For the first case the new query is modified to ensure a stable sorting to fulfill the R5 recommendation. In addition the *WHERE* clause is extended to exclude older versions and deleted entries like described in 4.2.1. The modified query is used as input for a hash function which generates an md5 hash string like described in 4.2.4. The output DataFrame shown in Listing 4.7 is created and is sent to the next module:

Listing 4.7: Output Port Definition (C)

Name	**Description**
query	original query
qsort	modified query
qhash	query hash

In case of a re-execution the entry with the *qid* parameter is received from the query store. Typically it is possible to access a database directly in a python script which would greatly simplify the whole workflow. However, this would require an internet connection and since the modules of the Machine Learning Studio are closed units, direct access is not possible. Because of this the necessary attributes of the query store have to be extracted with a *Reader* module (B). The module sends all the stored queries, their qid and the timestamp to the second input port of the python module. If the *originalts* input parameter is 'true' the script selects the query with the right qid, adds the timestamp in the *WHERE* clause and sends the query to the next module which executes the query. This implies that there is no new query that is inserted in the query store. An empty DataFrame is forwarded to the next module in this situation. This signalizes the remaining modules to skip their tasks and forward the empty DataFrame which results in no further actions.

If the *originalts* parameter is 'false' the same approach as for a new query is used. The old query is forwarded and is executed on the current state of the weather data table. The final script for this module is shown in Listing A.5 in the appendix.

4.3.3 Query Metadata Module (E)

Listing 4.8: Input Port Definition (E)

Input DataFrame1
Description: Resultset generated by Query

Input DataFrame2
Description: DataFrame from *Process Parameter Module (C)*

After the query is executed in the *Reader* module (D) and the result subset is sent to the experiment some additional metadata has to be captured. In this module the result hash is computed with an md5 function which uses the column names, all the weather IDs and versions IDs of the dataset as input as described in 4.2.5.

To fulfill R7 4.2.6 the execution time of the query is needed. The closed modules of the Machine Learning Studio however make it impossible to get the exact timestamp of the execution. Hence, the time directly after the execution module is recorded. Even if the timestamp does not resemble the true execution time and this can lead to inconsistencies, it is the only viablel solution in Azure ML.

The python script used for this module is shown in Listing A.6 in the appendix. The generated information is added to the received DataFrame. The final output frame shown in Listing 4.9 and is forwarded to the next module (G).

Listing 4.9: Output Port Definition (E)

Name	**Description**
query	original query
qsort	modified query
qhash	query hash
rhash	result hash
tmstmp	query execution timestamp

4.3.4 Hash Compare Module (G)

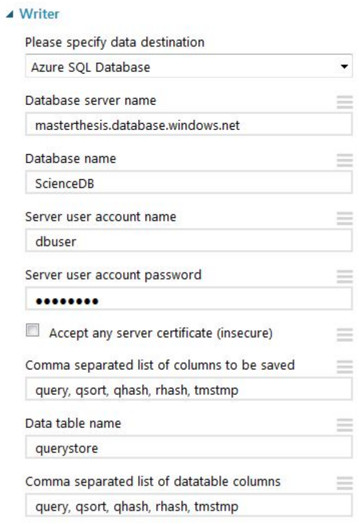
Listing 4.10: Input Port Definition (G)

Input DataFrame1
Description: Output of *Query Metadata Module (E)*

Input DataFrame2
Description: Entries from query store (F)

Name	**Description**
qhash	query hash
rhash	result set hash

In the final Python script module the result set verification described in 4.2.5 is performed. Once again the specific entry of the query store cannot be requested directly in the module. Hence, all the query and result hashes are requested by a *Reader* module (F) and received in input port 2. The module checks if an entry with the same query hash and result hash already exists in the query store. Both values have to be checked since a query can have different results if the database has changed. If no entry is stored yet, the whole DataFrame received in input port 1 is sent to a *Writer* module (H). It inserts this information as a new entry into the query store. The options of this module are shown


Writer

Please specify data destination: Azure SQL Database

Database server name: masterthesis.database.windows.net

Database name: ScienceDB

Server user account name: dbuser

Server user account password: ••••••••

Accept any server certificate (insecure)

Comma separated list of columns to be saved: query, qsort, qhash, rhash, tmstmp

Data table name: querystore

Comma separated list of datatable columns: query, qsort, qhash, rhash, tmstmp

Figure 4.4: Writer Module (F) Options

in Figure 4.4. If an entry matches the executed query an empty DataFrame is forwarded and no entry is created. The used script is shown in Listing A.7 in the appendix.

4.4 Evaluation

After the implementation several use cases are tested to evaluate if the workflow works as expected. As explained in the previous section the workflow always needs to be executed twice. The first time is needed to create the query which is manually copied to the *Reader* module. This is not explicitly mentioned for the following use cases of this chapter.

4.4.1 Subset Creation

For the first run the initial parameters (P1) shown in Listing 4.11 are used to create a subset S1:

Listing 4.11: Parameters P1

Parameter	**Value**
query	SELECT * FROM weatherdata WHERE drybulbcelsius = 28.3
qid	0
originalts	false

These parameters are inserted into the *Input Parameter* module manually before the workflow is started. The first Python Script module creates the query shown in Listing 4.12.

Listing 4.12: Normalized Query from P1

```
select * from weatherdata wd
where vid = (select max(vid) from weatherdata xd
             where wd.wid = xd.wid)
and deleted = 0
and drybulbcelsius = 28.3
order by wid DESC
```

The S1 subset created by this query through executing it in the Reader module has 7770 rows. Since this is the first query an entry is created in the query store illustrated in Listing 4.13. This entry represents the subset definition of S1.

Listing 4.13: Query Store Entry 1

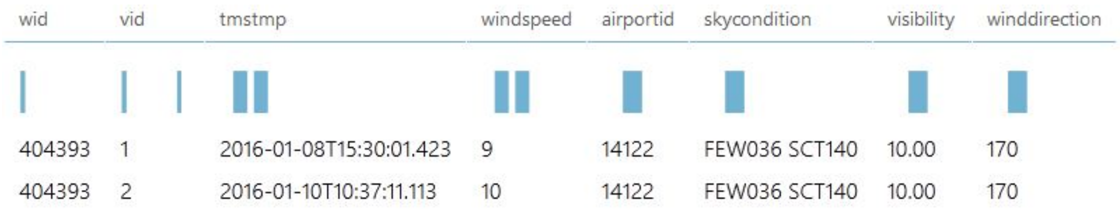
Attribute	**Value**
qid	1
query	select * from weatherdata where drybulbcelsius = '28.3'
qsort	select * from weatherdata wd where...
qhash	77b1070e32268ce385fe915ff17191a1
rhash	e297b690d14510de000fcf41d695cdeb
tmstmp	2016-01-10 10:36:53

4.4.2 Update Dataset Entries

When repeating the query from the last section the *Hash Compare* module identifies that the query is already stored and forwards an empty DataFrame. In order to alter the requested subset the first subset entry is changed with the following SQL statement:

```
UPDATE weatherdata
SET windspeed=10
WHERE wid = 404393
```

This update invoked the *weatherdata_update* trigger and creates a new version of the entry with the updated windspeed attribute. When querying the database directly the results showen in Figure 4.5 contain both versions with different timestamps.
After this change the workflow is once again executed with the parameter set P1. The obtained results are different even if the same query is used. Since the result hash is different a new entry shown in Listing 4.14 is created in the query store. Compared to the first entry it contains a different result hash and a new timestamp.



wid	vid	tmstmp	windspeed	airportid	skycondition	visibility	winddirection
404393	1	2016-01-08T15:30:01.423	9	14122	FEW036 SCT140	10.00	170
404393	2	2016-01-10T10:37:11.113	10	14122	FEW036 SCT140	10.00	170

Figure 4.5: Weatherdata Table Entries

Listing 4.14: Query Store Entry 2

Attribute	**Value**
qid	2
query	select * from weatherdata where drybulbcelsius = '28.3'
qsort	select * from weatherdata wd where...
qhash	77b1070e32268ce385fe915ff17191a1
rhash	49489e4a71a74acdc072d1d7b453d897
tmstmp	2016-01-10 10:43:20

