Metadata- and Ontology-based Data Provenance Collection in an Aircraft Digital Twin

Jos Lehmann[1,*†], Martin Lanz[1,†]

[1] German Aerospace Center, Institute for AI-Safety and Security, Safety-Critical Data Infrastructure, Germany

Abstract

This paper lays out initial analytical results of ongoing research and development activities about data provenance collection in the digital twin of an aircraft. A case-study-based approach to provenance collection is being applied to a data and metadata lake of an aircraft digital twin, to assess the increase in heterogenous data manageability through metadata integration. The research context and motivation are described in the introduction, including provenance in an aircraft digital twin. Next, a problem definition is formulated and approaches discussed based on relevant literature. The working architecture of a demonstrator is then presented (Data Provenance and Metadata Demonstrator). In its current version this demonstrator is implemented as a python/SPARQL/owlready script for metadata structuring, coupled with an example ontology for provenance. It is based on a pipeline from metadata to provenance (Data Collection, Metadata Extraction, Provenance Collection, Provenance Analytics). Most implementation activities have so far focused on the first three phases, with the interplay between the python script and the ontology occurring in Provenance Collection. This has allowed to identify, test and implement some of the technology for automating knowledge graph creation for provenance. This prototype is being further expanded to cope with robustness issues caused by the nesting of metadata, tackled through recursion, and by mismatches between incoming metadata and the demonstrator's knowledge base, tackled by calls to an LLM.

The final section draws some conclusions and describes potential future work including: further grounding of the problem definition with respect to digital twins; case studies for CAD models; analytics; visualization.

Keywords

Provenance, LLM Supporting Knowledge Graph Generation, Digital Twin, Research Artifacts

1. Introduction

This paper describes ongoing research and development activities about data provenance collection in the digital twin (DT) of an aircraft (ADT)[1]. A case-study-based approach to provenance collection is being applied to a data and metadata lake of an ADT, in order to assess the increase in heterogenous data manageability through metadata integration.

Recent years have seen a focus on defining and developing DTs. The rigorous definition of this type of software infrastructure and of its variants (e.g. Digital Shadow, Cognitive Digital Twin, etc.) is still the subject of debate [1]. However, most definitions include three main elements: a particular (physical) entity, its lifecycle, and software that allows to store and use data about the entity throughout its lifecycle. For instance, the following definition combines these three elements, and focuses on the maximal virtualization of a physical product: ''Digital Twin is a set of virtual information constructs that fully describes a potential or actual physical manufactured product from the micro atomic level to the macro geometrical level. At its optimum, any information that could be obtained from inspecting a physical manufactured product can be obtained from its Digital Twin'' [2].

Given the scope of definitions such as this, DTs are thought to be characterized by high-rate flows of heterogenous data, which raise the question of how to assess the trustworthiness of such data in

KGCW'25: 6th International Workshop on Knowledge Graph Construction, June 1 or 2, 2025, Portoroz, SVN

* Corresponding author.

† These authors contributed equally.

jos.lehmann@dlr.de (J. Lehmann); martin.lanz@dlr.de (M. Lanz)

[1] In this submission some project-related names are anonymized (as ProX).

order to ensure the DT's accuracy and reliability [3]. This question is particularly pressing for ADTs. An aircraft's lifecycle (from pre-design through to operations and maintenance) generates large amounts of data, for instance from onboard sensors or inspection equipment. All these data need to be organized, verified, and traced, for the ADT to deliver reliable predictions for health-monitoring or safety compliance.

Approaches to data provenance may vary over many dimensions, from the adopted definition of provenance down to the particular software or technology the approach is developed for [4] [5]. A key dimension is the type of information ecosystem an approach is meant to support. Controlled or rigid ecosystems, which allow only vetted operations and operators on homogenous data, for instance blockchains, can be equipped with efficient data provenance pipelines because of the limited data heterogeneity that characterizes them. Instead, ecosystems that cannot guarantee as much control or rigidity, and that require more user and data heterogeneity, for instance research or design data management systems, need to be equipped with data provenance pipelines that include semantic services. Ontology-based data provenance provides a flexible way to track and enrich data by linking it with clear relationships, making it possible to analyze differences and resolve inconsistencies between the physical and virtual models of an aircraft. This allows the DT to stay synchronized, enabling simulations, diagnostics, and optimization throughout the aircraft's lifecycle.

The next section provides an initial problem definition and possible approaches from the literature. Subsequent sections introduce the working architecture of a Data Provenance and Metadata Demonstrator (DPMD); describe recent DPDM developments in the area of knowledge graph construction[2]; draw some conclusions; outline future research directions.

2. Problem Definition and Possible Approaches

The following preliminary description of elements and phases in data provenance collection in a DT has guided subsequent work on problem definition and literature review in the project DigECAT[3] (Digital Twin for Engine, Components and Aircraft Technologies):

a. Data and Tool Management in a DT infrastructure, i.e. functionalities that enable the extraction of metadata from data storages.

b. Provenance Collection, i.e. a messaging pipeline from a to c, including automated filtering, aggregation and metadata extraction functionalities.

c. Provenance Storage, i.e. functionalities that support the storage of the output of Provenance Collection into a Triplestore with an ontology as conceptual schema that enables the integration of heterogenous provenance information, also by means of automated reasoning services.

d. Provenance Analytics, i.e. functionalities that support users in analysing provenance in various use cases.

