
# LOD-GEOSS: Basic Concept of the Distributed Database Architecture

LOD-GEOSS

InfAI Institute for Applied Informatics

Potsdam Institute for Climate Impact Research PIK

JÜLICH Forschungszentrum

Deutsches Zentrum für Luft- und Raumfahrt German Aerospace Center DLR

Universität Stuttgart IER Institut für Energiewirtschaft und rationelle Energieanwendung

REINER LEMOINE INSTITUT

Supported by:

Federal Ministry for Economic Affairs and Energy

on the basis of a decision by the German Bundestag

Project Number 03EI1005A

# Document properties

| | |
|---|---|
| Title | Milestone 1:<br>LOD-GEOSS: Basic Concept of the Distributed Database Architecture |
| Subject | |
| Institutes | German Aerospace Center (DLR),<br>Forschungszentrum Jülich,<br>Reiner Lemoine Institut |
| Compiled by | Carsten Hoyer-Klick |
| Participants | Johannes Frey, Martin Stoffers, Sophie Jentzsch, Tobias Hecking, Patrick Kuckertz, Leander Kotzur, Detlef Stolten, Christoph Muschner, Ludwig Hülk |
| Checked by | |
| Released by | |
| Date | Oct, 12th, 2021 |
| Version | 0.9.2 – Second public draft |
| File Path | DOI? |
| License | Licensed under CC BY 4.0<br>Please cite as:<br>Hoyer-Klick, C., Frey, J., Stoffers, M., Jentzsch, S., Hecking T., Kuckertz, P., Kotzur, L., Stolten, D., Muschner, C., Hülk, L. (2021): LOD-GEOSS: LOD-GEOSS: Basic Concept of the Distributed Database Architecture.<br>© DLR © FZJ © C © Reiner Lemoine Institut \| CC BY 4.0 |

# Table of contents

# 1. Introduction

## 1.1. Challenges

Energy systems analysis is driven by models and data. The goal is to analyze existing and future energy systems based on numerical models. Modelling in energy systems analysis needs data from a variety of different sources from different domains:

- Meteorological time series for renewable power generation
- Electricity demand for different sectors
- Technology data
- Socio economic data
- Geographical land use data

These data usually come from different sources and one of the first steps in energy systems analysis is data research and processing. The input data is compiled and evaluated. In practice, each source has different access mechanisms and additional information. Each dataset uses its own definitions. This type of data research is very often repeated as the results of this background work usually not shared openly.
One of the main ideas of the LOD-GEOSS project is to ease data discovery and sharing and to reduce the amount of this rather unproductive research and to increase reproducibility and collaboration.
The project therefore contributes to the implementation of the "Fair Data Principles" to make data **f**indable, **a**ccessible, **i**nteroperable and **r**eusable.

## 1.2. Learning from Earth Observation

Similar problems arose in earth observation about two decades ago. The answer was the development of a distributed data infrastructure for earth observation data, the Global Earth Observation Systems of Systems [GEOSS]. Data could be found through data catalogs, where providers registered their metadata and ways of data access could be discovered though standardized interface descriptions as WSDL (Web Service Description Language) [WSDL].
As the data stays within the hosting institutions, these institutions stay responsible hosting and maintenance, which eases legal questions and updates of the data. The idea of this current work is the development of a networked database concept based on the ideas of GEOSS and Linked Open Data (LOD) and the semantic web for input and output data of models in energy systems analysis.

## 1.3. The Databus

We are developing a distributed database architecture and use it in energy systems analysis. The infrastructure is based on the Databus[1], mainly developed by DBpedia. All data providers (this includes users which performed a series of transformation of existing data, but also publishers in a

[1] https://databus.dbpedia.org/

conventional sense) can publish standardized metadata on the Databus (which is based on a metadata catalog in its core). While data providers maintain sovereignty of hosting the data itself, the Databus can be utilized to announce, discover, retrieve, and contribute (to) various data sources in a standardized and interoperable way. The vision of the LOD-GEOSS project is to create an ecosystem between data life cycles of the energy domain itself, the Energy Databus.

# 2. Basic technologies

## 2.1. The Databus metadata catalog

The Databus is a user and community extendable platform built on top of a homogenous domain-independent metadata catalog It is designed to publish metadata in a structured, sustainable and automatized fashion. Basic guiding principle for the development of the Databus is to achieve a high degree of interoperability and automatization between:

- Access to data
- Discovery and analysis of data
- Recombination, transformation, and reuse of data
- Contribution of data modification like add-ins and addon (e.g. summary, error report, translation)
- Flexible import of data and deployment of software and services along with data such that users and communities can use the platform to coordinate and manage their data life cycles to establish a platform economy and benefit from network effects of it. [Databus]

### 2.1.1. Databus approach

The Databus is a platform or hub to capture invested effort by data scientists who needed better data (quality - fitness for use) to use and contribute improvements back to the data source and others. It allows exchanging, curating, and accessing data between multiple stakeholders. Any data entering the bus will be versioned, cleaned, mapped, linked and its licenses and provenance tracked. Hosting in multiple formats is provided to access the data either as dump download or as API.
In computer architecture, a *bus* is a communication system that transfers data between components inside a computer, or between computers. The DBpedia Databus elevates this principle to the Web in the following manner:

- Databus is **intended for data analysts**, who can mirror the original data and their extracted or cleaned derivatives to facilitate easier reuse, deployment, and further derivatives. Original data publishers can have a benefit by re-integrating derived value like validations and additions into their provided work. The Databus becomes a community and network.
- **Data providers** can also create a space on the Databus and add links to accessible files on their servers. It is made by posting a metadata file (*dataid.ttl*) with a parameter (*dcat:downloadURL*) to the Databus. The turtle file [TURTLE] is based on the Data Catalog Vocabulary [DCAT]. Data providers remain full authority over storage on their servers. This is useful for open data with open licenses and direct access and at the same time the coordination of internal projects with sensitive data.
- **Data user** can access the metadata search using a Query Language for RDF [SPARQL] for discovery and network coordination.

The *DataID* example shown below shows metadata for an example *dataid.ttl* wind power plant list of the German market core data register (MaStR). The hyperlinks are marked as bold text, data provenance is described through provid stable identifiers to reference the (abstract entity of the) dataset, the specific version of the dataset and the data file itself.

```
## Metadata for DataID (Meta-metadata)
<http://dbpedia-mappings.tib.eu/databus-repo/jj-author/mastr/bnetza-mastr/01.04.01/dataid.ttl>
       a                    dataid:DataId ;
       rdfs:comment         "Metadata created by the DBpedia Databus Maven Plugin:
https://github.com/dbpedia/databus-maven-plugin" ;
       dct:hasVersion       "1.3-SNAPSHOT" ;
       dct:issued           "2020-03-02T14:43:24Z"^^xsd:dateTime ;
       dct:license          <http://purl.oclc.org/NET/rdflicense/cc-zero1.0> ;
       dct:publisher        <https://jj-author.github.io/webid.ttl#this> ;
       dct:title            "DataID metadata for mastr/bnetza-mastr"@en .

# Metadata for Energy Core Data register dataset
<http://dbpedia-mappings.tib.eu/databus-repo/jj-author/mastr/bnetza-mastr/01.04.01/dataid.ttl#Dataset>
       a                           dataid:Dataset ;
       dct:title                   "MarktStammDatenRegister (MaStR) Processing"@en ;
       dataid:account              databus:jj-author ;
       dataid:artifact             <https://databus.dbpedia.org/jj-author/mastr/bnetza-mastr> ;
       dataid:group                <https://databus.dbpedia.org/jj-author/mastr> ;
       dataid:groupdocu            "# RLI demo for their MaStr extraction" ;
       dataid:version              <https://databus.dbpedia.org/jj-author/mastr/bnetza-mastr/01.04.01> ;
       dataid-debug:documentationLocation
              <https://raw.githubusercontent.com/kurzum/databus-poms/master/kurzum/mastr/bnetza-
mastr/bnetza-mastr.md> ;
       dataid-debug:feedbackChannel  <https://forum.dbpedia.org/u/kurzum> ;
       dataid-debug:issueTracker     <https://github.com/kurzum/databus-poms/issues> ;
       dct:description             "...";
       dct:hasVersion              "01.04.01" ;
       dct:issued                  "2020-03-02T14:43:24Z"^^xsd:dateTime ;
       dct:license                 <https://www.govdata.de/dl-de/by-2-0> ;
       dct:publisher               <https://jj-author.github.io/webid.ttl#this> ;
# 3 files belong to dataset
       dcat:distribution           <http://dbpedia-mappings.tib.eu/databus-repo/jj-author/mastr/bnetza-
mastr/01.04.01/dataid.ttl#bnetza-mastr_rli_type=hydro.nt.gz> ,
                                   <http://dbpedia-mappings.tib.eu/databus-repo/jj-author/mastr/bnetza-
mastr/01.04.01/dataid.ttl#bnetza-mastr_rli_type=wind.nt.gz> ,
                                   <http://dbpedia-mappings.tib.eu/databus-repo/jj-author/mastr/bnetza-
mastr/01.04.01/dataid.ttl#bnetza-mastr_rli_type=biomass.nt.gz> .

# Metadata for one of the files containing wind energy data
<http://dbpedia-mappings.tib.eu/databus-repo/jj-author/mastr/bnetza-mastr/01.04.01/dataid.ttl#bnetza-
mastr_rli_type=wind.nt.gz>
       a                           dataid:SingleFile ;
       dataid:contentVariant       "rli" , "wind" ;
       dataid:file                 <https://databus.dbpedia.org/jj-author/mastr/bnetza-
mastr/01.04.01/bnetza-mastr_rli_type=wind.nt.gz> ;
       dataid:formatExtension      "nt" ;
       dct:modified                "2020-03-02T14:36:54Z"^^xsd:dateTime ;
       dcat:downloadURL            <http://dbpedia-mappings.tib.eu/databus-repo/jj-author/mastr/bnetza-
mastr/01.04.01/bnetza-mastr_rli_type=wind.nt.gz> ;
       dataid:isDistributionOf     <http://dbpedia-mappings.tib.eu/databus-repo/jj-author/mastr/bnetza-
mastr/01.04.01/dataid.ttl#Dataset> ;
       dcat:byteSize               "3189271"^^xsd:decimal ;
       dataid:sha256sum            "05af23c7102c89b7fbf9806be331eb47203de41ba7fd3d4b90caf052e3285596" ;
       dataid:signature            "ew4Ccw2MO4CnBXKSt6nOKJLsF6a0B0zU3SB5GMwrhHFc919z3C ..." .
```