4.4.3 Remove Dataset Entries

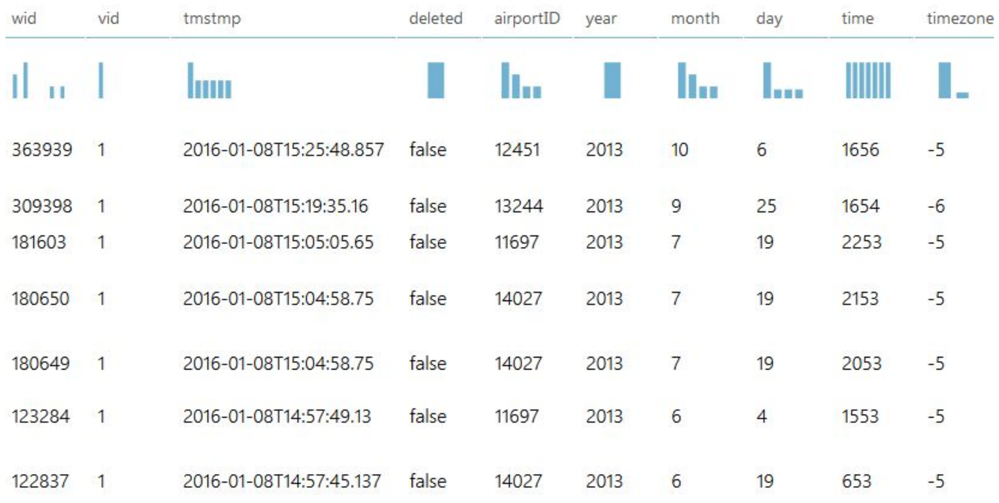
To create a different subset which does not contain any entry from the first subset the parameters (P2) shown in Listing 4.15 are used.

Listing 4.15: Parameters P2

Parameter	**Value**
query	SELECT * FROM weatherdata WHERE drybulbcelsius = 27.7
qid	0
originalts	true

The resulting subset S2 contains only seven entries 4.6.

The workflow once again identifies a new subset definition. Listing 4.16 contains the query store entry created.



wid	vid	tmstmp	deleted	airportID	year	month	day	time	timezone
363939	1	2016-01-08T15:25:48.857	false	12451	2013	10	6	1656	-5
309398	1	2016-01-08T15:19:35.16	false	13244	2013	9	25	1654	-6
181603	1	2016-01-08T15:05:05.65	false	11697	2013	7	19	2253	-5
180650	1	2016-01-08T15:04:58.75	false	14027	2013	7	19	2153	-5
180649	1	2016-01-08T15:04:58.75	false	14027	2013	7	19	2053	-5
123284	1	2016-01-08T14:57:49.13	false	11697	2013	6	4	1553	-5
122837	1	2016-01-08T14:57:45.137	false	14027	2013	6	19	653	-5

Figure 4.6: Subset S2

Listing 4.16: Query Store Entry 3

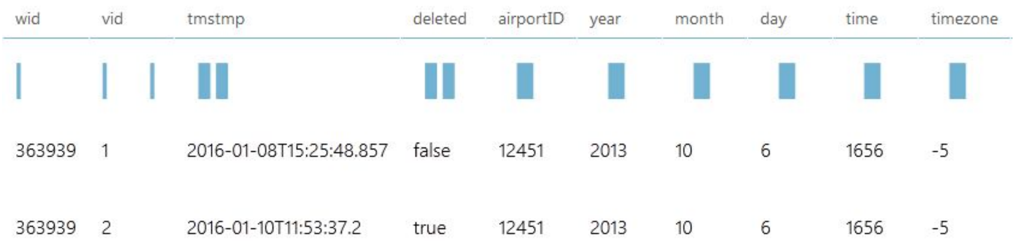
Attribute	**value**
qid	3
query	select * from weatherdata where drybulbcelsius = '27.7'
qsort	select * from weatherdata wd where...
qhash	c12934eb678f080f42399b28ffd2a9c2
rhash	7b868cff93dfb84a07cadb2f94bb9ccc
tmstmp	2016-01-10 11:32:46

To modify the subset S2 the first row is deleted with this statement:

```
DELETE FROM weatherdata
WHERE wid = 363939 AND vid=1
```

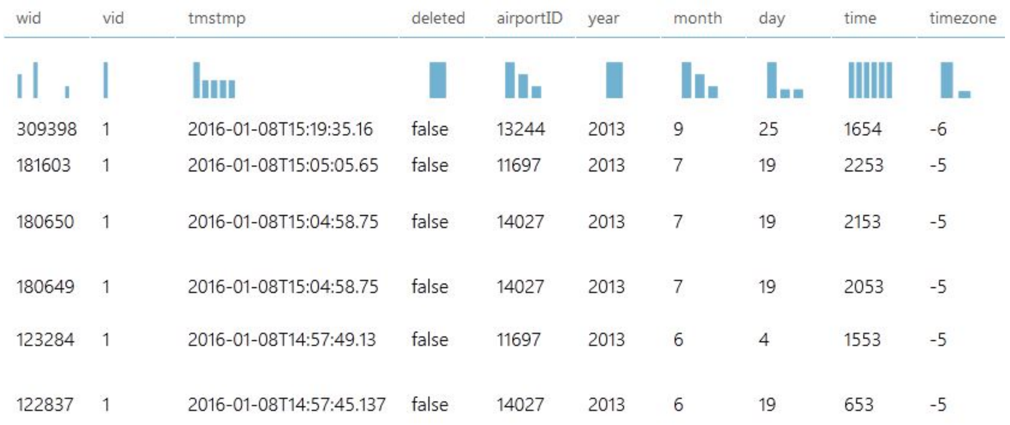
This SQL delete statement invokes the *weatherdata_delete* trigger. The trigger performs a rollback and instead of deleting the row, a new version is created with the *deleted* flag set to 1. Figure 4.7 shows both versions stored in the weatherdata table.

The workflow is executed again with the parameters P2 from Listing 4.15. Figure 4.8 contains the changed results (S3) of the workflow run. The new subset (S3) is saved by creating a query store entry with a different result hash and timestamp.



wid	vid	tmstmp	deleted	airportID	year	month	day	time	timezone
363939	1	2016-01-08T15:25:48.857	false	12451	2013	10	6	1656	-5
363939	2	2016-01-10T11:53:37.2	true	12451	2013	10	6	1656	-5

Figure 4.7: Weatherdata Table Entries



wid	vid	tmstmp	deleted	airportID	year	month	day	time	timezone
309398	1	2016-01-08T15:19:35.16	false	13244	2013	9	25	1654	-6
181603	1	2016-01-08T15:05:05.65	false	11697	2013	7	19	2253	-5
180650	1	2016-01-08T15:04:58.75	false	14027	2013	7	19	2153	-5
180649	1	2016-01-08T15:04:58.75	false	14027	2013	7	19	2053	-5
123284	1	2016-01-08T14:57:49.13	false	11697	2013	6	4	1553	-5
122837	1	2016-01-08T14:57:45.137	false	14027	2013	6	19	653	-5

Figure 4.8: S3 - originated from S2 after delete

4.4.4 Query Re-Execution 1

To re-execute a subset creation a *qid* stored in the query store is set as an initial parameter.

