Deutscher Luft- und Raumfahrtkongress 2022
DocumentID: 570066

# COMMON SOURCE & PROVENANCE AT VIRTUAL PRODUCT HOUSE: INTEGRATION WITH A DATA MANAGEMENT SYSTEM

F. Dressel[*], M. Rädel[†], A. Weinert[‡], M. Struck[‡], T. Haase[§], M. Otten[¶]

[*] German Aerospace Center (DLR), Institute of Software Methods for Product Virtualization, Dresden, Germany
[†] German Aerospace Center (DLR), Institute of Composite Structures and Adaptive Systems, Bremen and Braunschweig, Germany
[‡] German Aerospace Center (DLR), Institute for Software Technology, Cologne and Bremen, Germany
[§] German Aerospace Center (DLR), Center for Lightweight Production Technology, Augsburg, Germany
[¶] University Bremen, Center for Industrial Mathematics, Bremen, Germany

**Abstract**

At the Virtual Product House (VPH) a multi-stakeholder process concerning the development and certification of airplane components is implemented. An orchestrating tool is used that allows stakeholders to contribute their abilities to a common workflow. In order to allow a comprehensible virtual certification the generation and storage of provenance attributing the defined workflow execution and data flow is needed. In this paper the requirements we discovered, the improved structure of the provenance containers we used, the usage of a data management system to store and analyze the provenance & data and its application to the VPH process are described.

**Keywords**
Persistent data management, Provenance, Common source

## Acronyms

| | |
|---|---|
| API | Application Programming Interface |
| DLR | German Aerospace Center |
| JSON | JavaScript Object Notation |
| RCE | Remote Component Environment |
| REST | Representational State Transfer |
| VPH | Virtual Product House |

## 1. REQUIREMENTS IN COLLABORATIVE WORKFLOWS DURING VIRTUAL PRODUCT DEVELOPMENT

Above a certain level of product complexity, a development process requires multiple partners with specialized skills for certain aspects of the engineering processes. In addition to their individual contributions to the development process, the project partners must orchestrate the interaction of these sub-processes. In the VPH start-project we developed the common-source architecture, a software setup for this type of problem [1]. We moreover applied it to the virtual development of an aircraft moveable [2, 3, 4].

This architecture offers solutions and tools for collaborative product development while retaining the intellectual properties on data and assessment capabilities of each partner. It moreover allows the generation of provenance information of all data throughout of the process as well as the persistent storage of all information. The provenance information is needed for virtual certification.

In this paper we describe our approach for combining common source, provenance and a data management system within the context of the VPH to implement a consistent data handling for virtual certification. To do so, we first present the context, use case and derived requirements in the remainder of this chapter. Chapter 2 gives a short introduction to provenance and the provenance container concept, which we extended based on our previous work. Chapter 3 gives an overview about the data management system and the data structure we used and describes briefly its integration into the common source VPH process. Chapter 4 shows exemplary approaches to work with the provenance in the whole system we designed. Chapter 5 concludes and summarizes our work.

### 1.1. Context

The VPH is an integration and test center for virtual product development. The goal is to combine the capabilities in this field available at the institutes of the German Aerospace Center (DLR) with regards to different aspects, e.g. aerodynamics, structures

and systems. This includes technical aspects and issues of digitization and virtual collaboration. These capabilities can be combined with applications from partners from industry and research and are applied to use cases with the goal of a more simulation-based/virtual certification.
Initial use cases for the demonstration are the moveable of a passenger-aircraft configuration [5] and a liquid hydrogen $(\mathrm{LH}_2)$ tank [6].
The development process is implemented in *Remote Component Environment (RCE)* [7, 8] workflows. RCE allows workflow contributors to publish their tools on a server. A tool interface is defined by the in- and outputs. The execution itself is performed as a black-box at the corresponding tool server.
A RCE workflow describes the execution order and data transfer in a process. The workflow host has access to the initial input data as well as all data that is created and available as output during the workflow execution. Otherwise, the data is distributed to the clients, as seen in Figure 1a.

