TOWARDS A CLOSED-LOOP DATA COLLECTION AND PROCESSING ECOSYSTEM

T. Haase*, R. Glück*, D. Görick*, P. Kaufmann*, F. Krebs*, M. Mayer*
* German Aerospace Center (DLR), Institute of Structures and Design, Am Technologiezentrum 4, 86159 Augsburg, Germany

Abstract

The German Aerospace Center (DLR) in Augsburg demonstrates the use of the shepard data management system using the example of robot-controlled production of an aircraft upper-shell with thermoplastic tape-laying processes. In the process, measured data from production and quality assurance is automatically gathered, interconnected and stored centrally. The data can then be searched and evaluated in shepard or analyzed in external applications.

Keywords
Research Data Management, Data Analysis, Quality Assurance, Automated Fiber Placement

NOMENCLATURE

API	Application Programming Interfaces
CAD	Computer-Aided Design
CI/CD	Continuous Integration / Continuous Delivery
CSV	Comma-Separated Values
DRG	Data Reference Generator
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
JWT	JSON Web Token
KRC	Kuka Robot Control
MFFD	Multi Functional Fuselage Demonstrator
MTLH	Multi-Tow Laying Head
OPC UA	Open Platform Communications Unified Architecture
PLC	Programmable Logic Controller
REST	Representational State Transfer
sTC	shepard Timeseries Collector
T-AFP	Thermoplastic in-situ Automated Fiber Placement
TCP	Tool Center Point
TIFF	Tagged Image File Format
TPS	Tape Placement Sensor

1. MOTIVATION

Modern industrial production processes as well as research experiments consist of complex sub-processes distributed over multiple systems and carried out by different actors [1]. This results in a complex scenario where it is difficult to know which data set belongs to a particular produced part. These difficulties become apparent when the data is to be analyzed to gain knowledge about a part. Some data are in the form of time series and can only be assigned over time, which may be unknown for that part. Other data is in files, but is attached to either a sub component or the entire product, so the data must be extracted and collected from one or more files. Sometimes the data is even stored across different machines. All these aspects increase the effort required to review and analyze the data and also contradict the FAIR principles [2]. To address these issues, we developed a centralized data management solution called storage for heterogeneous product and research data (shepard), formerly known as integrated data management system (iDMS) [3].
Shepard is already used to collect, store, and manage data in various scenarios [4–7]. With a central storage solution in place it is now easy to collect data across large projects. This paper describes different approaches to analyzing the data stored in shepard, illustrated by the example of automated production of the upper shell of a single aisle aircraft. This allows correlations to be revealed between events during the process and variations in the quality of finished parts, as well as estimations of part quality based on data from production.

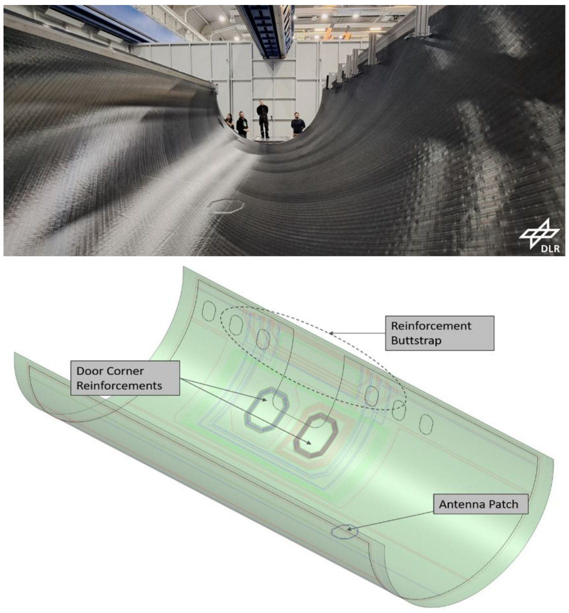


FIG 1. 8 x 4 m thermoplastic skin for the upper shell of a single aisle aircraft (top). Overview of the Computer-Aided Design (CAD) design with notable features (bottom).

2. USE CASE

In the scope of the Multi Functional Fuselage Demonstrator (MFFD) project funded by the EU and embedded in the joint undertaking Clean Sky 2, the upper shell of a single aisle aircraft was build at the Center for Lightweight Production Technology in Augsburg. The gist was to manufacture the shell using only thermoplastic materials to facilitate a dustless assembly. Because thermoplastic materials are malleable at certain high temperatures two components can be joined by welding. Dust-sensitive subassemblies pre-equipped with system and cabin elements can be incorporated in the dustless welding assembly line using this thermoplastic approach [8].

The key advantage of using thermoplastic materials in skin manufacturing is that no further process step is needed after the tape has been applied. In the traditional process, however, another process step is required by means of covering the skin with a vacuum bagging and putting the skin into an autoclave. The pressure and additional resin smooths minor defects. Skipping the autoclave saves time and reduces major costs. But to retain these advantages the tape placement has to be precise and flawless. Therefore, monitoring parameters and determining quality is a key enabler for thermoplastic in-situ automated fiber placement (T-AFP) [9].