2.1. Approaches to Data and Tool Management

Data and tool management may consist of methods that require decreasing levels of data curation, on a spectrum from data warehouses to data lakes. Data warehouses store processed and filtered data, making data ready for analysis, while data lakes store raw data, offering greater flexibility and scalability for handling diverse data types. Domain-specific data warehouses, such as PANGAEA [6],

[2] Throughout this paper the terms 'generation', 'creation' and 'construction' in the context of either knowledge graphs or ontologies are used synonymously.

[3] https://www.dlr.de/en/ki/research-transfer/projects/digecat

cater to particular scientific fields. In the context of Digital Twins, cloud services such as Azure Digital Twins[4] provide a structured environment suited for integrating various data sources. Conversely, data-lake-like solutions, such as Catena-X[5], maintain local data formats while promoting interoperability through semantic models. Federated database systems [7] also enable flexible integration across heterogeneous data sources. Furthermore, solutions like shepard [8] and twinstash [9] support the storage and management of heterogeneous product and research data (scientific data resp. DTs), contributing to an ecosystem that balances curated and flexible data management approaches.

In this research, the latter two proposals have a role to play in data and tool management (item a. above).

2.2. Approaches to Metadata Extraction

Metadata extraction can serve as either a preliminary step or an integral component of provenance collection. [10] [11] describe this process across four key dimensions: the categories of metadata it addresses (intra-object, inter-object, and global); the underlying modeling approaches (e.g., graph-based structures, data vaults); the methods employed for generating metadata (data ingestion tools, specialized extraction tools, metadata frameworks, or ad-hoc algorithms); the essential system features needed to support data lakes.

In the current version of DPMD, metadata extraction relies on the following:

- Intra-object metadata (e.g., dataset properties such as file names, sizes) and global metadata (e.g., semantic resources, both domain-specific and machine learning-oriented ontologies, as well as logs).

- A knowledge engineering-based modeling method. This approach is analogous to composition-centered models (which decompose data objects into component parts and arrange them within a graph) as well as to the satellite structures of data vaults (which facilitate the incremental addition of new object types).

- Features such as semantic enrichment, link generation, and metadata polymorphism, implemented through script-based metadata generation.

2.3. Provenance Collection

Ontology-based provenance typically relies on the PROV Ontology (link in footnote 9 below), which supports the creation of instances (a.k.a. individuals) of classes such as Activity, Agent, Entity, Role, and the assertion or inference of relations among such instances (such as being attributed to, being associated with, etc.). Figure 1 exemplifies a basic case of how instances are constructed:

- In (a) line 14, Bob appears as “data source creator” in the metadata; this key is modelled as the label of a subclass of Person in PROV, allowing the python script ms.py (for metadata structuring) to match the key and create instance Bob in (c).

- In (b) line 9, “state” appears with value “running” in the metadata; this key is modelled as the label of a property of entity Software Agent, characterized in PROV as “running software”, allowing ms.py to match the key and create instance “state” -- admittedly a misnomer -- in (c).

[4] https://azure.microsoft.com/de-de/products/digital-twins
[5] https://catena-x.net/de/

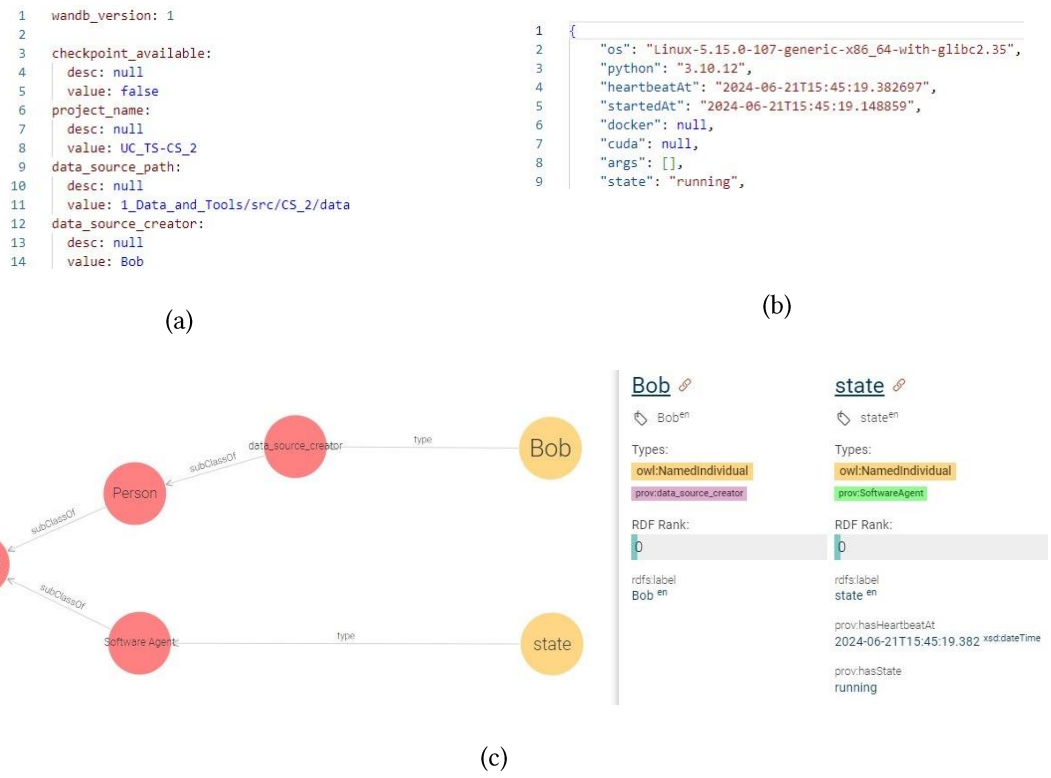

Figure 1: CS2's metadata for data source creator (a) and the ML algorithm (b) and resulting knowledge graph (c).