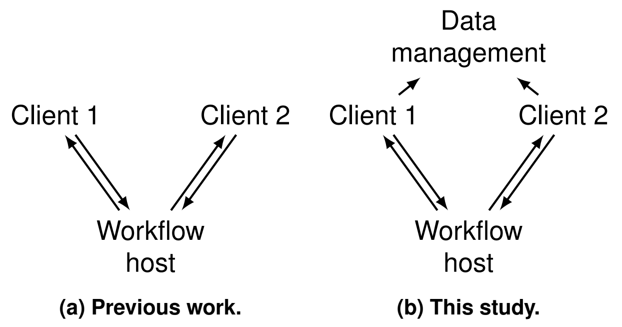

**FIG 1. Data flow.**

The heterogeneous nature of the data produced by the different assessment steps in the virtual product development process creates certain requirements on the data handling scheme, which we discuss in the following.

## 1.2. Requirements

The VPH use cases are quite complex. In a classical design process, engineers from one project partner would collaborate to analyze some property of a model. To do so, they often use proprietary tools and subsequently share the resulting data with engineers from other project partners. This data sharing typically happens asynchronously, e.g., via email or network shares.
It is one goal of the VPH to automate the evaluation of models and the sharing of data. Each stakeholder contributes functionality for their respective domain. They do so based on an input-process-output model: Each functionality accepts data in a predefined format and returns data in a (probably different) predefined format. The engineers chain the functionalities according to the needs of the design process in a workflow. Executing such a workflow requires orchestrating the data flow and tool execution.

Data flows used for virtual certification have to be reviewable. During certification, it is decided whether or not the given results are usable for a certification task based on the data and its provenance (see, e.g., [9, 10]). To be reviewable, the data needs to be persistent, accessible, consistent & traceable, attributable and any changes to the data need to be traceable. We describe these functional requirements and their manifestations in the context of VPH in the following. Subsequently, we address the non-functional requirements that originate from the business interests of the individual partners.

### Persistence

All data used in the virtual certification must be persistently stored over the full life time of the certified product according to EU regulation 748/2012, 21.A.55 [11]. In aerospace engineering, certified products usually have a life span of multiple decades. During the life time of a product, the industry standards for data storage are prone to change. Hence, the storage for certification data of aerospace products needs to take into account changing data management systems and necessary migrations between such systems.

### Data accessibility

During the development and execution of the workflow, all partners share data and functionality. The result of each workflow run is a complete data set for the given aerospace problem addressed with the workflow. Within the VPH, each partner contributes openly. The resulting data needs to be accessible by each of the partners. Later on, data may also be available for others. For such a case, access rights and permissions need to be in place (see also the A in FAIR [12] with that respect for this discussion).

### Data consistency & traceability

The tools, dataflow and data are constantly evolving during the virtual engineering processes and the development of the workflow. Multiple versions of the tools and data used and generated will be available. However, only one final data set will be certified and therefore a way to pull out data which belongs together out of the remaining data is needed.

### Attributability

Multiple stakeholders participate in the development of the workflow. Each stakeholder may adapt one or more system properties during the engineering process. Therefore, it is necessary to determine whether a specific project partner may be held liable for a given piece of data. Each data point or set of data has to be clearly attributed to one or more individual workflow steps, which has in turn to be clearly attributed to one or more stakeholders.

Modification detectability

Once certification is complete, the certified data need to be stored for the lifetime of the product. Ideally, these data should be immutable to prevent malicious or negligent changes after certification. While hardware-based approaches to immutable data storage exist, applying these makes sense only in certain specific scenarios. In the context of VPH we aim to be able to detect after-the-fact alterations to the data and require that each change has to be easily traceable.

Non-functional requirements

The requirements discussed so far contribute to the goal of generating results from a distributed virtual engineering process. The process involves multiple partners, both from industry and academia. On the one hand, each partner wants to provide their abilities to the other project members. On the other hand, they have an interest in protecting their intellectual property. This includes both the source code of their tools as well as executables, which may be disassembled. Therefore, each partner must be able to publish functionality without publishing the tool itself.

### 1.3. Previous work

In previous work [1] we showed the basic concepts for provenance (in the form of provenance containers) and common source.

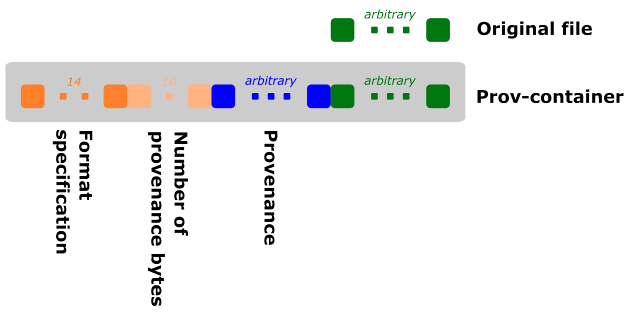

**FIG 2. Provenance container format from previous work [1]. Orange indicates control information, blue indicates provenance and green the original data. While the number of bytes for the control information is fixed, the length o provenance and data may be arbitrary.**

We combined provenance and data together in one file. The file contained some control information as headers, followed by the provenance and data (see Figure 2 for the file layout).

