This is the author's version (manuscript) of the work that was accepted for publication in AIAA SCITECH 2024 Forum.
The Version of Record is available online at https://arc.aiaa.org/doi/10.2514/6.2024-0482.

Integration of large data based on HDF5 in a collaborative and multidisciplinary design environment, Part A: Methodology and Implementation

M. Bröcker*, M. Siggel†, S. Reitenbach‡
German Aerospace Center (DLR), Cologne, Germany, D-51147

Collaborative processes are of increasing importance in many engineering disciplines, particularly in design and simulation. However, developing an integrative, consistent software architecture for collaborative applications remains a significant challenge in industry, research, and education teams. As digitalization progresses, the efficient processing of ever larger and more complex datasets becomes increasingly important. While various dedicated software platforms support the exchange and traceability of data as well as the interoperability of tools and workflows, the efficient integration of large amounts of data into such architectures is still a significant issue. This paper presents an innovative approach to integrate large datasets utilizing the HDF5 data format into a collaborative software framework for aircraft engine design and simulation. The application of the methodology is demonstrated by a practical example from the field of structural mechanics in part B [1].

I. Nomenclature

CFD	=	Computational Fluid Dynamics
CGNS	=	CFD General Notation System
CSM	=	Computational Structural Mechanics
DLR	=	German Aerospace Center
GTlab	=	Gas Turbine Laboratory
GUI	=	Graphical User Interface
HDF5	=	Hierarchical Data Format

II. Introduction

The evolving complexity and interdisciplinary nature of modern engineering applications make collaborative and multidisciplinary processes increasingly indispensable. Combining expertise from different disciplines is essential to address many of today's engineering challenges [2, 3]. However, the exchange and processing of data and the communication between the individuals involved in these processes can easily become complex and confusing. Nevertheless, not only the application of heterogeneous design and simulation tools with different levels of detail makes data processing and ensuring consistency of data even more challenging, the trend towards generating ever larger and ever greater amounts of data increases the complexity [4].

Various software strategies are employed to overcome these challenges. The approach of a central data model allows the integration and coupling of tools and enables the regulated exchange of data. This is achieved by providing standardized data schemes and interfaces. Examples for a central data model include the Common Parametric Aircraft Configuration Schema (CPACS) [5] and the data model of the Gas Turbine Laboratory (GTlab) framework [6, 7]. Additionally, data provenance is a promising solution for providing and ensuring data consistency. Embedded into a collaborative architecture, data can be tracked throughout its lifecycle. Changes, including metadata about who modified what data and why, are noted. This allows the identification of affected data, that must be updated, and processes, that must be reevaluated [8].

*Research Associate, Institute of Propulsion Technology, Linder Hoehe, 51147 Cologne, Germany, marius.broecker@dlr.de
†Research Associate, Institute of Propulsion Technology, Linder Hoehe, 51147 Cologne, Germany, martin.siggel@dlr.de
‡Research Associate, Institute of Propulsion Technology, Linder Hoehe, 51147 Cologne, Germany, stanislaus.reitenbach@dlr.de

However, a critical problem arises when dealing with simulation workflows such as high fidelity Computational Fluid Dynamics (CFD) and Computational Structural Mechanics (CSM) analysis that are executed on clusters or outside the collaborative architecture. As outlined by Cui et al. such workflows may be considered as opaque black-box-transformations in which each output element may depend on any combination of input elements [9]. Consequently, individual data modifications are difficult or even impossible to track with existing data provenance approaches, and data consistency cannot be guaranteed. Moreover, the substantial amount of data generated by these processes poses a significant hurdle. Typically, large datasets are stored in binary formats such as Hierarchical Data Format 5 (HDF5). Due to their format, these datasets are opaque and cannot be easily searched, indexed or compared. The exchange of large datasets or even subsets becomes impractical due to the increased amount of data that must be handled. This is of particular importance when applying the approach of a central data model and within collaborative environments.

This paper presents a methodology to overcome the challenges associated with managing large datasets, particularly those generated by black-box simulations. The proposed approach focuses on reducing the complexity of these datasets and effectively integrating them into an existing collaborative architecture. Data consistency is achieved even for large and opaque datasets without requiring changes to the existing framework architecture. In particular, data solely based on the binary format HDF5 is examined. The data format was chosen due to its prevalence for high-fidelity analysis, particularly in the area of collaborative simulation and design in engineering applications. A detailed application of the methodology presented is depicted in part B, in which a compressor blisk is examined using three-dimensional (3D) finite element analysis [10].