3. Working Architecture and Implementation

The main components of the DPMD architecture are listed in Figure 2 and illustrated in Figure 3. Fourth-level items in the list indicate that the corresponding functionality is at least in part implemented for two cases studies (CSs) as a python/SPARQL/owlready script ms.py.

Subsequent subsections provide details on items 1., 2., 3. below, including example ontology DigECAT-Prov.

Furthemore, section 4 expands on ongoing upgrade activities:

- the transition to a recursive rather than iterative version of ms.py, as described in point (6) in Figure 2;

- in cases of metadata not matching the knowledge base, i.e. when (iii) in point (6) in Figure 2 does not hold, the support to knowledge graph creation offered by an LLM, specifically in the areas of Figure 2 printed in red.

1. Data Collection
   (1) Tools + Data described by metadata
      - Functionalities: Generate Metadata
        - In CSs: wandb metadata stored as nested dictionaries or lists in .json and .yml files
2. Metadata Extraction
   (2) Load Metadata
      - Functionalities: Load Metadata
        - In CSs: python call at present included in code for 3.(6) below
   (3) Preprocess Metadata
      - Functionalities: Clean and Standardize Metadata
        - In CSs: various python calls at present included in code for 3.(6) below
   (4) Check Metadata
   (5) Add Metadata
3. Provenance Collection
   (4) Check Metadata
   (6) Create Knowledge Graph
      - Functionalities: Match Metadata to Ontology; Create Individuals for Metadata in Ontology
        - In CSs: five iterative loops in ms.py that:
          (i) access metadatum in nested dictionary or list, as key or value
          (ii) search for key in ontology through SPARQL query
          (iii) if key is found
            a. as a class do (iv)
            b. as property, search for domain and range
          (iv) create provenance-relevant individuals and/or properties
   (7) Provenance Store
      - Functionalities: Load Created Individuals on store
        - In CSs: GraphDB API in ms.py to load created knowledge graph into GraphDB
   (8) Quarantine Data
4. Provenance Analytics
   (9) Monitoring
   (10) Analytics and Visualization

Figure 2: Data Provenance and Metadata Demonstrator Functionality Description.

3.1. Data Collection

Flight maneuver classification is a key task in evaluating an aircraft's stability and safety during performance tests. Leveraging data from a specialized platform for managing digital twins, such as twinstash, engineers use labeled flight test sensor data to train machine learning models, as documented in [12].