Any user can query the Databus with a semantic query language for databases called SPARQL to identify their own datasets alongside datasets of other consumers and download them via retrieving the *dcat:downloadURL* or the Databus Client into their local machines and applications. The subsequent query fetches the download URLs of all CSV files from the MaStR dataset in version '01.04.00'

```
PREFIX dataid: <http://dataid.dbpedia.org/ns/core#>
PREFIX dct:    <http://purl.org/dc/terms/>
PREFIX dcat:   <http://www.w3.org/ns/dcat#>

SELECT DISTINCT ?file WHERE {

 ?dataset dcat:distribution ?distribution .
 ?distribution dcat:downloadURL ?file .
 ?dataset dataid:artifact <https://databus.dbpedia.org/jj-author/mastr/bnetza-mastr> .
 ?distribution dataid:hasVersion '01.04.00'^^<http://www.w3.org/2001/XMLSchema#string>
 ?distribution <http://dataid.dbpedia.org/ns/core#formatExtension> 'csv'^^<http://www.w3.org/2001/XMLSchema#string> .
}
```

A tutorial to learn the syntax of SPARQL -- which is similar to SQL-- can be found online.[2]

### 2.1.2. Data improvements, extensions, and contributions

If users either spot quality issues (such as parse errors, wrong data, missing coverage) or would like extensions, they are able to add value to the data in the following manner:

- **Derivatives:** Managing data quality is pareto-efficient, 20% of the initial effort make up 80% of the result. Then effort increases steeply, in fact so steep that the original publisher will struggle to maintain the data. Commercial data providers will weigh effort against income and therefore triage what to improve. Open data providers without an income model normally do not have the resources to accommodate all user needs and focus on own needs. The Databus allows users to re-publish patched data or community extensions, which can be picked up by publishers from the bus to complement their data. These "derive" processes are fully automatable and keep the original source in the parameter *prov:wasDerivedFrom*, therefore all processing can be traced back to the original data set and all modifications which have been done since then. New versions can be detected by querying the Databus, and processors can run and publish derivatives in third-party storage. Different from pipelines, derive operations form a data network where effort and value are discoverable and can flow freely through the network up to the sources in a trickle-up manner.
- **Mods** are routines for statistical and semantic analysis, continuous integration (CI) testing and enrichment, or any other tasks. Mods are an effective way to provide value for the whole Databus. Only one developer and one server (not necessarily provided by the developer) is required to add additional (meta)information to the datasets, producing a consistent layer of annotations to all data. In other projects this was done by user tagging, which quickly becomes confusing (and often messy). Mods can also be completely selfish. For the DBpedia knowledge graph itself, DBpedia will implement a statistic collection mod, allowing us to better understand how the knowledge graph evolves. Others need to deploy and run it for themselves. Consumers can write mods that check new data before they ingest it to avoid application breaks on data updates.
  There is an implemented demo mod[3] that measures uptime of all available links of all storage. It is a *prov:Activity* (see chapter 2.3 below) and adds "onlinerate" and "weeklyonlinerate", and links to detailed reports to the SPARQL-accessible database.

[2] https://www.stardog.com/tutorials/sparql/
[3] https://github.com/dbpedia/databus-mods/

- **User feedback:** Alongside the metadata, users can include feedback links to forums, issue-trackers, or directly to the code (e.g. on GitHub) where other users can specify the issue, fix the software, or provide test cases. This feedback is a valuable input for initial derivative, as it can improve data quality and speed up innovation. It is the technical basis for building a user community.

## 2.2. Overlay systems with Databus mods

The Databus Mod (Microservice) Architecture is designed to allow a flexible and community-driven augmentation of the Databus itself or to build overlay systems and services on top of a set of Mods forming a specialized "sub-Databus". An example how the functionalities of the Databus can be extended in a very generic, but useful way, can be given using the MIME-Type and Line-Count Mod. Detecting the file type and compression format is a challenging task. Using the MIME type mod this can be computed in a unified and deterministic way for all files registered on the Databus. Error fixes and improvements can be automatically applied without the need to perform re-registering of the files (metadata update) by the publishers / "uploaders" since the information is detached from the original DataID metadata core required to register files. However, the Mod-Server for a specific mod uses the Databus file identifiers to provide links from the files to the mod result e.g. the number of lines for the MaStR wind csv file (1). Using SPARQL federated querying (Databus + Mod Server SPARQL endpoints) Databus metadata and Mod data can be combined in one query. Thus, it would be possible to query for files of a textual MIME type having a specific number of lines by combining mod results from the MIMEType-Mod and LineCountMod. The Mod Server (provided as reference implementation) provides a SPARQL endpoint (5) with data and metadata from the mods (4) and takes care of monitoring the Databus for file updates (2) and handles scheduling and file management for Mod workers (3). Mod workers provide simple REST Interfaces to be controlled by the Mod Server. They can be written in any programming language and produce any kind of results that can be stored in a file.