There are four main steps involved to build the shell: 1. skin placement and steps 2.-4. which use different welding technologies to integrate stringers (2.), frames (3.) and cleats (4.) for stiffening the shell [10]. This paper focuses on the skin placement production step and its data collection and processing.

The skin was manufactured in full-scale with a length of 8 m and a diameter of 4 m (Fig 1 top). The buildup

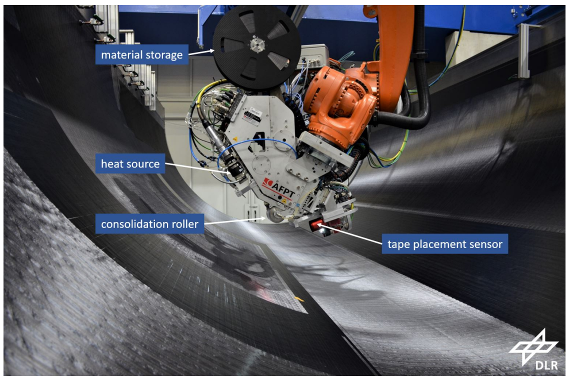


FIG 2. MTLH descriptions

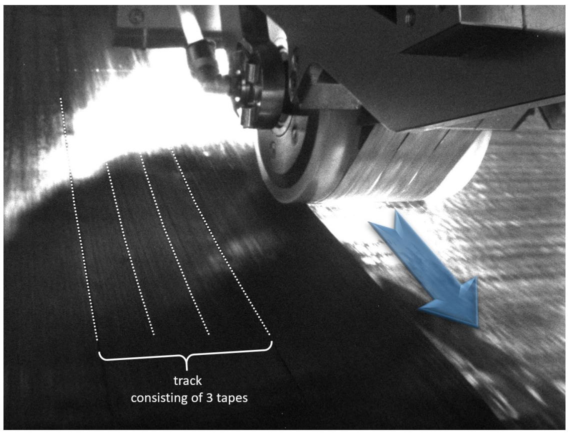


FIG 3. Process camera during layup and highlighting of one track and its tapes

of the skin comprises areas with different thicknesses to reinforce the door corners, buttstrap and antenna patch. The thickness varies from 1.6 mm to 12.5 mm (Fig 1 bottom). The multi-tow laying head (MTLH) (Fig 2) is capable of depositing three tapes in parallel (Fig 3). A path planning software divides each layer into tracks of 3 tapes and delivers the robot programs which are then executed by a ceiling mounted KUKA KR270 R2700 on a rail. The skin build up is divided into 53 ply groups, each consisting of one up to ten plies. The smallest unit is one track consisting of 3 tapes and can vary in length from about 0.3 m to 11.5 m. The tracks are adjacent to each other so that one ply can comprise 10 to 244 tracks.

The data sources of this particular use case can be divided into three categories: timeseries data, file based data and key-value pairs. The Kuka Robot Control (KRC), MTLH and tape placement sensor (TPS) provide timeseries data, whereas the TPS also supplies files. In addition, notes of the machine operator are available as well but have to be preprocessed to become machine-readable in form of key-value pairs.

The measurement principle of the TPS (cf. Fig 2) is based on laser triangulation. A line laser is deflected by a mirror onto the surface/track and the reflection is recorded by a camera. Every 2 mm a height profile of the just placed track is triggered by the robot and calculated by the camera. At the end of each track all height profiles are stored in a 16 bit Tagged Image

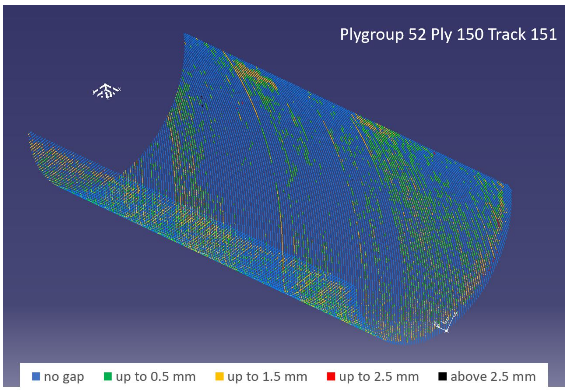


FIG 4. Gaps between tracks of one ply

File Format (TIFF) file. The corresponding recording positions with six degrees of freedom are retained in a Comma-Separated Values (CSV) file. Both files are essential to locate results in the part coordinate system, which enables superposition of part design and defects in CAD. To detect defects the 16 bit TIFF file is evaluated with computer vision algorithms, for more details refer to [11]. Defects are gaps and overlaps of different sizes between tracks and tapes; this paper focuses on gaps between tracks. Fig 4 depicts the gap results for the penultimate ply with full coverage of the 4 x 8 m long mould.
One major objective is to find correlations between different data sources which may lead to the ability to identify causality. If two or more data sources show a dependence on each other a selective parameter study can be conducted to improve the tape laying process and therefore the skin quality.