Listing 4.17: Parameters P3

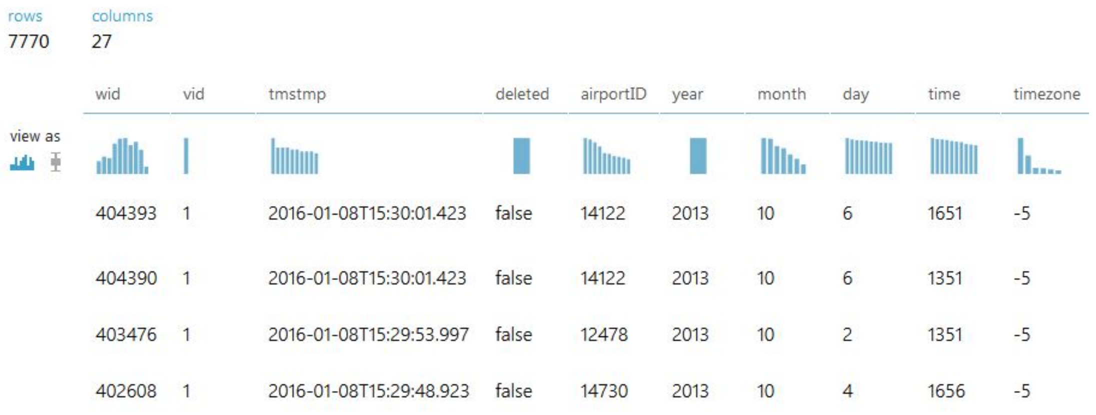
Parameter	**value**
query	none
qid	1
originalts	true

With the parameter set P3 shown in Listing 4.17 the first query of the evaluation is re-executed. Because the *originalts* parameter is set to 'true' the query is extended with the original timestamp by the *Process Parameter* module. The workflow extracts the query from the *qsort* attribute from query store entry shown in Listing 4.13. This query is modified with an additional timestamp check to request the state former state of the database. Listing 4.18 shows the final query executed by the workflow.

Listing 4.18: Normalized Query with Original Timestamp

```
select * from weatherdata wd
where tmstmp < '2016-01-08 15:36:53' and
vid = (select max(vid) from weatherdata xd
       where tmstmp < '2016-01-08 15:36:53'
       and  wd.wid = xd.wid)
and deleted = 0 and  drybulbcelsius = '28.3'
order by wid DESC
```

The result shown in Figure 4.9 is the original S1 subset with the unchanged version of the first row (wid = 404393). Since this is an exact re-execution the *Process Parameter* module does not forward any DataFrame and thus only the experiment section of the workflow is still executed.



rows 7770 columns 27

wid	vid	tmstmp	deleted	airportID	year	month	day	time	timezone
404393	1	2016-01-08T15:30:01.423	false	14122	2013	10	6	1651	-5
404390	1	2016-01-08T15:30:01.423	false	14122	2013	10	6	1351	-5
403476	1	2016-01-08T15:29:53.997	false	12478	2013	10	2	1351	-5
402608	1	2016-01-08T15:29:48.923	false	14730	2013	10	4	1656	-5

Figure 4.9: Re-generated S1 Subset

4.4.5 Query Re-Execution 2

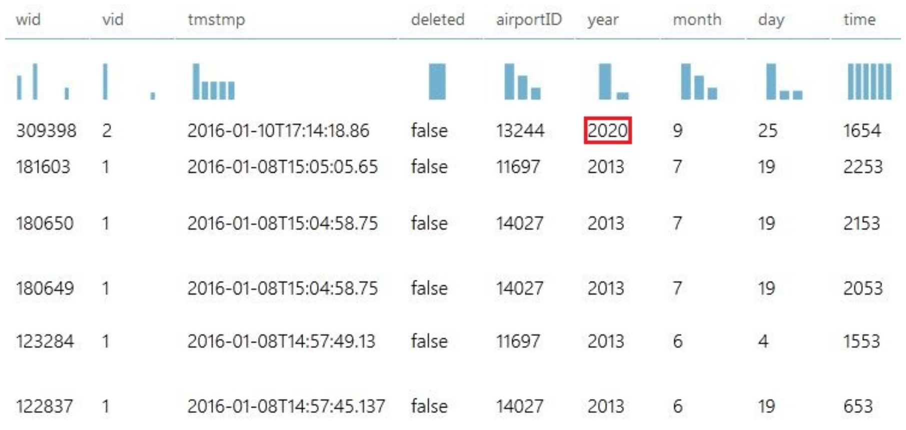
The other type of re-execution is done by using an old query on the current state of the database by setting the *originalts* parameter to 'false'. With the next initial parameters (P4) shown in Listing 4.19 the query for creating subset S2 is re-executed.

Listing 4.19: Parameters P4

Parameter	**value**
query	none
qid	3
originalts	false

Since this request can be a new subset, the workflow checks if the same query with a identical result hash is already stored in the query store. Because the query and the result hash matches the entry responsible for creating S3 no new entry is created.

To verify that a new entry is created, when the result set is different, the year attribute of the first row from subset S3 is updated. After this the same parameters (P4) are used once again which lead to the results with the changed attribute shown in Figure 4.10.



wid	vid	tmstmp	deleted	airportID	year	month	day	time
309398	2	2016-01-10T17:14:18.86	false	13244	2020	9	25	1654
181603	1	2016-01-08T15:05:05.65	false	11697	2013	7	19	2253
180650	1	2016-01-08T15:04:58.75	false	14027	2013	7	19	2153
180649	1	2016-01-08T15:04:58.75	false	14027	2013	7	19	2053
123284	1	2016-01-08T14:57:49.13	false	11697	2013	6	4	1553
122837	1	2016-01-08T14:57:45.137	false	14027	2013	6	19	653

Figure 4.10: Re-execution with current timestamp

Since the result hash is different a new query store entry shown in Listing 4.20 is created.

Listing 4.20: Query Store Entry 5

Attribute	**value**
qid	5
query	select * from weatherdata where drybulbcelsius = ’27.7’
qsort	select * from weatherdata wd where...
qhash	c12934eb678f080f42399b28ffd2a9c2
rhash	9c0ffae8d2527c08bed792e6885ae22e
tmstmp	2016-01-10 17:15:30

4.5 Discussion

The starting point of this chapter was an initial workflow which received the whole dataset from a SQL database and used it to conduct an experiment. The workflow and the database are modified according to the recommendations created by the RDA Working Group on Dynamic Data Citation [43]. The final implementation of this workflow supports the data citation of an evolving database. This makes it possible to repeat an experiment with the exact data subset even if the data source has changed.

By setting an initial query the workflow creates a subset which is used in an experiment.

For this subset a query store entry is created which contains the query, additional metadata and the execution timestamp. The workflow can detect subset duplicates by comparing hash strings. It can also re-generate the exact same subset by setting the corresponding query id from the query store as initial parameter. Since the database supports versioning the query is applied to the state the database was during the original query execution.

By using web service modules the workflow could be accessible by RESTful API calls. By removing the generic experiment the workflow could be directly used by a dynamic data citation application for subset generation.

The workflow is designed to support any dataset as long as it is stored in a database supporting data versioning. The versioning solution used for the database keeps the query complexity and storage space needed low. This is only possible because a time series dataset is used which only consists of one table. If the database would consist of several tables the versioning approach must be adapted. The ISO SQL:2011 standard introduces a versioning solution which can replace the custom approach used. However during the implementation no DBMS supporting this standard is available.

One drawback of using the Azure Machine Learning Studio is that recording the exact execution time of the query is not possible. Like for the provenance implementation the timestamp is recorded in the following module. This solution is not optimal but there are no alternatives possible in Azure ML.

Another issue is that there is no way to dynamically change properties of modules. This is once again a result of the closed unit concept of the Azure ML Studio. There is no way to transfer the modified subset query into a Reader module. As a consequence the citation workflow has to be executed two times in order to work properly. The first execution is necessary to generate the modified subset query. This query has to be copied manually to the Reader module which accesses the Query Store.

Another implication of this issue is that all the rows of the query store are extracted instead of just the rows needed. This is necessary because it would require an additional manual step to copy a more specific query into the *Reader* modules which access the query store. But since only a part of the attributes are retrieved by the *Reader* module and the query store should not have a huge amount of entries the performance is acceptable.

The Reader and Writer module are the only way to access a database. They only support SELECT and INSERT statements. It is not possible to properly lock or unlock the database after a change. Typically it is possible to access a database directly in a python script. This would greatly simplify the whole workflow structure as no additional Reader and Writer modules would be necessary. However, this would require an internet connection. Since the Python Script module is a closed units, direct access is not possible.

This is also the case for databases hosted by Azure.

The Microsoft Azure Machine Learning Studio could benefit from supporting repeatable experiments with a changing data source. The following points and their description are recommended to support such a feature.