We used RCE [7] as workflow orchestration tool to implement the core concepts of the common source approach. The data exchange between the tools of the different stakeholders was also done via RCE and file exchange. The files were stored locally on the computer of the person, who started the workflow. For the tool run time, the data used as input/output of a tool are additionally stored on the respective tool server. The person executing the workflow was the only person who got a complete data set of a whole workflow run (see Figure 1a).

## 2. PROVENANCE & PROVENANCE CONTAINER

Provenance is the information about the origin of data. The de facto standard for provenance is the W3C Prov model (see [10]). It consists of activities (actions which are done to generate data), agents (software and people, who perform the activities) and entities (data) as basic building blocks. There are relations between the elements defined. The model itself is extensible and besides the basic model there exist more elements for more specific use cases. We used only the basic building blocks with the attributes shown in Figure 3.
The RCE elements can be directly mapped to W3C prov model elements: activities are RCE components which are executed, agents are the tools and operators and entities are the used and generated files.

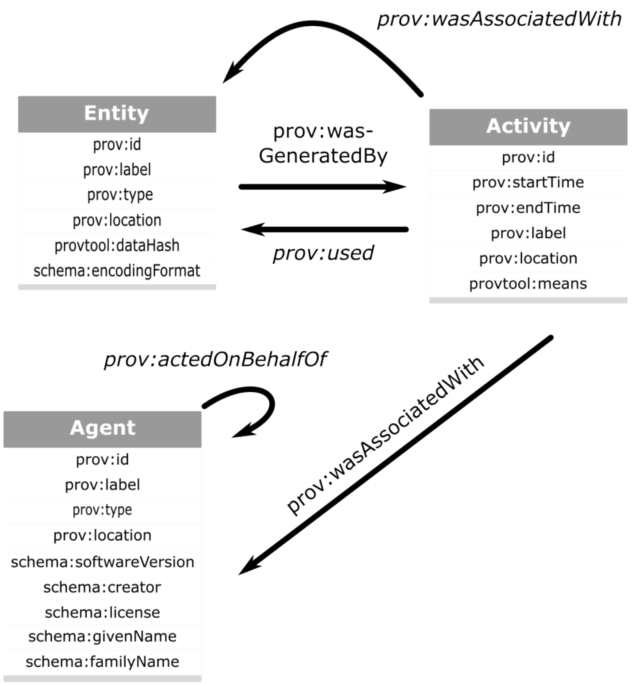

**FIG 3. Provenance model used in this study. It is based on the W3C Prov model but uses only a subset of all possible elements and relations.**

We use provenance containers to bring data and provenance together. We used an enhanced two-file layout for the study presented in this paper (see Figure 4). The provennace recorded is always referring to the immediate activity. If data is used which is generated by another activity, it is linked via reference. The two-file layout provides the same benefits like the layout from the previous work:

- Effective immutability of data and provenance
- Simple layout
- Content addressable
- Easy integration into arbitrary data management systems

It mitigates problems with the one-file layout:

1) No preprocessing needed to use the standard tools for the data
2) Unnecessary control information removed
3) Better performance when analysing provenance for container with large data parts (some GB)
4) Data and provenance are allowed to have different protection levels (encryption, access, ...)

**1)** Data is kept *as is* in a file and can be directly read by the tools. **2)** Provenance is stored as plain JavaScript Object Notation (JSON) text and no further processing is needed to use it. **3)** If provenance is analysed, the data (which may be huge) does not have to be read. **4)** Data and provenance are now physically separated and different protection measures like encryption, access regulations, ... can be applied to both in distinct ways.