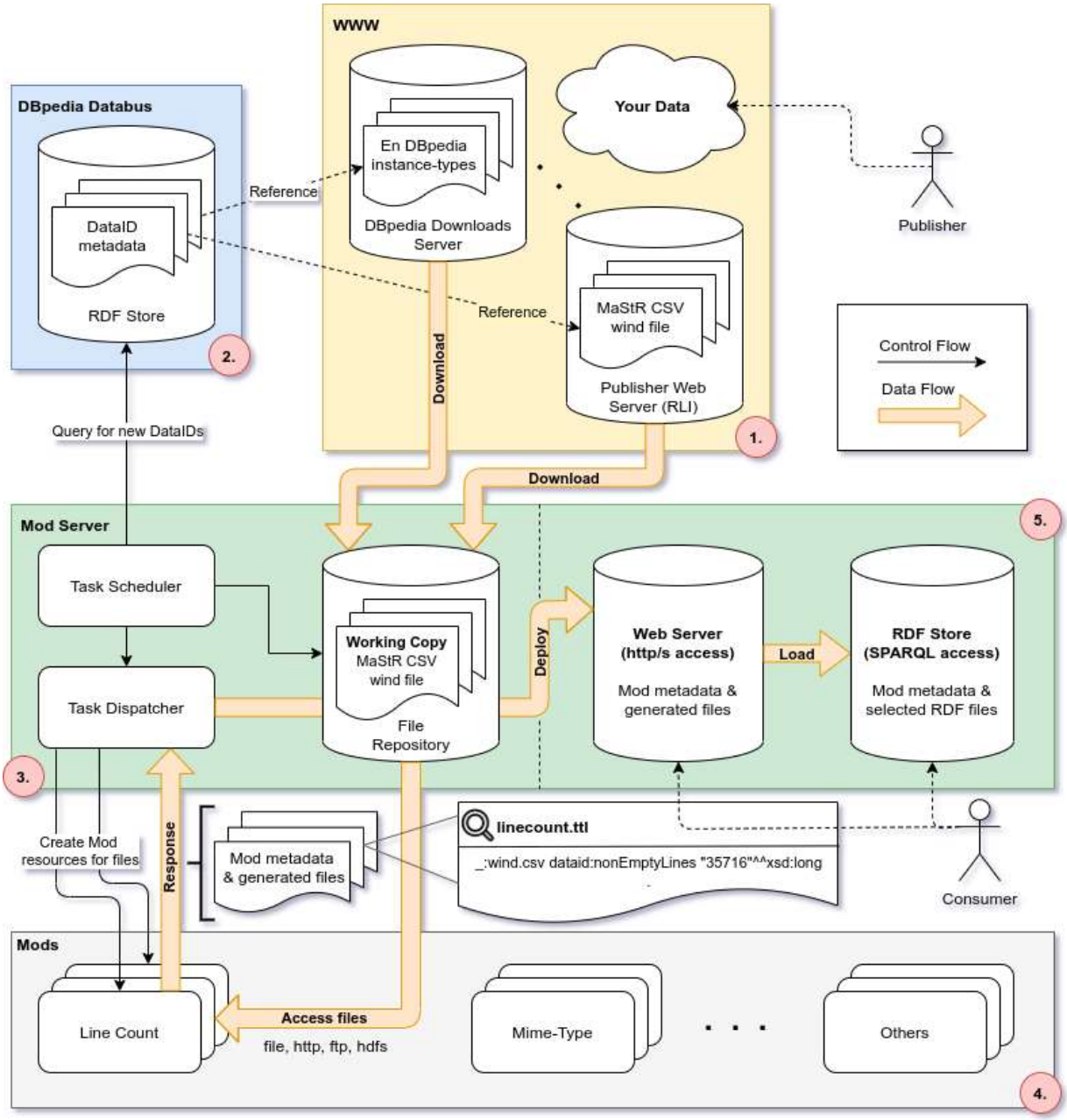


Figure 1: Mod processing with the Databus

## 2.3. Data provenance

In typical analysis workflows, datasets and models from multiple sources are constantly transformed, extended, and merged, which eventually yields a growing number of new entities. In order to maintain reliability and increase trustworthiness, it is inevitable that single processing steps, the origin of data and involved actors can be traced back. This kind of information is referred to as *provenance.* In order to collect provenance information in a consistent way, standards and technologies have been developed. They facilitate comprehensive post-hoc analysis

of workflows and backtracking of data products. The following section introduces the established provenance standard and its application to present use cases.

### 2.3.1. The PROV metadata standard

PROV[4] is a W3C standard defining a set of concepts and relationships to describe provenance of data, which is also adopted in the LOD-GEOSS data infrastructure. The general structure of the PROV ontology (PROV-O) is depicted below. A PROV document consists of a set of relation triples between PROV-O elements with additional meta information.