- Offer a Database Management System with a versioning support
- Offer Modules for repeatable experiments (Read Subset, Evaluate Subset)
- Manage subsets in an internal Query Store

The modules of the workflow shown in 4.3 can be combined which would reduce the module amount greatly. The first modules (A,B,C) can be combined into a *Read Subset* module. Using a database with versioning is still required. This database could also be provided through a cloud service of Azure. The module has the following properties:

- Method
- Query or QID
- Data Source connection details
- Additional Information (optional)

For the *Method* property the options to define a new subset or reuse an existing subset are possible. Depending on this choice the next property is a text field for inserting the query or the qid of a stored query. The next property contains the connection details for the data source. The last property is a text field for specifying some custom details about the subset which would be saved in the query store if a new entry is created.

The query store could be implemented as a central database which is managed internally. It could contain entries from different users since the structure of the query store is the same for different data sources. The user only needs to receive the generated qid of his subset and an interface for searching in the query store running in the background. The module would need two output ports. The first port sends the query to a *Reader* module which executes the query. The *Reader* module (D) would need an additional input port in this case. The second port would send the query and additional metadata to the new *Evaluate Subset* module.

The *Evaluate Subset* module is a combination of the remaining modules (E,F,G,H) from the workflow shown in 4.3. Using this module is only necessary if a user wants to store their subset in the query store. Since the query store is handled by Azure the modules for accessing it the modules (F,H) would no longer be necessary. Since all the information is send by the *Read Subset* module no properties are required. The function of this module is to identify if the executed query already exists in the query store. If this is not the case a new entry is created. In both cases the output of this module is the information of the corresponding query store entry.

The implemented workflow documented in this chapter makes it possible to re-generate an earlier created dataset. This makes it possible to verify experiments by repeating them with the exact same dataset. Since the data can be reused even if the data source has changed the workflow supports dynamic data citation. Hence, the goal of improving the repeatability of data-driven workflows was achieved.

CHAPTER 5

Summary

The general goal of this work is to improve the verifiability and repeatability of data-driven workflows. In the first chapter the motivation for choosing this topic was presented. Three research questions were defined to achieve this goal. They deal with the concepts of *Data Provenance* and *Dynamic Data Citation* and how they can be applied to machine learning environments. We also pointed out that a practical implementation is necessary for researching the defined questions.

In the next chapter we provided an overview of the related work. This includes the most recent scientific work about Data Provenance and Data Citation. It is also investigated how these topics are used by some established workflow management systems. This was useful to decide which technologies and models to use for the following chapters. Researching these topics also provided crucial insights for the structure of the practical implementation.

The next two chapters investigate the research questions by developing ways to incorporate the researched concepts into a high-performance, cloud-based workflow environment. The Microsoft Machine Learning Studio is used to apply the created approaches. It offers the possibility to create, test and publish predictive analysis models in the form of workflows. To use the full potential of this workflow management system all the offered functionalities, possibilities as well as their limitations had to be analyzed. To prove the feasibility of the created concepts Azure ML workflows which support the created concepts are developed.

The third chapter is focused on the first two research questions. The first objective was to identify which provenance information can be collected in the cloud-based environment. Therefore the possibilities offered by the Machine Learning Studio had to be analyzed. The next task was to map the collected information into a data model. For this the

PROV model was used which offers the proper tools and methods for documenting each step of a workflow execution.

The implementation of this chapter involved the extension of a generic workflow with additional components. These components are modules containing Python scripts which collect provenance information during workflow execution. This information is combined into a Provenance trail and saved in a database. The generically designed scripts were used to extend a second workflow to test the efficiency of the created approach.

The resulting implementation approach offers a way to document the result creation process of a workflow with the help of data provenance. Since the used scripts are generically designed the method can be applied to any workflow created in the Microsoft Azure Machine Learning Studio. The created implementation approach can be used for other workflow management systems as well.

The generated provenance data helps to understand the creation process of the results. Since the used PROV model is a common standard, the recorded data can interchanged and compared between different workflow environments. The provenance information can also be used to verify if results are still consistent after components or resources of a workflow have changed. Thus we can say that the created implementation corresponds with the general focus of this thesis.

In the fourth chapter the last research question is covered. It investigates how the Azure ML Studio can incorporate dynamic data citation as a service. As a starting point a system consisting of a simple workflow experiment which uses an SQL database as source was used. The database provides the dynamic component to this system as it is a constantly changing data source.

During the research a solution approach for handling dynamic data was found. The guidelines published by the RDA Working Group for Data Citation offered the fundament for the created implementation. The steps necessary to realize each recommendation in the Azure ML Studio were analyzed. By realizing these steps the system was modified to support the Dynamic Data Citation concept. For this a versioning system was implemented for the database. The initial generic experiment was also modified with a set of modules. These modules identify if a new subset of the database is used or an older query is re-executed. The requested subset is retrieved from the database and provided to the actual experiment. The final implementation was thoroughly tested to ensure that the system works as intended. The created Python scripts and design methods can be used to extend other Azure ML experiments as well. The only requirement is to have a database with versioning.

The final solution makes it possible to repeat experiments with the exact same dataset, even if the database has changed in the meantime. By supporting data citation it offers a mechanism for verifying the results of an experiment. Hence, the created implementation contributes to the goal of this thesis.

5.1 Azure Limitations and Recommendations

The Microsoft Azure Machine Learning Studio was a great environment to work with. It offers a lot of possibilities for the development of the implementation. Since the ML Studio is mainly intended for creating machine learning experiments we encountered some limitations. As a result the components of the Studio were sometimes used in other ways than intended.

The following list summarizes all the issues and limitations encountered and how they were handled. This does not mean that some features of the Azure ML studio have an unsatisfactory design, but rather that their intended function did not fit for the tasks conducted during this work. Each point is described in detail in the discussions of chapter 3.4 and 4.5.

- Most modules can only communicate by using DataFrames. In order to forward a simple string a complex data object has to be generated.
  Workaround: Create DataFrame in Python Script.

- The Python Script module has only two input ports and one output port.
  Workaround: The data output of only two modules could be combined by a Python Script modules. Additional modules were necessary to combine provenance information from several modules.

- Modules are closed units, there is no way to dynamically extract or change properties.
  Workaround: Combining property information into a ZIP bundle which can be accessed during runtime.

- The execution time of a module cannot be extracted directly from a module.
  Workaround: Additional Modules (Python Scripts) for time recording were used before and after each module of interest.

- Different Python Script behavior when activating a workflow via API, instead of using the Machine Learning Studio. If several Script modules are connected in a row only the first module is executed.
  This was reported to Azure ML developers. They are aware of the issue and are working on it.

- Recording information about the environment the workflow was executed in was not available in this cloud-based environment. Such information has to be collected on a higher level and was not possible without developer access.

- Setting a modified query for a Reader module was not possible during runtime. Workaround: As a consequence the citation workflow must be executed twice.
- The only way to access a database is the usage of SELECT and INSERT statements with a Reader or Writer module. It is not possible to manually lock or unlock the database during crucial changes.
- Since the Python Script modules are executed in a closed environment they cannot access Azure databases.
  Workaround: Extensive use of Reader and Writer modules.

The cloud-based Azure platform could benefit from supporting the methods presented in this thesis. Therefore the following recommendations were created for improving the verifiability and repeatability of workflows created in their machine learning environment:

General

- Offer a possibility for dynamically changing module properties (setting query for Reader Module during runtime).
- Option for recording the execution time of a module.
- Extend the port number of Python Script modules.
- Offer a way to send specific sql commands to a database.
- Provide information about the execution environment.

Data Provenance

- Enable automatic capturing of provenance information by automatically collecting meta data during workflow execution.
- Using the PROV model for describing the Provenance information[29].
- Display provenance information via GUI or web service.
- Offer provenance data export as XML or into a database.

Data Citation

- Offer a Database Management System with versioning support.
- Offer modules for repeating experiments with the same data (Read Subset, Evaluate Subset).
- Manage subsets in an internal Query Store by implementing the data citation functionality as a service centrally within the Azure ML Studio.