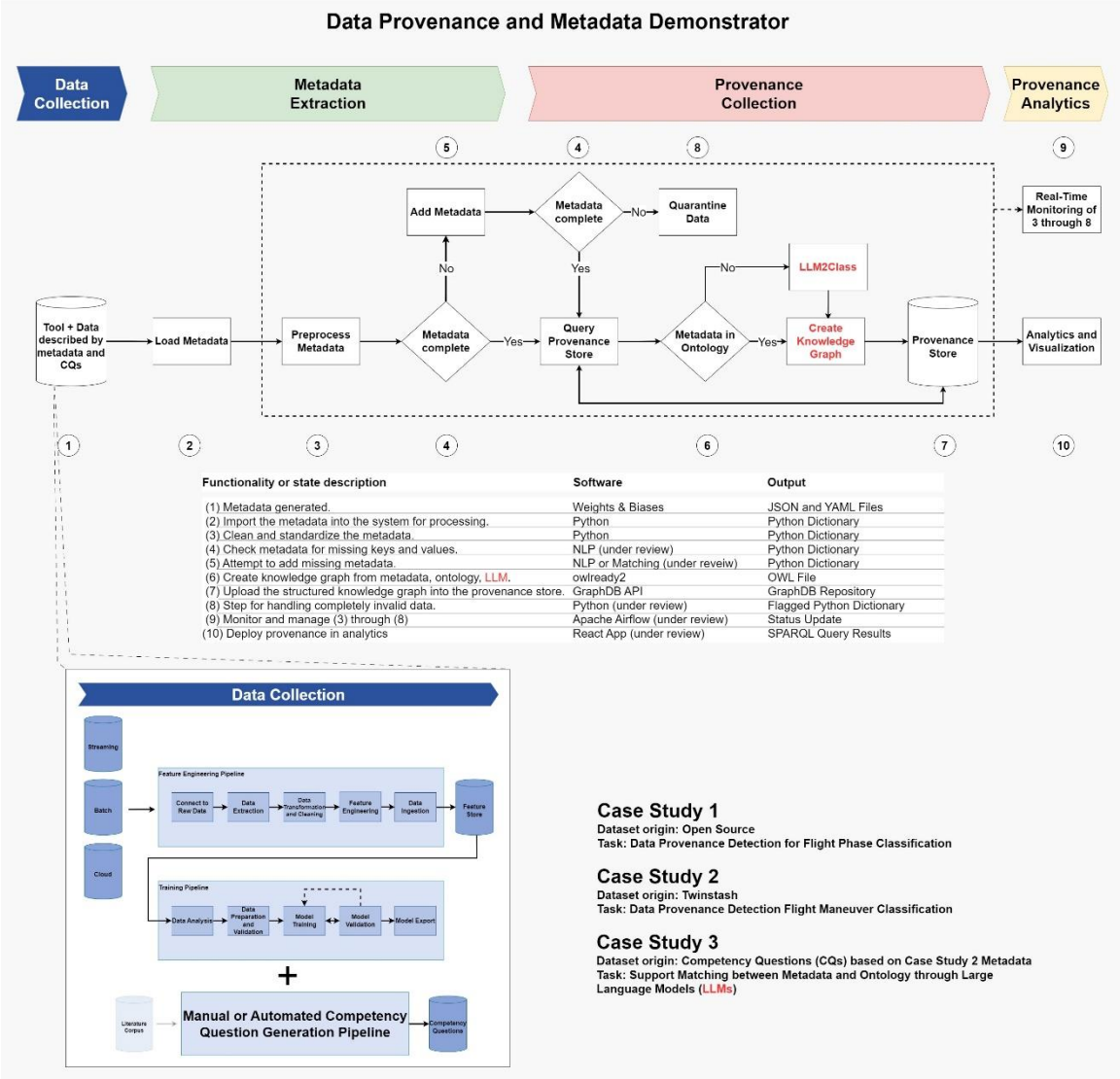

Figure 3: Data Provenance and Metadata Demonstrator Working Architecture.

3.2. Metadata Extraction

Once sensor data from flight tests are labeled with specific maneuvers, they are used to train machine learning models. Metadata about the training process captured during the training process, including system configurations, sensor calibrations, and training results, are tracked using wandb. This provides input to provenance collection.

Table 1 breaks down and categorizes the metadata for case study CS2. The categorization is based on the following interpretation of the metadata categories presented above:

- intra-object metadata describe only (part of) a particular dataset (e.g., file names and sizes);
- inter-object metadata describe dataset groups that are related (e.g., "2016 Flight Data," "2017 Flight Data,"), or are similar (e.g., Dataset A and Dataset B contain similar types of turbulence measurements) or belong to the same lineage (e.g., Dataset C was generated by processing a raw Dataset D);
- global metadata describe concepts valid on entire data lake (e.g. "Flights," "Maneuvers," "Sensors,") which enable semantic integration, or global indexes (e.g. indexes listing datasets,

files, or tables in a data lake, including references to storage locations, access methods, last update times) which improve discoverability and retrieval, or user interaction logs (e.g., searches performed, datasets accessed, analyses run) which provide a system-wide perspective on usage patterns.

Table 1
Breakdown and categorization of the case study CS2 metadata

Step	Description	Meta-data category
1. Project Metadata	Metadata regarding project configuration and data source information.	
- Project Name	UC_TS-CS_2	Global
- Data Source Path	1_Data_and_Tools/src/CS_2/data	Intra
- Data Source Creator	Bob	Intra
- Data Source Location	www.gitlab.dlr.de	Intra
- Data Source Type	API	Intra
2. Model Training Metadata	Metadata describing the machine learning model's configuration during training.	
- wandb Version	1	Intra
- Epochs	1	Intra
- Batch Size	64	Intra
- Save Top K	1	Intra
- Model Type	FCNN	Intra
- Run Feature Engineering	True	Intra
- Early Stopping	False	Intra
3. System Metadata	Information about the system environment where the model training was executed.	
- Accelerator	GPU	Intra
- Operating System	Linux-5.15.0-107-generic	Intra
- Python Version	3.10.12	Intra
- CLI Version	0.17.1	Intra
4. Code Execution Metadata	Metadata providing details about the code and execution environment.	
- Code Path	code/1_Data_and_Tools/src/CS_2/main.py	Intra
- Git Commit	a0f42659643694bf04efeef02c0132c146127e7c	Intra
- Program State	Running	Intra
5. Data Dimensionality and Features	Detailed information about input data and output labels.	Intra
- Flight Name	ProX16a	Intra
- Sensor Names	ABSHUM, IRS_EWV	Intra
- Maneuver Labels	Unclassified, Take-Off, Descent, Approach, Landing, Climb, Turn Left, Turn Right, Dutch Roll, elevator Sweep, Steady Pull-Up, Steady Push-Over	Global

3.3. Provenance Collection

Provenance Collection takes place across 6 and 7 in Figure 3, resulting in a RDF store structured by the ontology DigECAT-Prov-rd, a reduced version of the ontology DigECAT-Prov. Both import the same set of ontologies, covering the knowledge domains described below, relevant to provenance in DigECAT[6].

Air Traffic Management (atmonto[7]) specifies knowledge about air traffic, including classes for specific aircraft models, equipment, manufacturers;

Aircraft Design (Aircraft Ontology[8]) specifies knowledge about aircraft components and parameters, based on an ontology of quantities;

Provenance (PROV-O[9]) specifies the basic conceptual structure for provenance;

Provenance in Machine Learning (PMLM[10]) specializes PROV-O with classes for Machine Learning (ML).

Figure 4 shows, highlighted in blue, CS1's and CS2's foci on DigECAT-Prov's Class Hierarchy. Except for the top classes, which are imported from PROV-O, most of the other classes with prefix "prov:" are case-study-specific modifications. Classes with other prefixes are imported from the DigECAT-Prov-rd imports above.