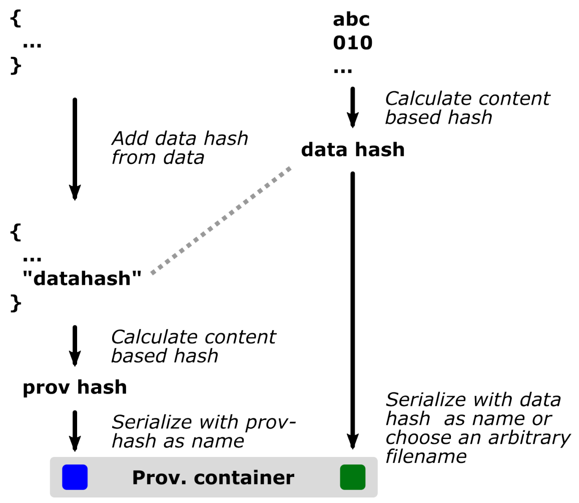
{
...
}
abc
010
...
Calculate content based hash
Add data hash from data
data hash
{
...
"datahash"
}
Calculate content based hash
prov hash
Serialize with prov-hash as name
Serialize with data hash as name or choose an arbitrary filename
Prov. container

**FIG 4. Two-file layout of provenance container. From top down: Provenance in json format and data is used to form a provenance container. First, the hash over the data content is calculated. Second, the provenance is extended to include the data hash as reference. Third, the hash over the provenance content is calculated. Fourth, data and provenance is serialized at arbitrary locations into something named according to the corresponding hash. The two serialized elements logically constitute the provenance container.**

## 3. DATA MANAGEMENT FOR COMMON-SOURCE & PROVENANCE

Some of the requirements identified in chapter 1 need a data management system. In contrast to our previous work, we extended the standard way when using RCE (which is storing everything locally on the machine of the workflow host) by using a data management system. We used shepard as a concrete implementation.

Shepard[1] (storage for heterogeneous product and research data) is a centralized data management system [13]. Shepard is designed to be the central place to store, explore and find data. Users and machines can access shepard though a standardized Representational State Transfer (REST) Application Programming Interface (API) as well as a web frontend. This allows continuous data acquisition and analysis across diverse processes of a product lifecycle and thus provides an ideal basis for the usage with virtual products, as shown by Krebs et al. [14].

Under the hood, shepard is designed to combine data from different data sources. Time series are stored in an internal InfluxDB. InfluxDB is a database that is specialized in storing time series data and is also capable of performing basic normalization and filtering functions on the data. Binary files are stored in an internal MongoDB via GridFS. Structured data (which is basically just JSON) is stored in the same MongoDB, allowing the usage of the MongoDB search functionality for documents. In this way, MongoDB can be used for two different purposes and there is no need to maintain two different databases without loosing performance or usability. Metadata and shepard-internal objects are stored in an internal Neo4j graph database. The graph database was chosen because shepard works mainly with data points and relationships between them, and Neo4j makes this very easy. The use of multiple databases while making the system more complex to set up and maintain, improves overall performance and usability by eliminating the need to find a single solution that meets all requirements sufficiently well. Instead, specialized tools can be used that excel in their respective fields. The individual parts of shepard and an exemplary outlined ecosystem can be seen in Figure 5.

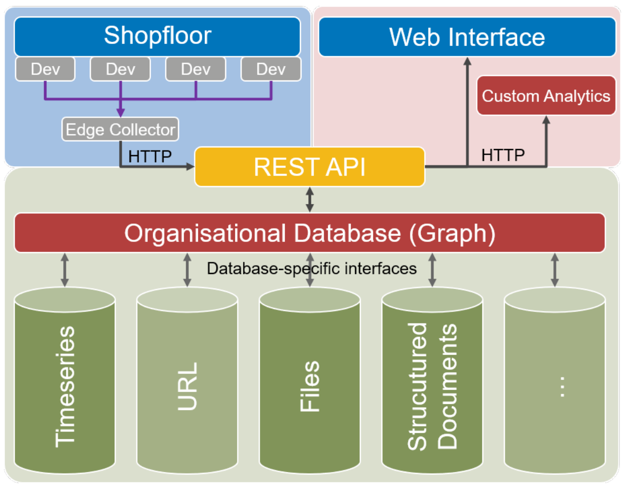

**FIG 5. Shepard architecture and ecosystem.**

[1]https://gitlab.com/dlr-shepard

In addition to the standardized REST API, shepard also provides a generic data model to store arbitrary data in a uniform way. This data model consists of collections in the first level and data objects below. A collection can contain data objects. The data objects in turn can be interconnected with other data objects of the same collection vertically as parent-child or horizontally as predecessor-successor. This enables the modeling of a wide range of configurations and processes in shepard.
However, collections and data objects themselves do not hold any data. Instead, there are containers that store data of different data types, such as time series, binary files, and structured data, utilizing the respective specialized databases. The actual payload data can be referenced by data objects using so-called reference objects. This separation between the actual data layer and the metadata layer allows for more flexibility in data collection and use. An example of this data model in use can be seen in Figure 6.