5.2 Future Work

Finally, we give the following suggestions for further research:

The applicability of the created method for recording data for other workflow management systems needs to be further investigated. The creation process of such applications works in a similar way. By using the same structure of the approach presented in this thesis it should be possible to record provenance information in other workflow environments. It should be possible to create almost the same provenance data in different applications. Researching this matter could help to further improve the verification of data-driven experiments.

Also the provenance implementation could be extended with a method for answering provenance related queries. The recorded provenance information is currently only stored in a single database table. The focus of this thesis was to create a method for record this data during workflow execution. But a system which fully supports data provenance also requires a convenient method for accessing the stored information.

Finally the remaining RDA guidelines for making dynamic data citable could be realized. The necessary steps for each recommendation are discussed in 4.2.8. This basically comprises the extension of the data citation implementation. This includes the implementation of an interface for triggering the created data citation workflow and accessing the query store.

APPENDIX A

Appendix

A.1 Provenance XML

Description: Provenance information collected during the execution of the Car Price Prediction Experiment 3.3.1.

Listing A.1: Provenance XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<prov:document xmlns:prov="http://www.w3.org/ns/prov#"  xmlns:ml="http://machinelearning.example.
     com/" xmlns:dct="http://purl.org/dc/terms/" xmlns:foaf="http://xmlns.com/foaf/0.1/">

  <!-- Entities -->

  <prov:entity prov:id="ml:cardataset">
        <ml:colsize>26</ml:colsize> <ml:rowsize>205</ml:rowsize>
   </prov:entity>
  <prov:entity prov:id="ml:dataset1">
        <ml:colsize>25</ml:colsize> <ml:rowsize>205</ml:rowsize>
  </prov:entity>
  <prov:entity prov:id="ml:datasetclean">
        <ml:colsize>25</ml:colsize> <ml:rowsize>193</ml:rowsize>
  </prov:entity>
  <prov:entity prov:id="ml:dataset2">
        <ml:colsize>10</ml:colsize> <ml:rowsize>193</ml:rowsize>
  </prov:entity>
  <prov:entity prov:id="ml:testdata">
        <ml:colsize>10</ml:colsize> <ml:rowsize>48</ml:rowsize>
  </prov:entity>
  <prov:entity prov:id="ml:traingdata">
        <ml:colsize>10</ml:colsize> <ml:rowsize>145</ml:rowsize>
  </prov:entity>
  <prov:entity prov:id="ml:linearReg1">
        <ml:params method="OLS" regulationWeight="0.001"
        interceptTerm="true" allowUnknownLevels="true"/>
  </prov:entity>
  <prov:entity prov:id="ml:model"/>
  <prov:entity prov:id="ml:scoredata">
        <ml:colsize>11</ml:colsize> <ml:rowsize>48</ml:rowsize>
  </prov:entity>
  <prov:entity prov:id="ml:evaldata">
        <ml:mae>1755.564342</ml:mae> <ml:rmse>2488.708654</ml:rmse>
        <ml:rae>0.29321</ml:rae> <ml:rse>0.091937</ml:rse>
        <ml:detCoeff>0.908063</ml:detCoeff>
  </prov:entity>

  <!-- Activities -->

  <prov:activity prov:id="ml:projection1">
    <prov:startTime>2015-09-23T09:21:11</prov:startTime>
    <prov:endTime>2015-09-23T09:21:36</prov:endTime>
    <ml:params exclude="normalized-losses"/>
  </prov:activity>
  <prov:activity prov:id="ml:cleanmissing">
    <prov:startTime>2015-09-23T09:21:36</prov:startTime>
    <prov:endTime>2015-09-23T09:22:06</prov:endTime>
  </prov:activity>
  <prov:activity prov:id="ml:projection2">
    <prov:startTime>2015-09-23T09:22:06</prov:startTime>
    <prov:endTime>2015-09-23T09:22:37</prov:endTime>
        <ml:params include="make,body-style,wheel-base,engine-size,
␣␣␣␣␣␣␣␣horsepower,peak-rpm,highway-mpg,price,fuel-type,drive-wheels"/>
  </prov:activity>
  <prov:activity prov:id="ml:split">
    <prov:startTime>2015-09-23T09:22:37</prov:startTime>
    <prov:endTime>2015-09-23T09:22:59</prov:endTime>
    <ml:params trainingfraction="0.75"/>
  </prov:activity>
  <prov:activity prov:id="ml:trainmodel">
    <prov:startTime>2015-09-23T09:23:02</prov:startTime>
    <prov:endTime>2015-09-23 09:23:43</prov:endTime>
     <ml:params column="price"/>
  </prov:activity>
  <prov:activity prov:id="ml:scoremodel">
    <prov:startTime>2015-09-23T09:22:59</prov:startTime>
    <prov:endTime>2015-09-23T09:23:43</prov:endTime>
  </prov:activity>
  <prov:activity prov:id="ml:evalmodel">
    <prov:startTime>2015-09-23T09:23:43</prov:startTime>
    <prov:endTime>2015-09-23T09:24:15</prov:endTime>
  </prov:activity>

  <!-- Usage and Generation -->

  <prov:used prov:id="u1">
    <prov:activity prov:ref="ml:projection1"/>
    <prov:entity prov:ref="ml:cardataset"/>
    <prov:atTime>2015-09-23T09:21:11</prov:atTime>
  </prov:used>
  <prov:wasGeneratedBy prov:id="g1">
    <prov:entity prov:ref="ml:dataset1"/>
    <prov:activity prov:ref="ml:projection1"/>
    <prov:atTime>2015-09-23T09:21:36</prov:atTime>
  </prov:wasGeneratedBy>
  <prov:used prov:id="u2">
    <prov:activity prov:ref="ml:cleanmissing"/>
    <prov:entity prov:ref="ml:dataset1"/>
    <prov:atTime>2015-09-23T09:21:36</prov:atTime>
  </prov:used>
  <prov:wasGeneratedBy prov:id="g2">
    <prov:entity prov:ref="ml:datasetclean"/>
    <prov:activity prov:ref="ml:cleanmissing"/>
    <prov:atTime>2015-09-23T09:22:06</prov:atTime>
  </prov:wasGeneratedBy>
  <prov:used prov:id="u3">
    <prov:activity prov:ref="ml:projection2"/>
    <prov:entity prov:ref="ml:datasetclean"/>
    <prov:atTime>2015-09-23T09:22:06</prov:atTime>
  </prov:used>
  <prov:wasGeneratedBy prov:id="g3">
    <prov:entity prov:ref="ml:dataset2"/>
    <prov:activity prov:ref="ml:projection2"/>
    <prov:atTime>2015-09-23T09:22:37</prov:atTime>
  </prov:wasGeneratedBy>
  <prov:used prov:id="u4">
    <prov:activity prov:ref="ml:split"/>
    <prov:entity prov:ref="ml:dataset2"/>
  <prov:atTime>2015-09-23T09:22:37</prov:atTime>
</prov:used>
<prov:wasGeneratedBy prov:id="g4">
  <prov:entity prov:ref="ml:testdata"/>
  <prov:activity prov:ref="ml:split"/>
  <prov:atTime>2015-09-23T09:22:59</prov:atTime>
</prov:wasGeneratedBy>
<prov:used prov:id="u5">
  <prov:activity prov:ref="ml:scoremodel"/>
  <prov:entity prov:ref="ml:testdata"/>
  <prov:atTime>2015-09-23T09:22:59</prov:atTime>
</prov:used>
<prov:wasGeneratedBy prov:id="g5">
  <prov:entity prov:ref="ml:trainingdata"/>
  <prov:activity prov:ref="ml:split"/>
  <prov:atTime>2015-09-23T09:22:59</prov:atTime>
</prov:wasGeneratedBy>
<prov:used prov:id="u6">
  <prov:activity prov:ref="ml:trainmodel"/>
  <prov:entity prov:ref="ml:linearReg1"/>
</prov:used>
<prov:used prov:id="u6">
  <prov:activity prov:ref="ml:trainmodel"/>
  <prov:entity prov:ref="ml:trainingdata"/>
  <prov:atTime>2015-09-23T09:23:02</prov:atTime>
</prov:used>
<prov:wasGeneratedBy prov:id="g5">
  <prov:entity prov:ref="ml:model"/>
  <prov:activity prov:ref="ml:trainmodel"/>
  <prov:atTime>2015-09-23T09:23:43</prov:atTime>
</prov:wasGeneratedBy>
<prov:used prov:id="u7">
  <prov:activity prov:ref="ml:scoremodel"/>
  <prov:entity prov:ref="ml:model"/>
  <prov:atTime>2015-09-23T09:22:59</prov:atTime>
</prov:used>
<prov:wasGeneratedBy prov:id="g6">
  <prov:entity prov:ref="ml:scoredata"/>
  <prov:activity prov:ref="ml:scoremodel"/>
  <prov:atTime>2015-09-23T09:23:43</prov:atTime>
</prov:wasGeneratedBy>
<prov:used prov:id="u8">
  <prov:activity prov:ref="ml:evalmodel"/>
  <prov:entity prov:ref="ml:scoredata"/>
  <prov:atTime>2015-09-23T09:23:43</prov:atTime>
</prov:used>
<prov:wasGeneratedBy prov:id="g7">
  <prov:entity prov:ref="ml:evaldata"/>
  <prov:activity prov:ref="ml:evalmodel"/>
  <prov:atTime>2015-09-23T09:24:15</prov:atTime>
</prov:wasGeneratedBy>