Figure 5 shows the knowledge graph resulting from functionality 7 for CS2. Above the figure, vertical lines indicate what ms.py takes as input, i.e. DigECAT-Prov-rd's classes and properties, some modeled into the ontology to match the metadata, and what it returns, i.e. individuals and relationships for the extracted metadata. Note that Figure 5 does not show ms.py's raw output: redundant entities were deleted from the underlying graph and entity names shortened for readability (only the branch below LabelledDataset retains the original camel notation).

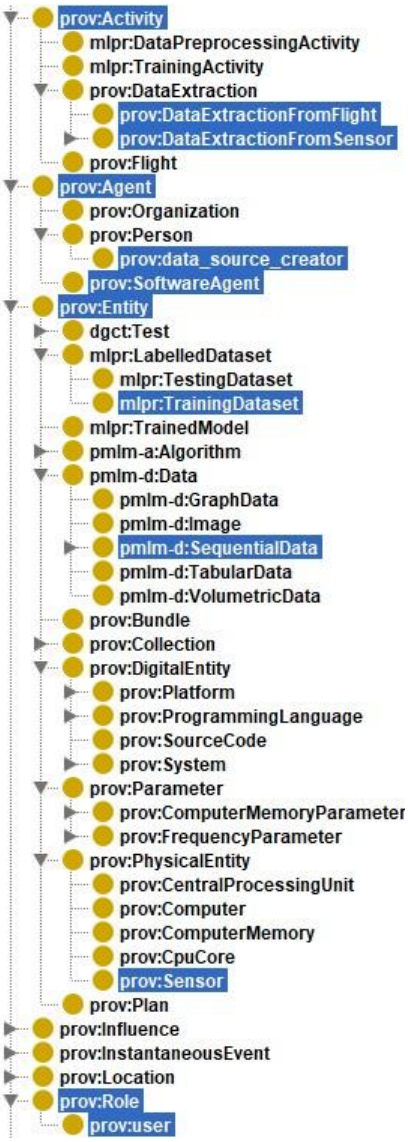

Figure 4: Class Hierarchy for CSs.

In a nutshell, a training data set is attributed to Bob, a data source creator, who was associated with user role dlr; flight activity ProX16a influenced a data extraction from flight activity, labelled ProX16a, which in turn informed two data extraction from sensor activities and has a qualified association with user13. Figure 5 shows present shortcomings of ms.py: the relation between user13 and dlr is not asserted, leaving unspecified the relation between training data set and data extraction activities. This depends on semantic gaps in the metadata or in ms.py itself.

[6] Note however that DigECAT-Prov-rd does not import all atmonto, to avoid inconsistencies detected in some atmonto's imports

[7] https://data.nasa.gov/ontologies/atmonto/NAS

[8] https://github.com/astbhltum/Aircraft-Ontology

[9] https://www.w3.org/TR/prov-o/

[10] https://github.com/KnowledgeCaptureAndDiscovery/pmlm

Input to ms.py from ontology imports	Input to ms.py modeled into ontology to match metadata	Output ms.py (constructed knowledge graph)