2.1. Setup

The system was set up so that most of the data can be collected automatically. A more elaborate description than the present one, referring also to shepard specific details, can be found in [4]. While the internal communication is mainly done via ProfiNet, the MTLH, the KRC as well as the programmable logic controller (PLC) of the robot cell provide data via Open Platform Communications Unified Architecture (OPC UA). These values are provided continuously over time, hence they are called time series data. To gather the data and send it to the local shepard instance, the OPC Router software[1] was used. A python script scans all OPC UA servers available and exports the discovered nodes into a CSV file. Concerning the use case the servers are the MTLH with nodes for laser power, tape temerature, tape cut, etc., the KRC with nodes for speed, temperatures, movement state, position etc. and the TPS with nodes for defect size, location, width, etc. Relevant nodes can be selected by hand and imported into the OPC Router configuration. In addition, the OPC Router was configured to send incoming data directly to shepard to be stored in a timeseries container. Contextualization in shepard via timeseries references can be done later on, thus no further configuration is necessary. The final configuration consists of 210 variables updated at up to 15 Hz. By default, the OPC Router forwards every single incoming value, resulting in a maximum of 3150 values per second. This is enough to keep the measurement computer that runs the OPC Router busy and to overload the server on which shepard is running. It became apparent that InfluxDB as well as the shepard backend together need more than 64 GB of memory to handle this specific workload. The load could be significantly reduced by combining the most recent values into a bulk and sending them to shepard once every few seconds.
When the machine operator starts the robot program, one ply group is manufactured automatically. To keep track of the overall process, a python script called Data Reference Generator (DRG) is subscribed to the KRC via OPC UA. Before each track is started, the robot notifies the DRG about the actual ply and track to be manufactured. The DRG knows the overall structure as well as the current state of the process. Therefore, it can create the necessary data objects in shepard accordingly. Also, the DRG knows when a process step begins and ends, and can therefore create the respective timeseries references in shepard.
The data collection of the TPS is started at each track beginning via the technology package EthernetKRL from KUKA. The TPS software is also connected to the DRG and receives the relevant data object and file container identifier via OPC UA to upload the measurement data to a file container linked to the track specific data object in shepard at the end of each track. The data comprises four CSV files and one 16 bit TIFF file.
Besides the automated gathering and uploading of data, there is another source of data that cannot be automatically processed and attached to a data object. This data source consists of manually written notes of the machine operator. These notes contain information about manual rework or maintenance of the MTLH like material storage or consolidation roller change. The information written in these notes are transferred as key-value pairs to a CSV which can be processed by a script to attach these attributes to each corresponding data object respectively track in shepard. This allows for fast filtering of data objects, for example to identify all tracks with a specific attribute of interest.

2.2. Resulting dataset

The resulting dataset contains everything that was collected during the experiments. This dataset is entirely stored in shepard and can be explored using the web frontend.
A well designed structure of collections, data objects, and references (see again [4] for details concerning shepard's data structure) is helpful when working with any unknown data. Our approach to such a structure is a hierarchical one. There is only one

[1] https://www.opc-router.de/

```
Collection
└─Layup
   ├─Ply Group 1
   │  ├─Ply 1
   │  │  ├─Track 1
   │  │  ├─Track 2
   │  │  ├─...
   │  │  └─Track n
   │  └─...
   └─...
```

FIG 5. Structure of the resulting dataset

container of each type, i.e. a file container, a timeseries container and a container for structured data. Multiple containers per type would be necessary to be able to set up different permissions for different data sets, but for this use case one container per type was sufficient. Furthermore, a single collection was created which includes the entire part to be assembled. The layup process is one step of several assembly steps. Therefore, a root data object called Layup was created which combines everything related to the experiments described above. The Layup data object is a parent object to other data objects representing different ply groups. The ply groups themselves are parents of the respective layers, which in turn are parents of the various tracks. Each layer and track is connected to its respective predecessors and successors via the corresponding relationship in shepard. Fig 5 shows the resulting structure.
Some data objects within this structure have relevant information stored in the form of attributes. Tracks, for example, are provided with the information if a material storage change was conducted or in which adjacent direction the predecessor track was placed, left or right. Furthermore, each record (timeseries, file, structured data) is linked to the corresponding data objects via the associated reference type. This is easy to do with atomic data such as files or structured data. Timeseries, on the other hand, by definition do not have a start or end time, so each timeseries reference has to define these timestamps. For example, all tracks refer to the same tape temperature timeseries but each timeseries reference is assigned a different start and stop time. With this reference approach, the payload data can be kept separate from the organizational data, and the need for duplicate data can be reduced. It is possible to link all track data objects to the same TPS calibration file stored in the file container, because the calibration routine was done only once prior to the tape laying start.
By using the TPS files stored in shepard, an evaluation pipeline for each track can be executed resulting in a file with coordinates and corresponding gap sizes. All files of one ply can be plotted as point cloud as Fig 6 visualizes. The depicted ply 4 will serve as concrete example to demonstrate some capabilities of shepard. Fig 6 is plotted by the CAD programm Catia V5. The gaps can be investigated by rotating and