<!-- Agents and Responsibility -->

<prov:agent prov:id="ml:user">
  <prov:type>prov:Person</prov:type>
  <foaf:givenName>Doe</foaf:givenName>
</prov:agent>
<prov:agent prov:id="ml:studio">
  <prov:type>prov:SoftwareAgent</prov:type>
</prov:agent>
<prov:actedOnBehalfOf>
  <prov:delegate prov:ref="ml:studio"/>
  <prov:responsible prov:ref="ml:user"/>
</prov:actedOnBehalfOf>
<prov:wasAssociatedWith>
  <prov:activity prov:ref="ml:projection1"/>
  <prov:agent prov:ref="ml:studio"/>
</prov:wasAssociatedWith>
<prov:wasAssociatedWith>
  <prov:activity prov:ref="ml:cleanmissing"/>
  <prov:agent prov:ref="ml:mlstudio"/>
</prov:wasAssociatedWith>
```

```
  <prov:wasAssociatedWith>
    <prov:activity prov:ref="ml:projection2"/>
    <prov:agent prov:ref="ml:mlstudio"/>
  </prov:wasAssociatedWith>
  <prov:wasAssociatedWith>
    <prov:activity prov:ref="ml:split"/>
    <prov:agent prov:ref="ml:studio"/>
  </prov:wasAssociatedWith>
  <prov:wasAssociatedWith>
    <prov:activity prov:ref="ml:trainmodel"/>
    <prov:agent prov:ref="ml:mlstudio"/>
  </prov:wasAssociatedWith>
  <prov:wasAssociatedWith>
    <prov:activity prov:ref="ml:scoremodel"/>
    <prov:agent prov:ref="ml:mlstudio"/>
  </prov:wasAssociatedWith>
  <prov:wasAssociatedWith>
    <prov:activity prov:ref="ml:evalmodel"/>
    <prov:agent prov:ref="ml:mlstudio"/>
  </prov:wasAssociatedWith>
</prov:document>
```

A.2 WeatherData Schema

Description: Table schema definition of the weather data which is used as a data source for the Workflow for supporting dynamic data citation.

Listing A.2: Weatherdata Table Schema

```
create table weatherdata (
wid INTEGER IDENTITY(1,1) NOT NULL,
vid INTEGER DEFAULT 1 NOT NULL,
tmstmp DATETIME NOT NULL DEFAULT current_timestamp,
airportID INTEGER,
year INTEGER,
month INTEGER,
day INTEGER,
time INTEGER,
timezone INTEGER,
skycondition VARCHAR(50),
visibility FLOAT,
weathertype VARCHAR(50),
drybulbcelsius FLOAT,
wetbulbcelsius FLOAT,
dewpointcelsius FLOAT,
relativehumidity INTEGER,
windspeed INTEGER,
winddirection VARCHAR(20),
valueforwindcharacter INTEGER,
stationpressure FLOAT,
pressuretendency FLOAT,
pressurechange INTEGER,
sealevelpressure FLOAT,
recordtype VARCHAR(20),
hourlyprecip VARCHAR(20),
altimeter FLOAT,
PRIMARY KEY (wid, vid)
)
```

A.3 WeatherData Update Trigger

Description: This trigger is necessary for the versioning solution of the database used in chapter 4. Instead of updating a row this trigger creates a new entry with the changed attributes.

Listing A.3: Update Trigger

```
CREATE TRIGGER weatherdata_update
ON weatherdata
AFTER UPDATE
as
begin
  DECLARE @wid int, @vid int,
  @airportID INTEGER,
  @year INTEGER,
  @month INTEGER,
  @day INTEGER,
  @time INTEGER,
  @timezone INTEGER,
  @skycondition VARCHAR(50),
  @visibility FLOAT,
  @weathertype VARCHAR(50),
  @drybulbcelsius FLOAT,
  @wetbulbcelsius FLOAT,
  @dewpointcelsius FLOAT,
  @relativehumidity INTEGER,
  @windspeed INTEGER,
  @winddirection VARCHAR(20),
  @valueforwindcharacter INTEGER,
  @stationpressure FLOAT,
  @pressuretendency FLOAT,
  @pressurechange INTEGER,
  @sealevelpressure FLOAT,
  @recordtype VARCHAR(20),
  @hourlyprecip VARCHAR(20),
  @altimeter FLOAT

  SET @wid = (SELECT wid FROM inserted)
  SET @VID = (SELECT vid FROM inserted) + 1
  SET @airportID = (SELECT airportID FROM inserted)
  SET @year = (SELECT year FROM inserted)
  SET @month = (SELECT month FROM inserted)
  SET @day = (SELECT day FROM inserted)
  SET @time = (SELECT time FROM inserted)
  SET @timezone = (SELECT timezone FROM inserted)
  SET @skycondition = (SELECT skycondition FROM inserted)
  SET @visibility = (SELECT visibility FROM inserted)
  SET @weathertype = (SELECT weathertype FROM inserted)
  SET @drybulbcelsius = (SELECT drybulbcelsius FROM inserted)
  SET @wetbulbcelsius = (SELECT wetbulbcelsius FROM inserted)
  SET @dewpointcelsius = (SELECT dewpointcelsius FROM inserted)
  SET @relativehumidity = (SELECT relativehumidity FROM inserted)
  SET @windspeed = (SELECT windspeed FROM inserted)
  SET @winddirection = (SELECT winddirection FROM inserted)
  SET @valueforwindcharacter = (SELECT valueforwindcharacter FROM inserted)
  SET @stationpressure = (SELECT stationpressure FROM inserted)
  SET @pressuretendency = (SELECT pressuretendency FROM inserted)
  SET @pressurechange = (SELECT pressurechange FROM inserted)
  SET @sealevelpressure = (SELECT sealevelpressure FROM inserted)
  SET @recordtype = (SELECT recordtype FROM inserted)
  SET @hourlyprecip = (SELECT hourlyprecip FROM inserted)
  SET @altimeter = (SELECT altimeter FROM inserted)

  rollback transaction
  set identity_insert weatherdata on
   insert into weatherdata(wid, vid, airportID, year, month, day, time, timezone ,
       skycondition, visibility, weathertype ,drybulbcelsius, wetbulbcelsius,
       dewpointcelsius, relativehumidity, windspeed, winddirection,
       valueforwindcharacter, stationpressure, pressuretendency, pressurechange,
       sealevelpressure, recordtype, hourlyprecip, altimeter)
  values(@WID, @VID, @airportID, @year, @month, @day, @time, @timezone ,@skycondition,
      @visibility, @weathertype ,@drybulbcelsius, @wetbulbcelsius, @dewpointcelsius,
      @relativehumidity, @windspeed, @winddirection, @valueforwindcharacter,
      @stationpressure, @pressuretendency, @pressurechange, @sealevelpressure,
      @recordtype, @hourlyprecip, @altimeter)
  set identity_insert weatherdata off
end
```

A.4 WeatherData Delete Trigger

Description: This trigger is necessary for the versioning solution of the database used in chapter 4. Instead of deleting a row this trigger creates a new entry with the *deleted* attribute set to 1.

Listing A.4: Delete Trigger