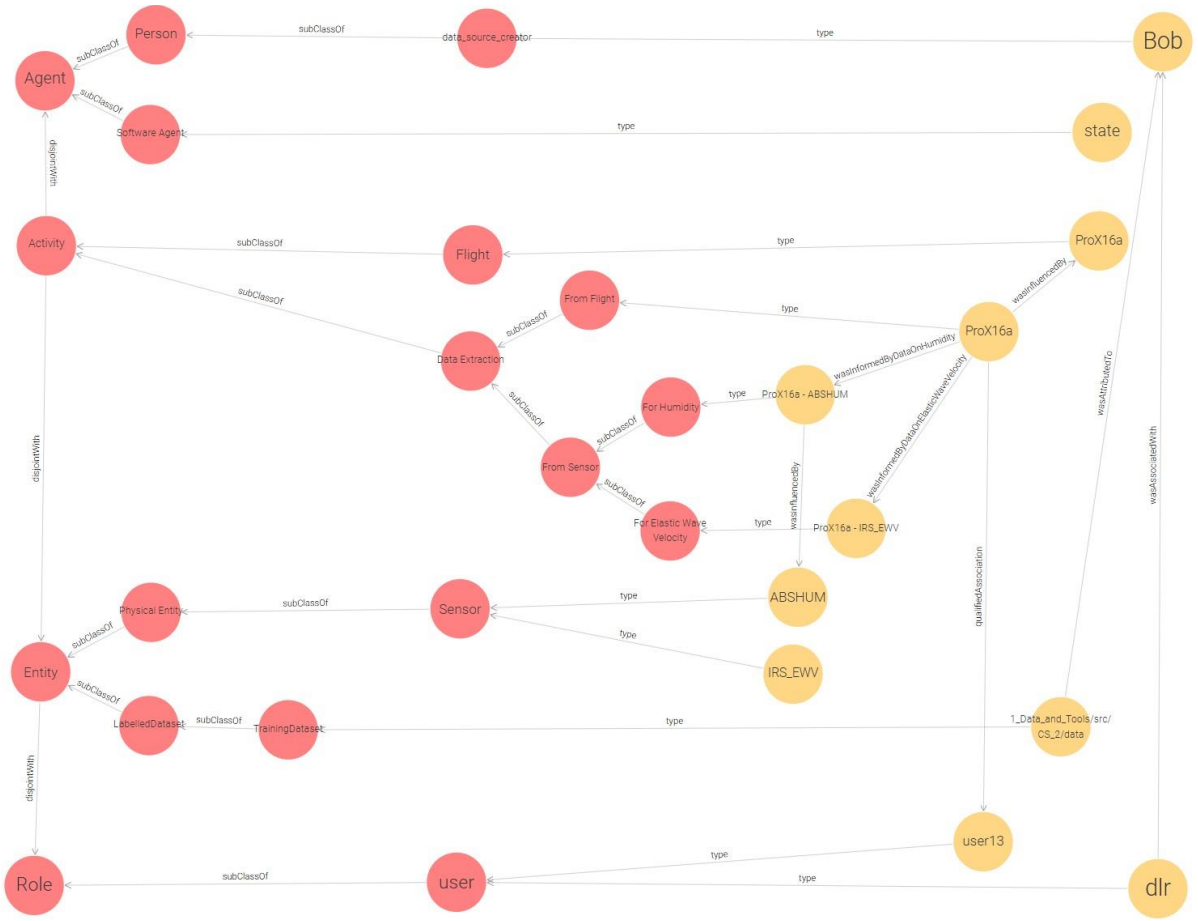

Figure 5: CS2's knowledge graph in GraphDB.

4. Ongoing DPMD upgrade activities

4.1. Recursion

The current version of step 6 of Figure 3 is based on a fixed number (namely, 5) of nested loops, in which the same calls are reiterated on the nested metadata. This poses both robustness and maintainability issues. Robustness is hindered in cases where the metadata nesting either takes place through varying data structures (e.g. dictionaries and/or lists) or has more levels than the nested loops in the code. Maintainability is compromised because for the code to work it may need to be updated depending on the metadata at hand.

In order to overcome these two sets of problems, a recursive extraction of the metadata to lists is being tested. Such extraction can cope with an undetermined number of nesting levels, as well as nesting in varying data structures.

While such extraction simplifies access to the metadata, it does bring its own code transparency challenges. In particular, the iterative version of the routine at step 6 of Figure 3, can be understood as a meta-programming function, which includes exec() calls for the creation, instantiation and evaluation of meta-variables for dictionary keys and values at each nesting level. Such variables are used in python/owlready clauses to assert individuals and/or properties in a knowledge graph based on the DigECAT-Prov-rd ontology.

The recursive version of the routine at step 6 of Figure 3 allows to avoid loop nesting, but requires that the names of the created meta-variables are indexed using the indexes in the metadata lists. This, in turn, requires the code to create meta-meta-variables, which are used in python/owlready clauses to assert individuals and/or properties.

4.2. LLM supporting Knowledge Graph Construction

Recent research (mid-2024 onward) highlights the growing synergy between large language models (LLMs) and knowledge graphs (KGs)/ontologies. LLMs are increasingly being utilized to automate and accelerate traditionally labor-intensive ontology engineering tasks, ranging from schema creation to knowledge extraction [13]. By leveraging LLMs' advanced natural language understanding, researchers have begun to (semi-)automate ontology generation, integrate diverse data through semantic classification, and enrich KGs with enhanced provenance and metadata tracking. These advancements are fostering more dynamic and adaptive knowledge representations, though challenges such as factual accuracy and alignment with existing ontologies remain [14] [15]. The remainder of this chapter presents a method for classifying concepts and their corresponding instances using an LLM, aimed at filling missing gaps within a provenance pipeline. Initial results indicate highly promising accuracy, with the model producing reliable classifications. Future work will focus on extending this approach to enable precise placement of newly added relationships between concepts, ensuring that the ontology remains coherent and contextually complete.

4.2.1. LLM2Class - Method

At step 6 of Figure 3 the pipeline checks whether a piece of metadata is found in the ontology by executing a SPARQL query. If the piece of metadata is found, the knowledge graph can be constructed. If not, the LLM2Class sub-routine is triggered.

The following three steps summarize the main functionalities of this sub-routine, which intervene between the check Metadata in Ontology and the Create Knowledge Graph functionality, as also detailed in Figures 6 and 7:

- LLM2Class classifies piece of metadata by mapping it to the most appropriate ontology class.
- If subclasses exist, the process iterates deeper into the ontology.
- Once such classification step is complete, the piece of metadata is integrated into the knowledge graph.

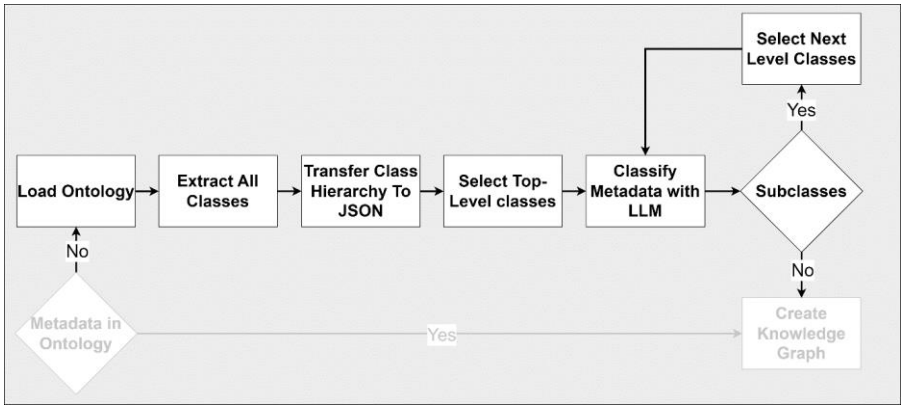

Figure 6: LLM2Class sub-routine

(6) (iii) - 1. Load Ontology
- The system loads the ontology to retrieve the existing class structure

(6) (iii) - 2. Extract All Classes
- It extracts all ontology classes, including rdfs:Class and owl:Class, forming a hierarchical structure.