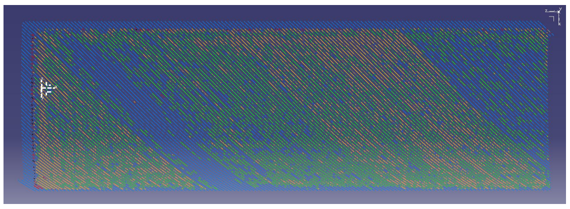


FIG 6. Gaps between tracks of ply 4 in plygroup 4

zooming in but the interpretation is limited because additional information, like compaction roller change has to be integrated manually. Shepard supports effective solutions to explore data in a more efficient way as described in the following sections.

3. TOOLS FOR EXPLORATORY DATA ANALYSIS

When it comes to working with uploaded data it is mandatory to have a tool for exploring data and getting a first impression of its shape and dimensionality. To fulfill this need of data exploration, shepard has its own tools for visualizing all kinds of data types.
The exploration process often starts with gaining an overview of the available data structures and payloads. While the default view of the shepard frontend allows for easy navigation and displays a lot of details in text form, a graphical representation of the entire project structure can be helpful in certain situations. Therefore, we have developed an interactive graph view that allows to get a quick overview of the data objects and their relations to each other (see Fig 7).

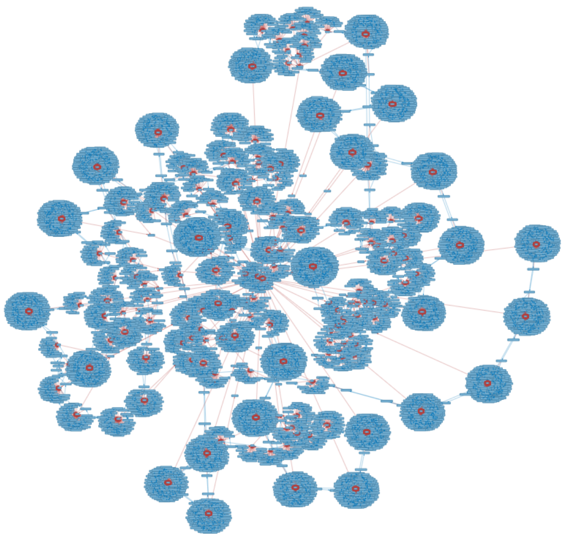


FIG 7. Interactive graph representation of the entire process structure with all plies and tracks.

This type of visualization was inspired by the graphical interface of the neo4j database. A graph shows the data objects of a collection, starting with the top-level objects without a parent. Each data object can be expanded to show its child objects. For large projects with many data objects, it may not be practical to display such a large number of data objects for detailed

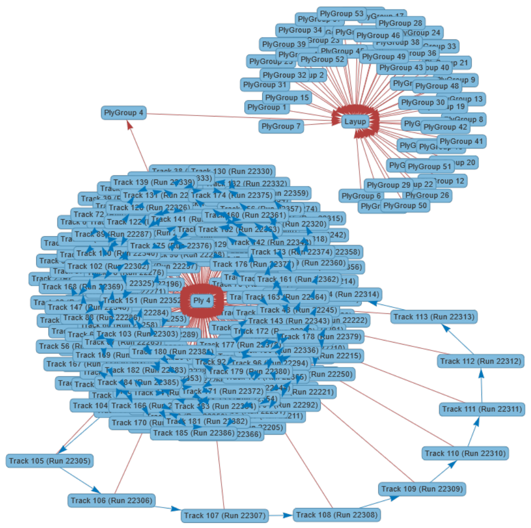


FIG 8. Interactive graph of a collapsed project structure. A single node (here ply 4) is shown in its expanded shape for a more detailed view.