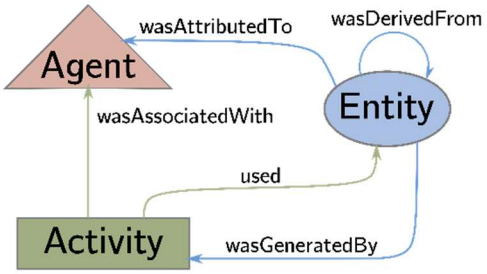


Figure 2: PROV Ontology

Elements can be either entities, activities or agents, connected by the relationships "wasAttributedTo" (entity to agent), "wasDerivedFrom" (entity to entity), "wasAssociatedWith" (activity to agent), "wasgeneratedBy" (entity to activity), or "used" (activity to entity).

### 2.3.2. ProvStore

ProvStore[5] is a publicly available service for storing and managing provenance information. It is built upon the W3C provenance standard and employs the prov python package. With ProvStore provenance data can be stored in documents and subsequently be transformed, visualized and downloaded easily in multiple ways. Further, it can be published, shared and accessed straightforwardly via RESTful API.
The provenance document can either be made publicly available or shared within specific users with different access roles. Data can be displayed and downloaded in JSON, SVG, Trig, Turtle or XML format. Provenance descriptions are displayed in PROV-N by default and come with some basic statistics regarding quantity of nodes, edges, entities, etc.
There are different visualization layouts available, as Hive Plots, Wheel Plots or Gantt Charts, or more simplified views, as "Data Flow", "Process Flow" and "Responsibility", which focus on the comprehensive presentation of flow of information, conducted process and assigned responsibilities, respectively.

[4] https://www.w3.org/TR/prov-overview/
[5] https://openprovenance.org/store/

# 3. Energy Databus – Usage of the Databus in energy system analysis community to form a decentralized data(base) architecture

## 3.1. An example data pipeline on the Energy Databus

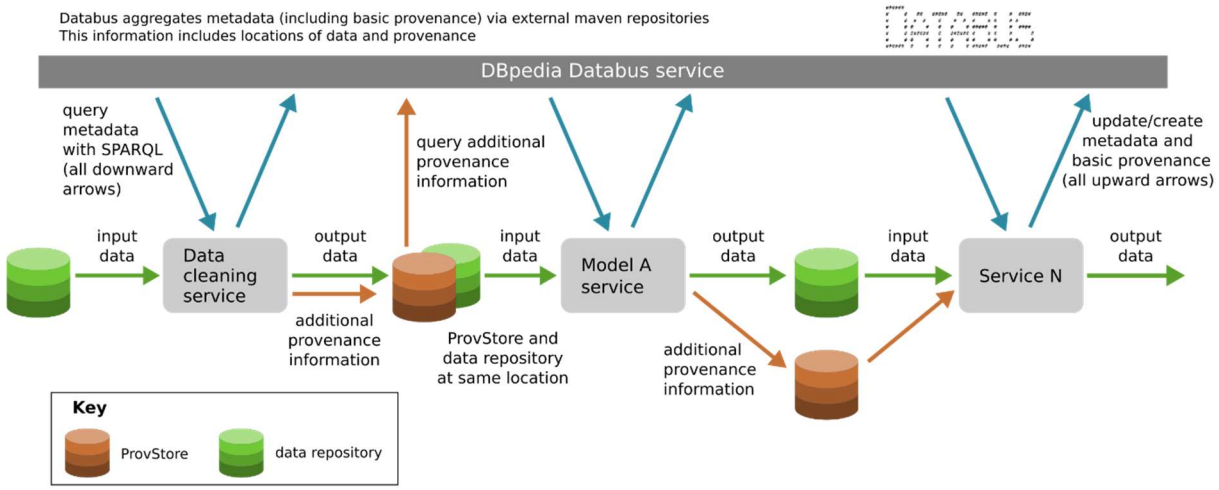


Figure 3: Example Pipeline of data processing with the data bus

In Figure 3 an example data pipeline is shown which will be used as running example to sketch how the Energy Databus can be used to support the use cases and tackle some of the challenges. The example workflow consists of 3 services and phases starting on the left side with a data preparation service. The cleaned/transformed data is then used in an energy system model. The simulation produces a new dataset which can be used by another service. The general concept is that the Databus is leveraged to collect and handle all metadata (catalog) for the individual services. In every service the following steps are performed:

- A service can query information about available data from the Databus. It contains:
  - Metadata about the dataset, including basic provenance (e.g. author, license, release date, etc.)
  - Download/Access Location of the dataset
  - Location of additional provenance (Fine-grained provenance information for different models on ProvStore, see Section 2.3)
- A service retrieves the metadata via the Databus and accesses the input data
- A service generates a new dataset based on the input (optionally recording detailed provenance) and standardized metadata description (Databus DataID vocabulary)
- The dataset and metadata are hosted in a user-specific/-controlled webspace
- The Databus is notified about the location of the new dataset and its metadata and includes (replicates) the metadata into the centralized catalog in the Databus user namespace/account