(6) (iii) - 3. Transfer Class Hierarchy to JSON
- The extracted class hierarchy is stored in JSON format, allowing efficient traversal.

(6) (iii) - 4. Select Top-Level Classes
- Identifies root ontology classes (those without parent classes) as starting points for classification.

(6) (iii) - 5. Classify Metadata with LLM
- The LLM receives the metadata and a list of top-level ontology classes with their definitions.
- The LLM assigns the metadata to the most suitable parent class.

(6) (iii) - 6. Check for Subclasses
- If the assigned class has subclasses, the process iterates and refines classification at the next level.

(6) (iii) - 7. Select Next-Level Classes (If applicable)
- The LLM classifies again based on more specific subclasses until the best match is found.

(6) (iii) - 8. Forward Classification Result
- Once the most specific class is determined, it is passed to the Create Knowledge Graph step.

Figure 7: Analytical description of LLM2Class sub-routine when (6) (iii) in Figure 2 fails.

4.2.2. LLM2Class - Prompting

In the LLM2Class sub-pipeline, a carefully designed LLM prompt plays a crucial role in mapping metadata to the appropriate ontology class. Since classification occurs at multiple levels of the ontology's *is-a* hierarchy, the prompt adapts dynamically based on the current classification stage.

The prompting strategy ensures that the LLM receives contextual information about ontology classes while maintaining a structured output format. The dynamic nature of the prompt allows it to:

- Present only the relevant classes based on the current hierarchical level.
- Include descriptions of candidate classes to improve classification accuracy.
- Guide the LLM towards a structured three-part output:
    a. Metadata
    b. Description (generated textual summary of the metadata)
    c. Most Convenient Class (ontology class assigned by LLM)

4.2.2.1. Prompt Template

The following is the prompt template used in classification:

> I am working on incorporating concepts into the PROV-O ontology. The concept given should be categorized as one of the following classes. Note that for each class you have a description which allows you to get further information of how to categorize the concept.
>
> **Classes:**
> **{target_classes}**
>
> **Metadata:**
> **{concepts}**
>
> Classify the given concept to the most convenient class along a description of the concept and provide the result in the following output format:
>
> ["<Metadata>", "<Description>", "<Most Convenient Class>"]
>
> Example Output:
> ["Neural Network Layer", "A dataset used for training machine learning models.", "Activity"]
>
> Make sure that the output contains one single list and nothing else. If no output matches, return "None" as the <Most Convenient Class>.

4.2.2.2. Dynamic Adaptation of the Prompt

The placeholders {target_classes} and {metadata} are dynamically replaced at runtime depending on the classification step:

Table 2
Prompt dynamic adaptation

Placeholder	Description	Dynamic Replacement
{target_classes}	Ontology classes at the current hierarchy level	Updated based on subclass candidates
{metadata}	Metadata to be classified	Metadata instance being processed

- At the top level, {target_classes} contains only root classes.
- If a subclass exists, {target_classes} is updated to reflect only subclasses at the next level.
- This iterative refinement continues until a final classification is reached.

4.2.2.3. Example of Fully Populated Prompt

At the first classification step, the LLM receives the following structured prompt to classify a given concept within the PROV-O ontology. The prompt provides a list of candidate ontology classes, each with a description, to assist in determining the most appropriate classification. The LLM processes this information and returns a structured output indicating the concept, a brief description, and the most suitable class. If no valid classification is found, the response defaults to "None" to ensure accuracy.

⋮ [As per template]

Classes:

Agent:
Description: An agent is something that bears some form of responsibility for an activity taking place, for the existence of an entity, or for another agent's activity.

Activity:
Description: An activity is something that occurs over a period of time and acts upon or with entities; it may include consuming, processing, transforming, modifying, relocating, using, or generating entities.

Entity:
Description: An entity is a physical, digital, conceptual, or other kind of thing with some fixed aspects; entities may be real or imaginary.

Concept:
OperatingSystem
⋮ [As per template]

4.2.3. LLM2Class - Example Predictions

The table below presents metadata classifications generated by the LLM2Class subroutine. Each column represents an intermediate result at a specific hierarchy level. If an entry is "None", no valid class could be determined. In such cases, the final prediction is marked False if an appropriate classification was expected. Initially, the classification failed because the DigitalEntity class lacked a definition. After adding a definition, the classification was successfully evaluated as True.

Table 3
Example predictions

Metatada	Lv. 1	Lv. 2	Lv. 3	Prediction
OperatingSystem	Entity	None		False
OperatingSystem	Entity	DigitalEntity	System	True
os: Linux-5.15.0-107-generic-x86_64-with-glibc2.35	Entity	Digital Entity	System	True
DataSourceCreator - Bob	Agent	Person		True
username - dlr	Agent	Person		True
Sensor name: ABSHUM	Entity	PhysicalEntity		True

4.2.4. LLM2Class - Complexity Analysis

The complexity of LLM2Class depends on the ontology structure and classification depth.