interaction. Hence, the nodes can be individually expanded and collapsed so that separate branches of the project graph can be explored. One example of how a collapsed structure of a large project may look like can be seen in Fig 8. While red arrows represent parent-child relationships, blue arrows indicate predecessor-successor relationships. The graph view is built using the open source vis-network library[2] and is rendered entirely by the client's web browser at runtime.
Another important part of exploring data in the shepard frontend is the visualization of the referenced data. Six different reference types are used for storing different kinds of data. These reference types are called structured data-, file-, timeseries-, URI-, data object- and collection references. In addition to the aforementioned graph structure for visualizing data objects and collection data relationships, shepard provides different visualization options for the rest of the data types. While URI data consist of a web link and do not need complicated visualization, we developed a modal for structured data. This special designed editor modal uses the JSON format of the structured data to visualize the data in a fast and efficient way. By integrating an advanced JSON editor[3] even lager JSON structures can be explored easily.
Data of experiments are often recorded and uploaded in the timeseries data format. Timeseries data have the advantage to be already in an easy accessible format since time can be used as x-axis value and the measured data can be used as y-axis value. The shepard timeseries plotting modal makes use of this characteristics and uploaded data can immediately be visualized. The visualized data appear in an separate modal which offers the option to save the plot to the local system in order to use the visualization in presentations or in scientific work. An exemplary visualization of a tape temperature which was measured during a production process in the MFFD project can be seen in Fig 9.

[2]https://github.com/visjs/vis-network
[3]https://github.com/josdejong/jsoneditor

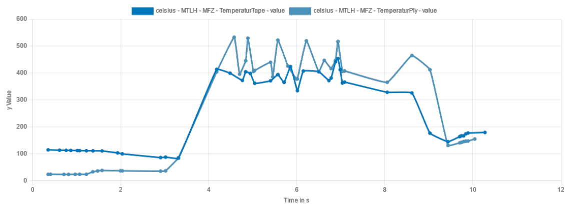
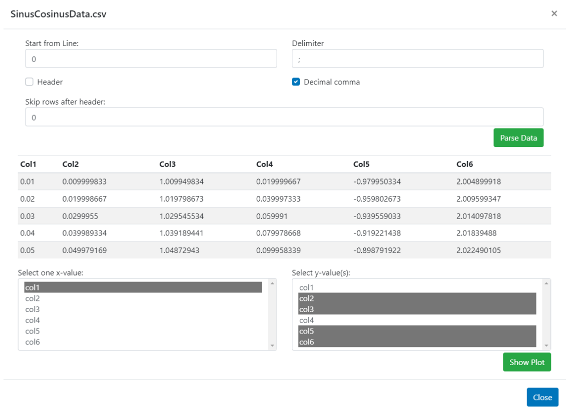


FIG 9. Timeseries data visualization. Tape temperature during a production process.

When it comes to file reference data, shepard is able to differentiate between images, JSON, plain text and CSV data. Text files and images are visualized by using a blank modal and simply displaying the content of the data package in this modal. For JSON files, the aforementioned JSON editor is reused. In contrast to these, visualizing CSV data is a more complicated task. For the investigation of CSV data, shepard provides the option to open the data packages as plane text (similar to the visualization of figures and text data), as a table or in a modal which enables the user to create a visualization in form of a plot. CSV files can have headers in different shapes, use different kinds of delimiters and store values with decimal commas or decimal points. Due to the huge range of possible CSV data shapes the plotting modal in shepard is able to query data-shape-related information from the user in order to be able to parse the CSV data correctly (see Fig 10). This modal allows to define a row in which to start the parsing, to set a specific delimiter, to define if there is a header in the data file and to define its shape. The data is parsed accordingly to the information provided by the user and shepard automatically gives names to all identified columns ('Col1', 'Col2', ... 'Col n') if the CSV file has no header.
The result of the parsing process is then displayed in a table below the parsing options. As soon as different columns are found, the user can select one column to be used as x-axis values and one or multiple columns for y-axis values. Finally the data can be visualized in the previously mentioned visualization modal (see


SinusCosinusData.csv

Start from Line: 0

Delimiter: ;

☐ Header

✓ Decimal comma

Skip rows after header: 0

Col1	Col2	Col3	Col4	Col5	Col6
0.01	0.009999833	1.009949834	0.019999667	-0.979950334	2.004899918
0.02	0.019998667	1.019798673	0.039997333	-0.959802673	2.009599347
0.03	0.0299955	1.029545534	0.059991	-0.939559033	2.014097818
0.04	0.039989334	1.039189441	0.079978668	-0.919221438	2.01839488
0.05	0.049979169	1.04872943	0.099958339	-0.898791922	2.022490105

Select one x-value: [✓ col1] col2 col3 col4 col5 col6

Select y-value(s): col1 [✓ col2] [✓ col3] col4 [✓ col5] [✓ col6]

FIG 10. Modal for parsing and visualizing CSV data.

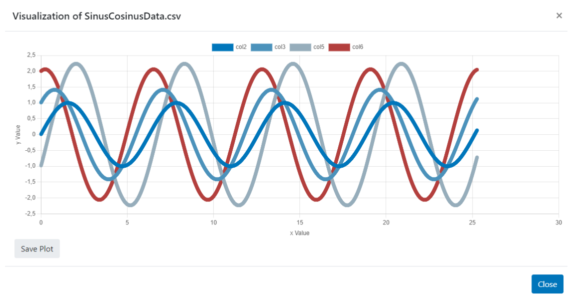


FIG 11. Visualization of exemplary CSV data.