## 3.2. Use Case 1: Sharing and publishing of static data

**Use case:** A scientist wants to store static data (dump files) in a way that it becomes easily reusable for herself or himself and fellow scientists.
**Challenges:** There are many uncertainties associated with this supposedly simple task. In which format should the data be released, what metadata is necessary, how to realize stable referenceability, perform update notifications, and how to structure the data to be reusable in a flexible way while being not too complex to understand. These uncertainties in combination with the little amount of time a scientist has to deal with them, leads to temporary solutions. The scientist often lacks concrete action examples to make it better. The consumers are left to deal with the issues and resulting heterogeneity.
**Application:** DBpedia Databus does not host the files itself, these are hosted on the servers (i.e. storage) of its users. Databus consumers upload their files on their own server. Next to their files, they generate (e.g. with the help of an upload client) the `dataid.ttl` file in RDF-Turtle containing the metadata. The metadata is then registered/announced at the DBpedia Databus and will show up on the Databus user profile (via a Webinterface and the SPARQL backend). The Databus DataID metadata requires a minimal set of metainformation from the publisher in order to support data/service interoperability (FAIR data principles). The Databus system takes care that basic metadata is provide when registering a file.
Databus capable tools (like the DBpedia Databus Client[6]) can reduce the effort for both publishers and consumers by allowing flexible on-the-fly conversion between file formats. The Databus assigns for every data asset (file) a stable referenceable identifier. "Update notifications" can be performed in a unified way by registering a new version of the dataset. The Databus offers a tradeoff between fixed and unified but still flexible organization scheme to structure data.

## 3.3. Use Case 2: Recording data provenance and attach descriptive metadata

**Use case:** A scientist wants to describe his data with metadata, to make it reusable for himself and fellow scientists. He thereby is confronted with different file formats, e.g. Shape Files or Excel files, and internal and external databases and repositories. His datasets are either of his own making or come from one or more different sources.
A scientist wants to version her scenarios or individual datasets to allow the evolvement of data without losing identifiability.
**Challenges:** It is often unclear, where to put the metadata. Does it go in the files or database tables themselves or will it go in new files, e.g. README files, or tables? How to secure, that data and metadata is not separated or confused?
How should the data be modeled, what makes a dataset? Few large datasets lead to few version numbers which are updated quite often. Many small datasets lead to many version numbers which are updated quite seldom. Which is the better practice?

[6] https://github.com/dbpedia/databus-client

**Application:** In order to keep track of the origin of datasets and transformations of them provenance techniques described in Section 2.3 will be adopted to the case of energy system analysis.
In the architecture diagram in Figure 3 the red arrows indicate the flow of provenance data. Every service that consumes data from and stores data in the Databus records metadata about the data lineage by additionally connecting to ProvStore (Section 2.3.2) as a service for storing and managing provenance information. While *Derivatives* and *Mods* (Section 2.1.3) already collect high-level provenance about the origin of datasets and activities that generated them, in certain cases it is desirable to trace the workflow on a more fine-grained level. For example, a script applying an agent-based energy system model usually consists of many different calculations and data transformations. For the sake of reproducibility/comparability of data analyses, and to be able to identify reasons for erroneous data, data processing must be recorded step-by-step. This can be done by annotating processing scripts with provenance information that is then automatically stored in ProvStore. An example for annotating Python scripts with additional provenance information can be found here[7].
Versioning of datasets is a strict requirement of the Databus. Once a version of a dataset was registered it is supposed to be not changed anymore. The Databus offers stable identifiers for the abstract identity but also for individual version. Issued time stamps in the metadata allow a time-oriented view on the evolution of datasets.

### 3.3.1 Structure of provenance data

The PROV W3C standard described in Section 2.2.1 defines a basic skeleton for documents with provenance information. The LOD-GEOSS data infrastructure adapts and expands this by mapping specific entities, actors, and activities that are involved in energy system analysis to the PROV-O ontology.

### 3.3.2 Storage of provenance data in ProvStore

The ProvStore infrastructure (Section 2.3.2) will be used for storing and managing provenance data in LOD-GEOSS as it already provides many functionalities for storing data in different PROV formats and analysis. Together with the Databus it is part of the LOD-GEOSS federated database infrastructure. Services that process datasets communicate with ProvStore using its RESTful API. One PROV document in the ProvStore can be associated to one or more datasets on the Databus linked through the D*ataID* property of entities. Note that a document can also contain entities that do refer to Databus entries, for example, intermediate results of particular processing activities that are relevant to document an analysis workflow but will not be stored persistently. To this end, libraries for communication with ProvStore will be developed for different types of processing services.

### 3.3.3 Provenance graph example

[7] https://pypi.org/project/provenance/

Energy system analysis in the context of the LOD-GEOSS infrastructure can be broken down to 4 main activities for which provenance data is recorded:

1. ReadDB: Reading of data from a resource listed on the the Databus.
2. WriteDB: Publishing data on the Databus.
3. DataTransformation: An offline data transformation that leads to a modified dataset.
4. ModelSolving: Application of a solver to an energy system model.