Table 4
Initial complexity analysis

Step	Process	Complexity
Ontology Processing	Extracting and structuring classes	O(C)
Building Class Hierarchy	Identifying parent-child relationships	O(C)
Top-Level Class Selection	Finding root-level classes	O(1)
LLM-Based Classification	Querying the LLM at each level	O(H)
Hierarchical Traversal	DFS/BFS search through ontology levels	O(H)
Final Classification Storage	Writing JSON results	O(1)

Based on this analysis, the overall complexity would be: O(C+H) where:

- C = Total number of ontology classes
- H = Ontology tree depth

Best Case: Classification occurs at the top level ⟶ O(1)

Worst Case: Full-depth traversal requiring multiple LLM calls → O(C + H)

5. Conclusion

Initial analytical results of research and development activities on a pipeline for data provenance collection in the digital twin of an aircraft show the potential of a metadata- and ontology-based approach. Metadata is standardized with domain ontologies that extend a provenance ontology, allowing to assert standard compositional provenance statements. Such statements, or networks thereof, may contribute to enhance users' trust in a (A)DTs, by providing an analytical support to trace the provenance of data imported into or generated by the (A)DT.

However, bottlenecks that affect the pipeline's generality are apparent, such as varying metadata nesting, as well as ubiquitous mismatches between incoming metadata and the demonstrator's knowledge base. Currently, varying nesting is resolved by a fixed number of iterations while mismatches are resolved by modelling, increasing maintenance efforts, in turn limiting the pipeline's scalability. To cope with such robustness issues the pipeline is being further expanded through recursion resp. by calls to an LLM.

Finally, areas of future work are under consideration and may include theoretical analysis (for instance further grounding of the problem definition with respect to digital twins), more use cases (for instance, case studies for CAD models or for unstructured (textual) data); provenance analytics; provenance visualization.

Acknowledgements

Research supported by DLR's project DigECAT (Digital Twin for Engine, Components and Aircraft Technologies). Figure 4 created with Protégé[11]. Figures 1(c) and 5 created with GraphDB[12]

Declaration on Generative AI

The author(s) have not employed any Generative AI tools for any of the contribution types listed in taxonomy on ceur-ws.org/genai-tax.html.

References

[1] E. Karabulut, S. F. Pileggi, P. Groth and V. Degeler, "Ontologies in digital twins: A systematic literature review, Future Generation Computer Systems," *Future Generation Computer Systems,* vol. 153, pp. 442-456.

[2] M. Grieves and J. Vickers, "Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems," in *Transdisciplinary Perspectives on Complex Systems: New Findings and Approaches*, Springer, 2017, pp. 85-113.

[3] S. Suhail, R. Hussain, R. Jurdak and C. S. Hong, "Trustworthy Digital Twins in the Industrial Internet of Things with Blockchain," *IEEE Internet Computing*, Vols. 26,3, p. 58–67, 2022.

[4] M. Herschel and M. Hlawatsch, "Provenance: On and behind the screens," in *In Proceedings 2016 International Conference on Management of Data*, 2016.

[11] http://protege.stanford.edu
[12] https://graphdb.ontotext.com/

[5] C. Putnam, J. Waters, J. Chao and B. Rasmussen, "Enabling and Managing Data Provenance in a Command and Control Environment," in *Proceedings 1st IEEE International Conference on Human-Machine Systems*, 2020.

[6] M. Diepenbroek, H. Grobe, M. Reinke, U. Schindler, R. Schlitzer, R. Sieger and G. Wefer, "PANGAEA-an information system for environmental sciences," *Computers & Geosciences,* vol. 28, pp. 1201-1210, 2002.

[7] L. G. Azevedo, E. Figueiredo de Souza Soares, R. Souza and M. Ferreira Moreno, "Modern Federated Database Systems: An Overview," in *Proceedings 22nd International Conference on Enterprise Information Systems (ICEIS 2020)*, 2020.

[8] T. Haase, R. Glück, P. Kaufmann and M. Willmeroth, "shepard - storage for heterogeneous product and research data," Zenodo, 2021.

[9] S. Haufe, M. Bäßler, C. Pätzold, M. Tchorzewski, H. Meyer, E. Arts and A. Kamtsiuris, "Digital Twins Storage and Application Service Hub (Twinstash)," in *Deutscher Luft- und Raumfahrtkongress 2022*, Dresden, 2022.

[10] P. Sawadogo, T. Kibata and J. Darmont, "Metadata Management for Textual Documents in Data Lakes," in *Proceedings 21st International Conference on Enterprise Information Systems (ICEIS 2019)*, 2019.

[11] P. Sawadogo and J. Darmont, "On data lake architectures and metadata management," *Journal of Intelligent Information Systems,* vol. 56, pp. 97-120, 2021.

[12] P. Schmalbruch, „Vergleichende Betrachtung von Deep Learning Ansätzen zur automatischen Detektion von Flugmanövern," Ostfalia - Hochschule für angewandte Wissenschaften, Wolfsburg, 2024.

[13] C. Shimizu, "Accelerating Knowledge Graph and Ontology Engineering with Large Language Models," 2024.

[14] V. K. Kommineni, "From human experts to machines: An LLM supported approach to ontology and knowledge graph construction," 2024.

[15] L. Ding, "Automated Construction of Theme-specific Knowledge Graphs," 2024.