```
CREATE TRIGGER weatherdata_delete
ON weatherdata
AFTER DELETE
as
begin
  DECLARE @wid int, @vid int,
  @airportID INTEGER,
  @year INTEGER,
  @month INTEGER,
  @day INTEGER,
  @time INTEGER,
  @timezone INTEGER,
  @skycondition VARCHAR(50),
  @visibility FLOAT,
  @weathertype VARCHAR(50),
  @drybulbcelsius FLOAT,
  @wetbulbcelsius FLOAT,
  @dewpointcelsius FLOAT,
  @relativehumidity INTEGER,
  @windspeed INTEGER,
  @winddirection VARCHAR(20),
  @valueforwindcharacter INTEGER,
  @stationpressure FLOAT,
  @pressuretendency FLOAT,
  @pressurechange INTEGER,
  @sealevelpressure FLOAT,
  @recordtype VARCHAR(20),
  @hourlyprecip VARCHAR(20),
  @altimeter FLOAT

  SET @wid = (SELECT wid FROM deleted)
  SET @VID = (SELECT vid FROM deleted) + 1
  SET @airportID = (SELECT airportID FROM deleted)
  SET @year = (SELECT year FROM deleted)
  SET @month = (SELECT month FROM deleted)
  SET @day = (SELECT day FROM deleted)
  SET @time = (SELECT time FROM deleted)
  SET @timezone = (SELECT timezone FROM deleted)
  SET @skycondition = (SELECT skycondition FROM deleted)
  SET @visibility = (SELECT visibility FROM deleted)
  SET @weathertype = (SELECT weathertype FROM deleted)
  SET @drybulbcelsius = (SELECT drybulbcelsius FROM deleted)
  SET @wetbulbcelsius = (SELECT wetbulbcelsius FROM deleted)
  SET @dewpointcelsius = (SELECT dewpointcelsius FROM deleted)
  SET @relativehumidity = (SELECT relativehumidity FROM deleted)
  SET @windspeed = (SELECT windspeed FROM deleted)
  SET @winddirection = (SELECT winddirection FROM deleted)
  SET @valueforwindcharacter = (SELECT valueforwindcharacter FROM deleted)
  SET @stationpressure = (SELECT stationpressure FROM deleted)
  SET @pressuretendency = (SELECT pressuretendency FROM deleted)
  SET @pressurechange = (SELECT pressurechange FROM deleted)
  SET @sealevelpressure = (SELECT sealevelpressure FROM deleted)
  SET @recordtype = (SELECT recordtype FROM deleted)
  SET @hourlyprecip = (SELECT hourlyprecip FROM deleted)
  SET @altimeter = (SELECT altimeter FROM deleted)

  rollback transaction
  set identity_insert weatherdata on
  insert into weatherdata(wid, vid, deleted, airportID, year, month, day, time,
      timezone ,skycondition, visibility, weathertype ,drybulbcelsius, wetbulbcelsius,
       dewpointcelsius, relativehumidity, windspeed, winddirection,
      valueforwindcharacter, stationpressure, pressuretendency, pressurechange,
      sealevelpressure, recordtype, hourlyprecip, altimeter)
  values(@WID, @VID, 1, @airportID, @year, @month, @day, @time, @timezone ,
      @skycondition, @visibility, @weathertype ,@drybulbcelsius, @wetbulbcelsius,
      @dewpointcelsius, @relativehumidity, @windspeed, @winddirection,
      @valueforwindcharacter, @stationpressure, @pressuretendency, @pressurechange,
      @sealevelpressure, @recordtype, @hourlyprecip, @altimeter)
  set identity_insert weatherdata on
end
```

A.5 Process Parameter Script

Description: Python script used for the Process Parameter Module 4.3.2

Listing A.5: Process Parameter Script

```python
def azureml_main(dataframe1 = None, dataframe2 = None):
    import pandas as pd
    import hashlib
    from datetime import datetime
    import numpy as np

    # Extract parameter
    query = dataframe1['query'][0]
    sortquery = query

    # order query
    if 'order' not in query:
        sortquery = query+'␣order␣by␣wid␣DESC'

    # normalize query
    q1, q2 = sortquery.split('where')
    sortquery = q1 + "wd␣where␣vid␣=␣(select␣max(vid)␣from␣weatherdata␣xd␣where␣wd.wid␣=␣xd.wid)␣and␣deleted␣=␣0␣and␣"+q2

    qid = dataframe1['qid'][0]
    originalts = dataframe1['originalts'][0]

    # Check QID field
    # if id set, get old query
    if(qid > 0):
        olddf = dataframe2.loc[dataframe2['qid'] == qid]
        oldquery = olddf['qsort'].iloc[0]
        oldts = str(datetime.fromtimestamp(olddf['tmstmp'].iloc[0]))

        # if orignal ts should be used, create old query and send empty dataframe
        if(originalts):
            preq, postq1, postq2 = oldquery.split('where')
            newq = preq + "where␣tmstmp␣<␣'"+oldts+"'␣and␣"+postq1 + "where␣tmstmp␣<␣'"+oldts+"'␣and␣" +postq2
            print newq

            df = pd.DataFrame(columns=['query', 'qsort', 'qhash', 'rhash', 'tmstmp'])
            return df
        # just use old query with current time
        else:
            sortquery = oldquery

    # Create hash
    h = hashlib.md5()
    h.update(sortquery)
    m = h.hexdigest()

    # Create return attribute
    querydata = {"query":[query],"qsort":[sortquery],"qhash":[m]}

    insertframe = pd.DataFrame(querydata, columns=['query','qsort', 'qhash'])

    return insertframe
```

A.6 Query Metadata Script

Description: Python script used for the Query Metadata Module 4.3.3

Listing A.6: Query Metadata Script

```python
def azureml_main(dataframe1 = None, dataframe2 = None):
    import pandas as pd
    import hashlib

    #Check for iteration run
    if(dataframe2 is None):
        df = pd.DataFrame(columns=['query', 'qsort', 'qhash', 'rhash', 'tmstmp'])
        return df

    # Check for new Query
    if(len(dataframe2.index) == 0):
        return dataframe2

    # Create result hash
    columnames = dataframe1.columns.values
    keyval = dataframe1['wid'].values
    versionval = dataframe1['vid'].values

    h = hashlib.md5()
    h.update(str(columnames)+str(keyval)+str(versionval))
    m = h.hexdigest()

    # Create Query Timestamp
    time = pd.Timestamp('now')

    dataframe2['rhash'] = pd.Series(m, index=dataframe2.index)
    dataframe2['tmstmp'] = pd.Series(str(time), index=dataframe2.index)

    return dataframe2,
```

A.7 Hash Compare Script

Description: Python script used for the Hash Compare Module 4.3.4

Listing A.7: Hash Compare Script

```python
def azureml_main(dataframe1 = None, dataframe2 = None):
    import pandas as pd
    import md5

    # Check for new Query
    if(len(dataframe1.index) is 0):
        return dataframe1

    qhash = dataframe1['qhash'][0]
    rhash = dataframe1['rhash'][0]

    qhashseries = dataframe2['qhash']
    rhashseries = dataframe2['rhash']