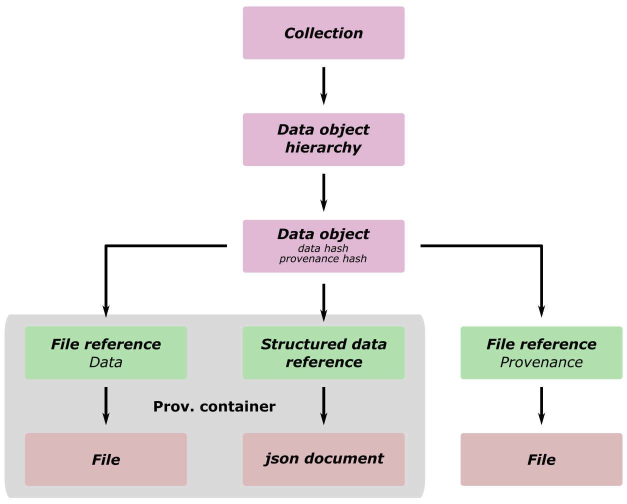

FIG 6. Shepard data structure for the VPH.

We structured the data of the VPH process in shepard in the following way: A top level collection according to each workflow run, a hierarchy of data objects according to the different disciplines and the output structure of each tool, references to the respective files (see Figure 6). The last data object in this hierarchy and its references represents a provenance container and contains additionally provenance and data hash as attributes. Each independent workflow run gets a unique uuid assigned, which is used to name the collection and therefore allow a clear separation of data between the different runs. If there are iterations within the workflow (in the VPH use case for example to converge a fluid structure interaction simulation) data objects are used to represent each run of the iteration separately and are connected via predecessor-successor relation. We created a RCE component based on the shepard python API called RCE2Shepard which can be seamlessly integrated into the VPH workflow by any partner. It uses the provenance container generated by the workflow tools to store the output of each tool according to the defined data structure. An exemplary cutout of a test run can be seen in Figure 7.

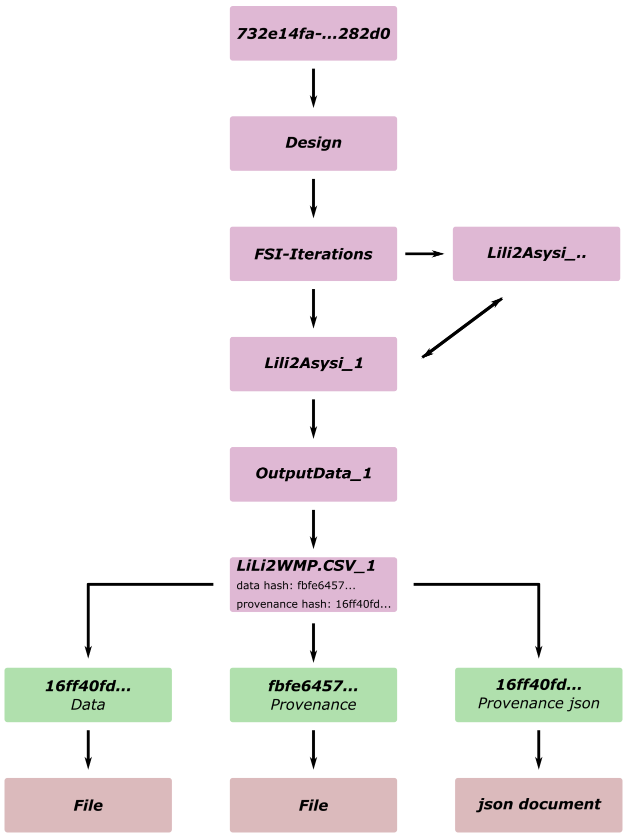

FIG 7. Exemplary data structure of a workflow output savd in shepard.