Fig 11). In contrast to Fig 9 this figure shows a screenshot of the modal and not the exported visualization plot. The modal has a save button on its lower left side and a button to close the visualization on its lower right side.

4. ADVANCED DATA ANALYSIS

The web frontend provides easy access to the available data and allows exploration without any additional tools or knowledge. However, a more detailed analysis is not within the scope of the frontend, as more specific tools show a better performance for this kind of task. This includes in-depth analysis of existing timeseries, evaluation of uploaded images or dashboarding applications for a live preview of the available data.
Other tools can access the data either directly via the REST API or more easily via provided software libraries. The source code of the shepard backend is automatically analyzed to create an precise OpenAPI specification of the provided REST endpoints [12]. This specification can be used as documentation, but also to automatically generate clients that are compatible with the specified server. We included a build step into our Continuous Integration/Continuous Delivery (CI/CD) pipeline that makes use of the OpenAPI Generator[4] to generate various clients for Python, Java, Typescript, and C++. These client libraries are automatically uploaded to Gitlab and provided there as part of the shepard repository.
The shepard API is protected using OpenID Connect[5] which builds on top of OAuth 2.0[6] [13]. Using modern web techniques allows easy identification of the user. For example, the shepard frontend uses OpenID Connect to authenticate the user. While this works well in a web browser environment, OpenID Connect cannot be easily utilized by machines or scripts. Therefore, shepard offers an alternative authentication function with static API keys. These API keys are JSON Web Tokens (JWT) signed by the shepard backend [14]. From the moment an API key is created up to its deletion, the key can be used to authenticate the user. This makes API keys the ideal solution for analytic applications such as those described next.

[4] https://openapi-generator.tech/
[5] https://openid.net/developers/how-connect-works/
[6] https://oauth.net/2/

With these client libraries it is easy to develop small analysis scripts or even large applications for individual processes. For example, Jupyter Notebooks[7] can be used to display and compare timeseries data.
The following example extends the previous setup to display the measured temperature of the tape laying head. First, the web frontend was used to identify a ply of interest. For this example, a large ply was chosen that covers much of the shell. After the selection process a Jupyter notebook was set up to retrieve all tracks for that ply. The shepard API allows selection of data objects by parent id so the data assessing can easily be done with a single query. There is only one timeseries reference per track which holds all timeseries data. The timeseries payload data can be filtered in a way that only selected timeseries are included in the response. This helps to reduce the memory footprint on the client side. The timeseries in question were previously chosen via the frontend. In this case, we select the coordinates of the Tool Center Point (TCP) of the robot and the measured temperature of the MTLH. Since these values were measured independently, resampling is required to get matching values at a single point in time. The InfluxDB already provides this functionality, which in turn is also provided by the shepard API. The temperature values can be plotted in relation to the TCP values using the Plotly Express library[8]. The resulting plot can be seen in Fig 12. In this plot it can be seen that the temperature is relatively steady in the main area of the measurement but also single errors are visible where the process did not work properly (Fig 12, diagonal violet line of dots). In addition, the start-up and shut-down phases of can be easily identified as their temperature is lower than during the main phases.

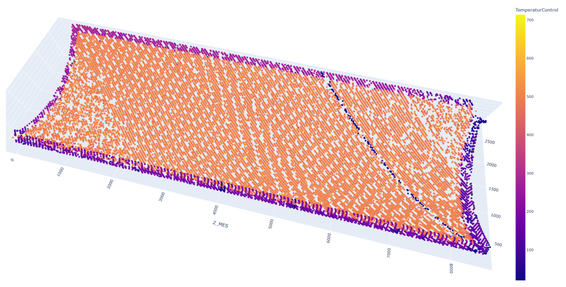
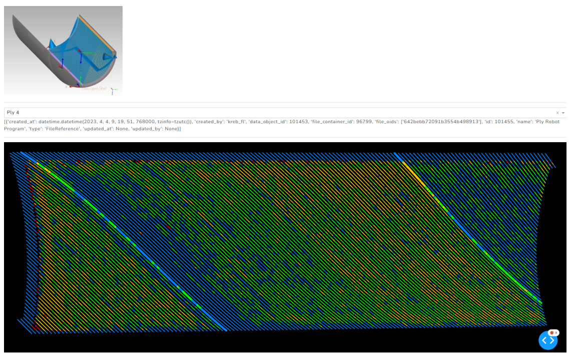


FIG 12. Temperature graph showing the measured temperature of the placed tape in relation to the current TCP position.