Figure 4 presents a generic provenance graph as it could be generated automatically while recording information about the aforementioned activities. Ideally, the graph can be read from bottom to top. In the present example, Open Energy Modelling Framework (oemof) scripts are applied on data from the Databus and finally handed back to it. Two scientists, scientist A and scientist B, from the same organization perform different actions on the data and the model. First, oedb (open energy data base) data is retrieved from a Databus reference, which is recorded as a ***readDB*** activity associated to scientist B and the Databus as involved agents. Any kind of data modification such as cleaning is captured as a ***data transformation*** activity, which creates an intermediate dataset. This dataset is in turn used for configuring (***model configuration***) and solving (***model solving***) an open energy system model. Finally, the results are stored and a reference is published on the Databus in a ***writeDB*** activity. The resulting provenance graph depicted below allows to keep track of all these activities, involved users, and produced data entities. Interim results are displayed as individual entities, which enhances versioning control. In contrast to utilized data and models, autonomously performing scripts, as e.g. the CBC (Coin-or branch and cut) solver, are not classified as entity but as (software) agents. Each entity and each agent comes with a set of individual characteristics, like parameters, IDs, timestamps or resources. Provenance Graphs are highly adaptable and thus constitute a powerful tool to meet different requirements. The focus of analysis can be set differently, e.g. on the involved persons, characteristics of data, or chronological proceeding. The list of captured metainformation is arbitrarily expandable, as well as the granularity of displayed steps. Therefore, a reasonable implementation of storing and visualizing provenance information is a great benefit and can help to meet the challenges in multiple use cases (see Section 3.1.).

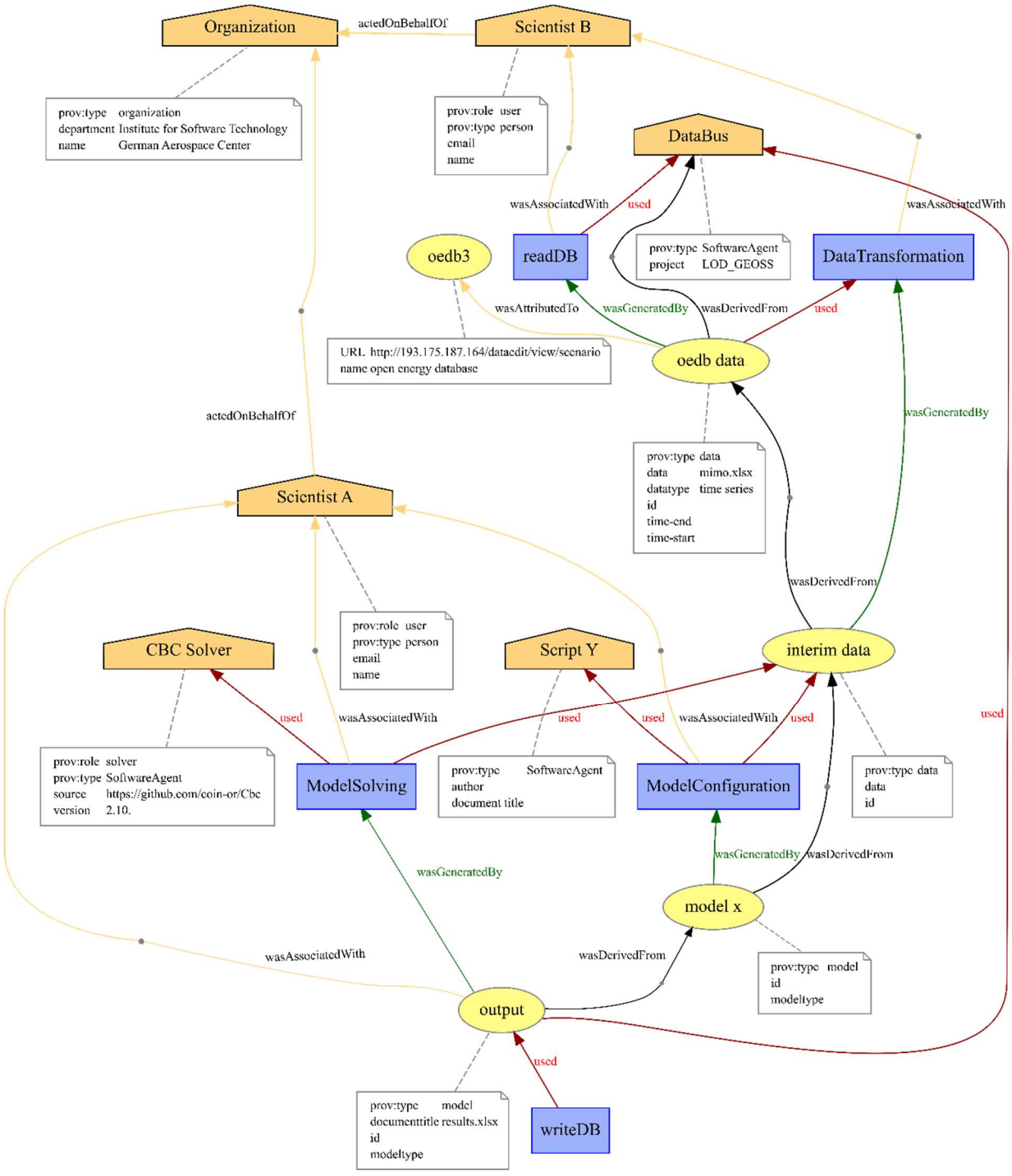


Figure 4: A generic provenance graph as it could be generated automatically while recording information about the aforementioned activities.