## 4. PROVENANCE APPLICATION

To review data (for virtual ertification), the whole provenance back to the datas primary sources should be taken into account. Within the architecture shown in chapter 2 and 3 the provenance for each file is stored in shepard. To make use of the provenance, it need to be aggregated depending on the use case. In this study we used shepard and applied its search functionality to look at three different use cases:

- Find a provenance container based on its id
- Find provenance container based on provenance attributes
- Trace an artefact to its primary sources

Each use case was implemented with a Jupyter notebook using the shepard python client libraries. In the following pseudo code examples, we simplified the library usage for simplicity.
Based on the data object architecture in shepard (see chapter 3), we used the following approach to get the provenance container (provenance and data) from a data object.

**Algorithm 1** Function getP4DO to get the provenance for a given data object

```
1: $dobj \leftarrow < dataobject >$
2: $sr \leftarrow getStructDataRef(dobj)$
3: $\underline{prov} \leftarrow getPayload(sr)$
```

**Algorithm 2** Function getPC4DO to get a provenance container based o an already given data object

```
1: $dobj \leftarrow < dataobject >$
2: $\underline{prov} \leftarrow$getP4DO($dobj$)
3: $datahash \leftarrow getHashAttribute(dobj)$
4: $frs \leftarrow getFileReference(dobj)$
5: while $fr$ in $frs$ do
6:     $tmp \leftarrow getFile(dobj, fr)$
7:     if $calculateHash(tmp) == datahash$ then
8:         $\underline{data} \leftarrow tmp$
9:         break
10:    end if
11: end while
```

***prov*** and ***data*** form the (logical) provenance container for the given hash.

## 4.1. Find a provenance container based on id

Data objects can be searched in shepard via their attributes. Each leaf data object stores both data hash and provenance hash (see chapter 3). The provenance hash is unique for the whole provenance container and therefore, the following approach can be used:

**Algorithm 3** Find provenance container for given id

```
1: $container\_id \leftarrow < containerid >$
2: $dobj \leftarrow doForPHash(container\_id)$
3: $pc \leftarrow getPC4DO(dobj)$
```

## 4.2. Find provenance container based on provenance attributes

With shepard, it is possible to search provenance based on a JSONPath [15] expression for a field within the JSON representation of the provenance model (see chapter 2).
We used the following approach:

**Algorithm 4** Find provenance container for attributes

```
1: $path \leftarrow < jsonpath >$
2: $value \leftarrow < value >$
3: $dobj \leftarrow doForSearch(path, value)$
4: $pc \leftarrow getPC4DO(dobj)$
```

## 4.3. Trace an artefact to its primary sources

Used entities are in a provenance container only referenced by the underlying provenance container hash and are not included directly. These references need to be followed to trace all used entities along the whole entity-activity graph to finally reach the primary sources.
Provenance over multiple entities forms a directed acyclic graph. We generated a representation of the graph in memory for further analysis by an recursive depth-first approach. We did not used a specific data structure like graph data structures but used the data object layout described in chapter 3 (performance was acceptable with the data volume observed). The following algorithm was used to build the graph:

**Algorithm 5** Generate the parent child relationship for the provenance graph.

```
1: $nodes \leftarrow []$
2: $current \leftarrow < containerid >$
3: function TRACE($id$)
4:     $dobj \leftarrow doForPHash(current)$
5:     $prov \leftarrow getP4DO(dobj)$
6:     $nodes[current] \leftarrow \{'id' : current, 'used' : []\}$
7:     for $u$ in $prov.used$ do
8:         $nodes[current].used += u$
9:         if $u$ not in $nodes$ then
10:            TRACE($u$)
11:        end if
12:    end for
13: end function
```

The result of Algorithm 5 is the graph from a given provenance container in shepard to each of its primary sources via all used entities including any intermediate steps. Line 6 is a simplification: More information from the provenance can be integrated into the graph depending on the analysis, which should be performed.

## 5. CONCLUSION

In this work we demonstrated our approach of combining a collaborative multi-stakeholder, multidisciplinary workflow, provenance and a data management system within the VPH. The VPH process takes place within a common source scenario, in which data and functionality is shared between stakeholders but no sharing of tools or source code takes place. We build upon our previous work [1] by extending the common source and provenance container concept and combine it with shepard (a data management system) to fulfill the identified functional requirements: Persistence, data accessibility, data consistency & traceability, attributability and modification detectability. The non-functional requirements were satisfied as before with the usage of RCE as the workflow orchestration tool. Attributability, data consistency & traceability and modification detectability were satisfied with the help of provenance and provenance containers. We used shepard as a data management system for data persistence, data accessibility and performing the analysis of provenance

information for tracing. The data structure we have chosen within shepard allowed easy persistence of the data and easy access for all project partners. On the other hand it implies no constraints on the data itself which is a strict requirement if data need to be available for decades (like for virtual certification). We showed our algorithmic approach for using shepard and its search capabilities for tracing data through the workflow back to its primary sources based on the recorded provenance.
The combination of common source, provenance & provenance container and shepard successfully satisfied the requirements we identified for the use cases. With the approach shown in this study we are confident to deliver data which can be used for virtual certification tasks.