A different approach builds upon Dash Open Source[9]. Similar to Jupyter, Dash allows the creation of interactive plots. While Jupyter combines the source code with the respective output in form of a notebook, Dash creates an interactive web page and hides the source code from the user. It is also possible to embed input forms on the web page so that the user can interact with the data and navigate through structures. Thus,

[7] https://jupyter.org/
[8] https://plotly.com/python/plotly-express/
[9] https://github.com/plotly/dash



```
[{'created_at': datetime.datetime(2023, 4, 4, 9, 19, 51, 768000, tzinfo=tzutc()), 'created_by': 'kreb_fl', 'data_object_id': 101453, 'file_container_id': 96799, 'file_oids': ['642bebb72091b3554b498913'], 'id': 101455, 'name': 'Ply Robot Program', 'type': 'FileReference', 'updated_at': None, 'updated_by': None}]
```

FIG 13. Dash Application displaying gaps and roll changes

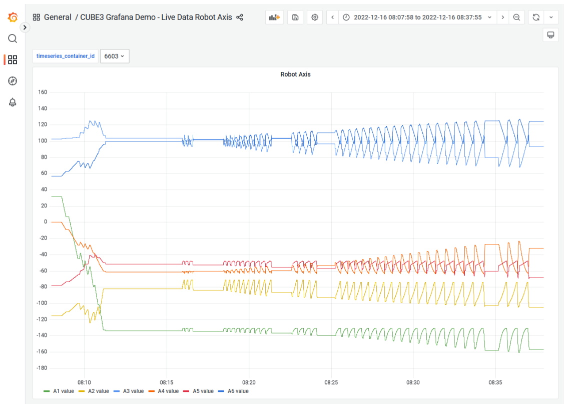


FIG 14. Grafana Dashboard displaying robot movements

simple analysis tools can be built easily. The example shown in Fig 13 displays the measured gaps between different tracks. Because of differences in the tape laying quality, it was suspected that the change of the compaction roller influenced the formation of gaps between tracks. What was previously difficult to visualize can now be easily done in a dash app. The change of the compaction roller is stored as an attribute of the track. This information can be displayed in the form of highlights in the point cloud.

In contrast to many CAD programs as depicted in Fig 6, approaches such as Dash and Jupyter are generally considered more flexible and, at least for computer and data scientists, more openly accessible. Thus, it is easily possible to take various types of data into account. The Jupyter example (Fig 12) relies entirely on time series data while the Dash example (Fig 13) uses annotations to enrich the information in point clouds from files.

Although shepard receives incoming data in batches rather than instantaneously, the delay is still short enough for dashboards to display live data. A low-code option for creating such a dashboard is Grafana[10] in combination with the Infinity Datasource plugin[11]. The Infinity Datasource plugin can be configured to fetch data directly from shepard via the provided REST API. Configured accordingly, the HTTP requests contain both necessary headers such as the API key, and query parameters for the filtering of the timeseries. This way, only necessary data is fetched and processed, which leads to an increase in performance. The setup can be configured to fetch the data at specific intervals so that sufficient up-to-date data is always visible. An example of this setup can be seen in Fig 14. It shows the most recent robot movements as axis position in degrees.

5. CONCLUSION

This paper shows some approaches how data in shepard can be analysed during and after the process. This was done using a real-life application of an automated production process. It was made clear how state-of-the-art systems can be connected to shepard in order to efficiently collect data of all kinds. Data processing was made as seamless and automated as possible in order to be traceable and error resistant. It became apparent how important it is to have a well-designed data structure in shepard so that subsequent analysis steps can easily and intuitively access data of all scopes. The shepard frontend provides generic tools for reviewing data, but sophisticated data science tools are needed to address specific questions. Shepard provides a convenient way for these tools to access the data in question through its API. This way we were able to show the influence of the change of the compaction roller on the formation of gaps and overlaps between the tracks and thus on the quality of the finished part.

We have also identified some aspects that can be improved. First, there is the collection of time series data. The more data that is to be written to shepard, the more important it is to aggregate multiple data points into one request to reduce the load on both shepard and the network along the way. The OPC Router works event based, therefore it is not impossible but cumbersome to configure that the data is sent in batches. At the Center for Lightweight Production Technology in Augsburg, we have developed an application that is tailored exactly to this use case. The shepard Timeseries Collector (sTC)[12] can be used as an edge device to collect data from different sources and send it in batches to shepard. We plan to test this software in a larger use case in the future. Furthermore, the DRG can be improved. At the moment, the DRG is a command line tool that runs in the background. In the future, this could be extended with a graphical user interface so that the operator can directly see the current state as well as manually create annotations or influence the process. This could replace the handwritten notes including the resulting parse and import process.

Contact address:

tobias.haase@dlr.de

[10]https://grafana.com/
[11]https://sriramajeyam.com/grafana-infinity-datasource/
[12]https://gitlab.com/dlr-shepard/shepard-timeseries-collector

References

[1] C. Frommel, F. Krebs, T. Haase, M. Vistein, A. Schuster, L. Larsen, M. Körber, M. Malecha, and M. Kupke. Automated manufacturing of large composites utilizing a process orchestration system. Procedia Manufacturing, 51:470–477, 2020. ISSN: 23519789. DOI: 10.1016/j.promfg.2020.10.066.