## 3.4. Use Case 3: Finding and citing sources for static data

**Use case:** Like any data which is used as input for scientific analyses, a scientist wants to find as many (high quality) sources of information as she can. One wants to build up a well-documented and well referenced collection of values, enabling her/him to determine the robustness of the

data and derive forecasts and generally secure the quality of the chosen input data. Once the data is collected, it is not being changed and rather serves as static basis for further statistical processing.

**Challenges:** The documentation of data sources goes often hand in hand with literature reviews and reference management. Thereby it is often not clear how to best attach literature information to the data. The data and its formats are used by different scientists and is to be used in publications in a homogeneous and most efficient way.

**Application:** A plethora of options exists to discover (new) datasets:

- Databus basic metadata freetext search (overall search, search per user, per artifact) based on a (lucene) index on metadata
- Fine grained search with SPARQL on metadata
- Exploring user Spaces and Collections via Databus Website
- Follow Basic file Databus file provenance chain to identify datasets which are used as input for other datasets or collections
- Use deep provenance data (e.g. in ProvStore) and combine via federated querying with Databus metadata – allows to search e.g. for outputs of a specific model
- Use another "overlay" search system based on a specific Mod – a Mod providing annotations for the content of the dataset using the Open Energy Ontology could enable an (energy domain) community tailored search

The versioned Databus identifiers realize a stable way to cite a specific version of a dataset/file independent of its current file location. Moreover, Databus collections allow a self-documenting way to report but also reuse a set/catalog of Databus identifiers (e.g. to reproduce calculations for a paper) by using a single collection ID.

## 3.5. Use Case 4: Integration of data updates

**Use case:** Scientists want to always use the latest data, e.g. the newest statistics of the expected population growth published by a federal ministry. They therefore regularly scan the respective websites and converts and integrates the data in her data repository.

**Challenges:** First of all, it must be avoided that this work has to be repeated by every scientist. In addition, it would be desirable if this work were no longer performed by scientists, but by the data producer. This could be achieved through an automated connection between the data infrastructure of the data producer and an overarching data catalog.

**Application:** Dynamic Databus collections allow to import the latest data from various users based on the special "Latest version property". The user can create such a collection representing their information need and the collection will automatically update whenever a newer of the imported data exists. This allows that an energy model can be run automatically based on the latest data available on the Databus.

## 3.6. Use Case 5: Transformation and modification of data

**Use case:** Scientist want to reuse an existing multi-dimensional dataset, e.g. for comparing, validating or supplementing their own data, but need it with (slightly) different resolutions. They e.g. need 15 minutes instead of one-hour intervals, regions instead of countries or the energy consumption by sector instead of a cumulated value. They want to use especially this dataset, since this is already the most adequate one. They can judge the differences as being minor and ignore them, while stating that they did in their publication. Or they can utilize conversion functionality, thereby manipulating their own or the reference dataset.
**Challenges:** The more dimensions a dynamic dataset contains, the more likely it is, that the respective resolutions do not fit the needs. At the same time the complexity and size of a dataset grows with the number of its dimensions, reinforcing the scientist's desire to reuse this dataset instead of having to generate it herself/himself. Usually, when a dataset contains aggregated data, the underlying more detailed data is not referenced and/or available.
**Application:** Users can derive converted datasets and publish them for reproducibility and reuse on the Databus. For very common transformations mappings can be created to perform on-the-fly conversion during consuming the data from the Databus on the consumer side. Additionally, a third party could write Databus mods, realizing transformation or shaping of data in a general fashion (e.g. convert all units of measurement into a specific resolution). Moreover, services and application descriptions (metadata) can be linked to specific artifacts, therefore e.g. announcing (conversion) APIs for a dataset.

## 3.7. Use Case 6: Handling of scenarios

**Use case:** Scientists want to define a certain scenario they are investigating and give it a unique name. In their perspective a scenario is purely a data collection containing all the data they used for the calculations. They are thereby able to quickly and explicitly reference their input, thereby raising the transparency regarding used assumptions and level of details, etc. For the scientist the usage of identifiable scenarios makes the data more tangible, versionable and exchangeable.
**Challenges:** Usually, a scenario is a list of references pointing to the respective datasets. But it is not always clear, how to persistently handle those references. And sometimes copies of the original datasets are preferred over a mere reference because of potentially ephemeral external sources
**Application:** The Databus uses and assigns PIDs (persistent identifiers) for persistent handling of references. The PIDs will persist even if the original source data may vanish. The Databus can handle multiple and distributed copies of the same dataset, giving users the possibility to upload their contents to multiple repositories increasing the robustness of accessing their data (more than one URL for one URI / PID). The Databus allows the handling of data collections (in this context a data collection shall be equal to a scenario). In this case, a data collection is a list of PIDs referencing other datasets. A data collection can have its own PID.

References:


- GEOSS - https://www.earthobservations.org/geoss.php
- WSDL - https://www.w3.org/TR/wsdl20/
- Databus - https://downloads.dbpedia.org/repo/lts/publication/strategy/2019.09.09/strategy_databus_initiative.pdf
- TURTLE - https://www.w3.org/TeamSubmission/turtle/
- DCAT - https://www.w3.org/TR/vocab-dcat-2/
- SPARQL - https://www.w3.org/TR/rdf-sparql-query/
- DataID - https://link.springer.com/chapter/10.1007/978-3-319-49157-8_28