## 6. ACKNOWLEDGEMENTS

The Virtual Product House start-up project is funded by the German federal state of Bremen and the European Regional Development Fund (ERDF).

**Contact address:**

frank.dressel@dlr.de

## References

[1] F. Dressel and A. Doko. Common source & provenance at virtual product house. In *Deutscher Luft- und Raumfahrtkongress (DLRK)*, Bremen, 09 2021. DOI: 10.25967/550061.

[2] F. Lange, A. S. Zakrzewski, M. Rädel, R. W. Hollmann, and K. Risse. Digital multidisciplinary design process for moveables at virtual product house. In *Deutscher Luft- und Raumfahrtkongress (DLRK)*, Bremen, 09 2021. DOI: 10.25967/550048.

[3] M. Rädel, D. P. P. Delisle, D. Bertling, R. Hein, and T. Wille. Towards robustness assessment in virtual testing - manufacturing influences by simulation-based methods in the virtual product house. In *Deutscher Luft- und Raumfahrtkongress (DLRK)*, Bremen, 09 2021.

[4] R. W. Hollmann, A. Schäfer, O. Bertram, and M. Rädel. Virtual testing of multifunctional moveable actuation systems at virtual product house. In *Deutscher Luft- und Raumfahrtkongress (DLRK)*, Bremen, 09 2021.

[5] F. Lange, A. S. Zakrzewski, J. Wild, R. W. Hollmann, and M. Rädel. Multidisciplinary and multifidelity end-to-end wing design process at dlr virtual product house. In *Deutscher Luft- und Raumfahrtkongress 2022*, Dresden, 09 2022.

[6] S. Freund and F. Franzoni. Automated liquid hydrogen tank design optimization using filament winding simulation and subsequent comparison with aluminium vessels. In *Deutscher Luft- und Raumfahrtkongress 2022*, Dresden, 09 2022.

[7] B. Boden, J. Flink, N. Först, R. Mischke, K. Schaffert, A. Weinert, A. Wohlan, and A. Schreiber. Rce: An integration environment for engineering and science. *SoftwareX*, 15:100759, 2021. DOI: https://doi.org/10.1016/j.softx.2021.100759.

[8] J. Flink, R. Mischke, K. Schaffert, D. Schneider, and A. Weinert. Orchestrating tool chains for model-based systems engineering with rce. In *2022 IEEE Aerospace Conference*, pages 1–9, 2022. DOI: 10.1109/AERO53065.2022.9843838.

[9] L. Moreau, P. Groth, S. Miles, J. Vázquez-Salceda, J. Ibbotson, S. Jiang, S. Munroe, O. F. Rana, A. Schreiber, V. Tan, and L. Z. Varga. The provenance of electronic data. *Commun. ACM*, 51(4):52–58, 2008. DOI: 10.1145/1330311.1330323.

[10] Prov-dm: The prov data model. `https://www.w3.org/TR/prov-dm/`, 2013. Accessed: 2022-09-13.

[11] European Union. Regulation (eu) 748/2012 of the european commission. *OJ L*, 224:1–85, 2012.

[12] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, et al. The fair guiding principles for scientific data management and stewardship. *Scientific data*, 3, 2016.

[13] T. Haase, D. R. Glück, P. Kaufmann, and M. Willmeroth. shepard - storage for heterogeneous product and research data, July 2021. DOI: 10.5281/zenodo.5091604.

[14] F. Krebs, M. Willmeroth, T. Haase, P. Kaufmann, R. Glück, D. Deden, L. Brandt, and M. Mayer. Systematische erfassung, verwaltung und nutzung von daten aus experimenten. In *Deutscher Luft- und Raumfahrtkongress (DLRK)*. Deutsche Gesellschaft für Luft- und Raumfahrt - Lilienthal-Oberth e.V., 2021. DOI: 10.25967/550315.

[15] Json path (jsonpath). `https://datatracker.ietf.org/wg/jsonpath/about/`. Accessed: 2022-09-06.