[2] Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J.G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Jaap Heringa, Peter A.C 't Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J. Lusher, Maryann E. Martone, Albert Mons, Abel L. Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna-Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A. Swertz, Mark Thompson, Johan van der Lei, Erik van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons. The FAIR Guiding Principles for scientific data management and stewardship. Scientific Data, 3(1):160018, Mar. 2016. ISSN: 2052-4463. DOI: 10.1038/sdata.2016.18.

[3] Tobias Haase, Roland Dr. Glück, Patrick Kaufmann, and Mark Willmeroth. shepard - storage for heterogeneous product and research data, July 2021. Language: en. DOI: 10.5281/ZENODO.5091604, https://zenodo.org/record/5091604.

[4] 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. page 8 pages, 2021. Publisher: Deutsche Gesellschaft für Luft- und Raumfahrt - Lilienthal-Oberth e.V. DOI: 10.25967/550315.

[5] F. Dressel, M. Rädel, A. Weinert, M. Struck, T. Haase, and M. Otten. Common Source & Provenance at Virtual Product House: Integration with a Data Management System. page 7 pages, 2022. Publisher: Deutsche Gesellschaft für Luft- und Raumfahrt - Lilienthal-Oberth e.V. DOI: 10.25967/570066.

[6] A. Schuster, M. Mayer, M. Willmeroth, L. Brandt, and M. Kupke. Inline Quality Control for Thermoplastic Automated Fibre Placement. Procedia Manufacturing, 51:505–511, 2020. ISSN: 23519789. DOI: 10.1016/j.promfg.2020.10.071.

[7] Alfons Schuster, Monika Mayer, Lars Brandt, Dominik Deden, Florian Krebs, and Michael Kupke. Inline Quality Control for Thermoplastic Automated Fiber Placement by 3D Profilometry. In Proceedings SE Conference 21 Baden / Zürich, Sept. 2021.

[8] Dominik Deden, Lars Brandt, Olivia Hellbach, and Frederic Fischer. Upscaling of in-situ Automated Fiber Placement with LM-PAEK - From Panel to Fuselage. In ECCM 2022 - Proceedings of the 20th European Conference on Composite Materials: Composites Meet Sustainability, June 2022.

[9] M. Mayer, A. Schuster, L. Brandt, D. Deden, and F. Fischer. Integral quality assurance method for a CFRP aircraft fuselage skin: Gap and overlap measurement for thermoplastic AFP. Lecture Notes in Mechanical Engineering, accepted, to appear. 32nd International Conference on Flexible Automation and Intelligent Manufacturing (FAIM2023).

[10] Frederic Fischer, Manuel Endraß, Dominik Deden, Lars Brandt, Manuel Engelschall, Philipp Gänswürger, Monika Mayer, Michael Vistein, Manfred Schönheits, Alfons Schuster, Stefan Jarka, Simon Bauer, Olivia Hellbach, Lars-Christian Larsen, Michael Kupke, and Stefan Ferstl. How to Produce a Thermoplastic Fuselage. In Axel Herrmann, editor, ITHEC 2022, 6th International Conference & Exhibition on Thermoplastic Composites, volume 6 of International Conference & Exhibition on Thermoplastic Composites. M3B GmbH, Bremen, Germany, Oct. 2022.

[11] M. Mayer, A. Schuster, L. Brandt, D. Deden, F. Fischer, D. Schmorell, and M. Vistein. Quality Assured Aircraft Fuselage Production: Data Evaluation of a Quality Control Sensor for Thermoplastic Automated Fiber Placement. page 5 pages, 2022. Publisher: Deutsche Gesellschaft für Luft- und Raumfahrt - Lilienthal-Oberth e.V. DOI: 10.25967/570129.

[12] Darrel Miller, Jeremy Whitlock, Marsh Gardiner, Mike Ralphson, Ron Ratovsky, and Uri Sarid. OpenAPI Specification V3.1.0, Feb. 2021. Publication Title: OpenAPI Specification v3.1.0. https://spec.openapis.org/oas/v3.1.0.

[13] Dick Hardt. The OAuth 2.0 Authorization Framework, Oct. 2012. Issue: 6749 Num Pages: 76 Series: Request for Comments Published: RFC 6749. DOI: 10.17487/RFC6749, https://www.rfc-editor.org/info/rfc6749.

[14] Michael B. Jones, John Bradley, and Nat Sakimura. JSON Web Token (JWT), May 2015. Issue: 7519 Num Pages: 30 Series: Request for Comments Published: RFC 7519. DOI: 10.17487/RFC7519, https://www.rfc-editor.org/info/rfc7519.