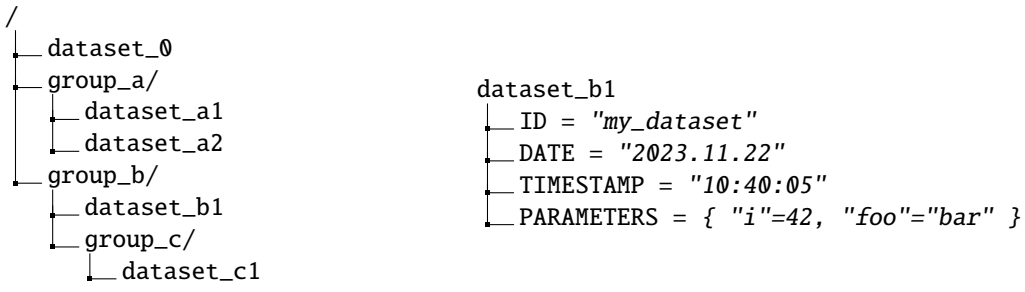
III. Hierarchical Data Format

HDF5 is a binary data format that is optimized to store and manage large data [11]. Internally, such data is organized in a tree hierarchy consisting of groups and datasets. Groups represent nodes in the hierarchy, which may have groups and datasets as child nodes. A dataset represents a leaf node and cannot have any child nodes. A simplified representation of an HDF5 file is depicted in Fig. 1a. For accessing the internal structure and data of an HDF5 file, a variety of programming interfaces in the most common programming languages are available.

A dataset stores a set of data, defined by its dataspace and datatype. The dataspace denotes the layout of the data which may range from a single entry to any multidimensional layout. The datatype in turn describes the data stored in a single entry. Examples may include a plain integer, a floating point, but also complex compound types with variable length entries are possible. Further, a dataset may be chunked and compressed to improve the access to data or decrease its disk usage.

All nodes are associated with metadata. This includes the dataspace and datatype in case of a dataset. Additionally, each node may contain user-defined metadata, also known as attributes. An attribute is similar to a dataset and is also defined by a dataspace and a datatype property. However, ideally it should only be used to add additional information to a node instead of storing complex data, since attributes can neither be chunked nor compressed. Examples for attributes may include a timestamp of creation or parameters used for a tool or simulation that produce the data that the attribute is referring to. A schematic representation of attributes attached to a dataset is shown in Fig. 1b.

HDF5 is a widely used standardized file format in many scientific fields. It forms the basis of data formats and standards like CGNS [5] – well known in computational fluid dynamics – or VMAP [12] – a novel standard for computational structural mechanics.



(a) Internal node hierarchy **(b) Attributes attached to a dataset**

Fig. 1 Representation of the internal structure of an exemplary HDF5 file.

IV. Collaborative Framework for Design and Simulation

GTlab, an open-source framework developed by the Institute of Propulsion Technology at the German Aerospace Center (DLR), serves as the software foundation for the implementation of the new methodology presented in this paper. The framework addresses the multidisciplinary and collaborative design and simulation of aircraft engines [6]. A plug-in architecture facilitates the system's extension and the integration of new functionalities and features.

Various engineers and designers work collaboratively on data records that are subject to the aircraft engine design process or other applications. A data record consists of individual *data objects* organized in a tree hierarchy. Data objects may hold any set of data and provide various functionalities for accessing and modifying their state. In GTlab, all data objects are derived from a common object type, which is intended to provide uniform handling of all data objects within the framework.

The common object type provides a standardized interface to extend the functionalities and operations of data objects. For example, individual data objects or entire data trees may be used in workflows for processing. On a graphical user interface (GUI) level, different types of views can be created that operate on the data of a data object, like schematic models, plots, or fully-fledged editors.

The framework employs a data processing unit that provides various features for data handling. It allows the serialization of all data objects within the application using the common object type. Hence, data records can be written to disk, exchanged via network, or similar. All data objects are serialized by default into the human-readable Extensible Markup Language (XML) data format. Moreover, snapshots of data trees can be taken based on the memento design pattern [13]. By forming the difference between snapshots, changes to the data can be identified. Further, these differences and snapshots can be applied to existing data records.