    # Compare Hashes
    if(qhash in qhashseries.values and rhash in rhashseries.values):
        df = pd.DataFrame(columns=['query', 'qsort', 'qhash', 'rhash', 'tmstmp'])
        return df

    return dataframe1,
```

Bibliography

[1] Taverna components. http://www.taverna.org.uk/documentation/taverna-2-x/components/ Accessed June 2015.

[2] Óscar Corcho, Daniel Garijo Verdejo, K Belhajjame, Jun Zhao, Paolo Missier, David Newman, Raul Palma, Sean Bechhofer, Esteban García Cuesta, Jose Manuel Gomez-Perez, et al. Workflow-centric research objects: First class citizens in scholarly discourse. *Proceeding of SePublica 2012, pages 1-12*, 2012.

[3] The OPM provenance model (OPM). http://openprovenance.org/ Accessed June 2015.

[4] Paolo Missier, SatyaS. Sahoo, Jun Zhao, Carole Goble, and Amit Sheth. Janus: From workflows to semantic provenance and linked open data. In *Provenance and Annotation of Data and Processes*, Lecture Notes in Computer Science, pages 129–141. Springer Berlin Heidelberg, 2010.

[5] Daniel De Oliveira, Eduardo Ogasawara, Fernanda Baião, and Marta Mattoso. Scicumulus: A lightweight cloud middleware to explore many task computing paradigm in scientific workflows. In *Proccedings of the 3rd IEEE International Conference on Cloud Computing (CLOUD)*, pages 378–385. IEEE, 2010.

[6] Micah Altman and Mercè Crosas. The evolution of data citation: From principles to implementation. *IAssist Quarterly*, 63, 2013.

[7] Brad Severtson. Execute python machine learning scripts in azure machine learning studio. https://azure.microsoft.com/en-us/documentation/articles/machine-learning-execute-python-scripts/ Accessed July 2015.

[8] World Wide Web Consortium et al. Prov-overview: an overview of the prov family of documents. 2013. http://www.w3.org/TR/prov-overview/.

[9] Anthony JG Hey, Stewart Tansley, Kristin Michele Tolle, et al. *The fourth paradigm: data-intensive scientific discovery.* Microsoft Research Redmond, WA, 2009.

[10] Andrew McAfee, Erik Brynjolfsson, Thomas H Davenport, DJ Patil, and Dominic Barton. Big data. *The Management Revolution. Harvard Business Review*, 90(10):61–67, 2012.

[11] José Manuel Gómez-Pérez, Esteban García-Cuesta, Aleix Garrido, José Enrique Ruiz, Jun Zhao, and Graham Klyne. When history matters - assessing reliability for the reuse of scientific workflows. In *Proccedings of the Semantic Web–ISWC 2013*, pages 81–97. Springer, 2013.

[12] Jun Zhao, Jose Manuel Gomez-Perez, Khalid Belhajjame, Graham Klyne, Esteban Garcia-Cuesta, Austin Garrido, Kristina Hettne, Maree Roos, David De Roure, and Carole Goble. Why workflows break - understanding and combating decay in taverna workflows. In *8th IEEE International Conference on E-Science (e-Science)*, pages 1–9. IEEE, 2012.

[13] Kristina M Hettne, Katherine Wolstencroft, Khalid Belhajjame, Carole A Goble, Eleni Mina, Harish Dharuri, David De Roure, Lourdes Verdes-Montenegro, Julián Garrido, and Marco Roos. Best practices for workflow design: How to prevent workflow decay. In *Proceedings of the 5th international workshop on semantic web applications and tools for life sciences*, 2012.

[14] Stefan Proell. Dynamic data citation. `http://datacitation.eu/` Accessed June 2015.

[15] Boris Glavic and Klaus R Dittrich. Data provenance: A categorization of existing approaches. In *Datenbanksysteme in Business, Technologie und Web*, number 12, pages 227–241. Citeseer, 2007.

[16] Paolo Missier, Stian Soiland-Reyes, Stuart Owen, Wei Tan, Alexandra Nenadic, Ian Dunlop, Alan Williams, Tom Oinn, and Carole Goble. Taverna, reloaded. In *Scientific and Statistical Database Management*, Lecture Notes in Computer Science, pages 471–481. Springer Berlin Heidelberg, 2010.

[17] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew B 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.

[18] 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.

[19] Rudolf Mayer and Andreas Rauber. A quantitative study on the re-executability of publicly shared scientific workflows. In *e-Science (e-Science), 2015 IEEE 11th International Conference on*, pages 312–321. IEEE, 2015.

[20] David De Roure, Khalid Belhajjame, Paolo Missier, José Manuel Gómez-Pérez, Raúl Palma, José Enrique Ruiz, Kristina Hettne, Marco Roos, Graham Klyne, Carole Goble, et al. Towards the preservation of scientific workflows. In *Proceedings of the 8th International Conference on Preservation of Digital Objects (iPRES 2011)*, 2011.

[21] Joel T Dudley and Atul J Butte. In silico research in the era of cloud computing. *Nature biotechnology*, 28(11):1181–1185, 2010.

[22] Susan B Davidson and Juliana Freire. Provenance and scientific workflows: challenges and opportunities. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 1345–1350. ACM, 2008.

[23] Ewa Deelman, Dennis Gannon, Matthew Shields, and Ian Taylor. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009.

[24] Paul T Groth, Steve Munroe, Simon Miles, and Luc Moreau. Applying the provenance data model to a bioinformatics case. In *Proceedings of the High Performance Computing Workshop*, pages 250–264, 2006.

[25] Shawn Bowers, Timothy M McPhillips, and Bertram Ludäscher. Provenance in collection-oriented scientific workflows. *Concurrency and Computation: Practice and Experience*, 20(5):519–529, 2008.

[26] 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.

[27] 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.

[28] Rudolf Mayer, Stefan Proell, and Andreas Rauber. On the applicability of workflow management systems for the preservation of business processes. *Proceedings of the 9th International Conference on Digital Preservation (iPres 2012)*, page 110, 2012.

[29] Luc Moreau, Paul Groth, James Cheney, Timothy Lebo, and Simon Miles. The rationale of PROV. *Web Semantics: Science, Services and Agents on the World Wide Web*, 35, Part 4:235 – 257, 2015.

[30] Tom De Nies, Sara Magliacane, Ruben Verborgh, Sam Coppens, Paul T Groth, Erik Mannens, and Rik Van de Walle. Git2prov: Exposing version control system content as W3C PROV. In *Proceedings of the International Semantic Web Conference*, pages 125–128, 2013.

[31] *Getting Started with Kepler Provenance 2.4*, 10 April 2013. https://kepler-project.org/users/documentation.

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

[33] Sarah Callaghan. Preserving the integrity of the scientific record: data citation and linking. *Learned Publishing*, 27(5):15–24, 2014.

[34] Stefan Pröll and Andreas Rauber. Citable by design: A model for making data in dynamic environments citable. In *Proceedings of the 2nd International Conference on Data Management Technologies and Applications (DATA2013)*, pages 206–210, 2013.

[35] DataCite International Data Citation Metadata Working Group et al. Datacite metadata schema for the publication and citation of research data version 3.0.

[36] Mercè Crosas. The dataverse network®: an open-source application for sharing, discovering and preserving data. *D-lib Magazine*, 17(1):2, 2011.

[37] Data Citation Synthesis Group FORCE11. Joint declaration of data citation principles. *Martone M. (ed.) San Diego CA*, 2014. https://www.force11.org/datacitation.

[38] Annie PS Wong, Gregory C Johnson, and W Brechner Owens. Delayed-mode calibration of autonomous ctd profiling float salinity data by θ-s climatology*. *Journal of Atmospheric and Oceanic Technology*, 20(2):308–318, 2003.

[39] *Data Citation of Evolving Data*, 24 September 2015. https://rd-alliance.org/rda-wgdc-recommendations-vers-sep-24-2015.html, Accessed January 2016.

[40] Stefan Proll and Andreas Rauber. Scalable data citation in dynamic, large databases: Model and reference implementation. In *Proceedings of the IEEE International Conference on Big Data*, pages 307–312. IEEE, 2013.

[41] Stefan Pröll, Rudolf Mayer, and Andreas Rauber. Data access and reproducibility in privacy sensitive escience domains. In *Proceedings of the 11th IEEE International Conference on e-Science*, pages 255–258. IEEE, 2015.

[42] Belhajjame K B'Far R Cheney J Coppens S Cresswell S Gil Y Groth P Klyne G Moreau L, Missier P. Prov overview. Technical report, World Wide Web Consortium, 2012. http://www.w3.org/TR/prov-dm/ Accessed July 2015.

[43] Dieter van Uytvanck Stefan Pröll Andreas Rauber, Ari Asmi. Identification of reproducible subsets for data citation, sharing and re-use. Technical report, RDA Working Group on Data Citation (WGDC), 2016.