To facilitate the collaborative workflow, data records are organized in projects, which denote the root node of the data tree hierarchy. The hierarchical structure of a project is defined by the model-based approach of the central data model [7]. An example of a data record in GTlab is depicted in Fig. 2. In this example, the individual data objects denote components of an aircraft engine and are organized in a hierarchical structure. The root node represents the project that combines the data records. The component of a compressor is expanded, visualizing the parent-child-like relations that make up a complex data structure.

Previously, binary data generated by CFD or CMS simulation software, particularly those based on HDF5, had not been integrated into the framework architecture. As part of the presented approach, the following sections describe how HDF5 data can be integrated transparently and efficiently into a collaborative data framework, while retaining existing data handling and processing functionalities.

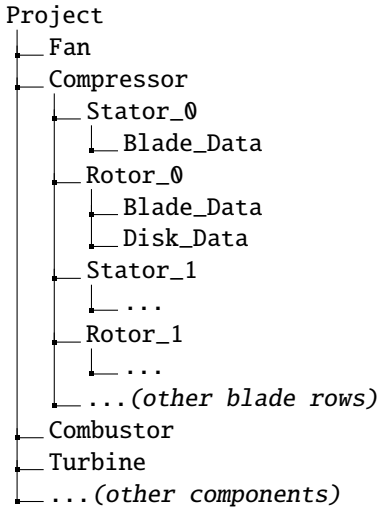


Fig. 2 Exemplary representation of a data record in GTlab. Here the data tree of a compressor is expanded.

V. Methodology and Implementation

This section addresses the integration of HDF5 structures into the GTlab software framework. It involves exposing the internal HDF5 file structure without altering the core framework. The resulting hierarchical representation mirrors the internal layout of HDF5 files. The approach loads data on demand and implements a smart resource-handling mechanism. High-level graphical utilities are presented that facilitate the integration.

A. Representation of HDF5 Structures in the Framework

The first challenge involves the integration of the HDF5-based binary data into the GTlab software framework while retaining the current data processing capabilities. This includes checking for differences between two data objects, the unique identification, and the referencing of data objects. Therefore, the first step is to make the internal structure of an HDF5 file accessible within the framework. To facilitate the integration of HDF5-based binary data, a dedicated HDF5 plugin was developed. This plugin is designed to extend and enhance the capabilities of the GTlab software framework. The key advantage of this approach is that it allows integration without any modifications to the core structure or the underlying data handling mechanisms of the framework.

Based on the common object type architecture of the framework, different types of data objects have been implemented, representing HDF5 groups, datasets, attributes and the files themselves. By means of parent-child relations, a one-to-one transformation of an HDF5 file hierarchy into a corresponding data tree in the framework can be achieved. A dedicated object denotes the root node of an HDF5 data tree and references the HDF5 file on the hard drive or other storage types. All groups, datasets, and attributes of the corresponding HDF5 file are organized in an object hierarchy according to the internal layout of the file. Accordingly, the structure of HDF5 files can be embedded into the framework.

Since all data objects are based on a common object type, uniform data handling is ensured. Consequently, the structure of an HDF5 file in GTlab can be serialized already. Individual nodes of the HDF5 file are made uniquely identifiable and, as such, referenceable. Entire HDF5 files with numerous branches and datasets become accessible and hence, are no longer opaque to the collaborative framework. In addition, high-level changes to the HDF5 data can now be tracked within the framework – a critical requirement in a collaborative environment.

Whereas HDF5 files are made accessible by mirroring their structure into an adequate data tree, the actual data is not loaded into the framework by default. Instead, the data objects only contain the most necessary information to allow referencing their corresponding nodes.

B. Embedding HDF5 Structures into Collaborative Data Records

In collaborative workflows, HDF5 data offers a wide range of interaction possibilities. From a user's perspective an HDF5 file and its corresponding data tree can be attached to any other data object in a data record using parent-child relations. Further, the same file may be appended multiple times to the project's data tree. Thus, in the context of the collaborative environment of GTlab, HDF5 data can easily be affiliated with components subject to aircraft engines or other applications. Users can manipulate, analyse and share this data in a variety of ways to meet different requirements and scenarios. This adaptability allows teams to efficiently process and interpret the data, tailoring their approach to the specific needs of their project. Furthermore, HDF5 files can originate from different locations on hard drives or other storage types, but are combined in a centralised manner using this approach. For the example of a compressor, the results of a CSM analysis of the turbomachinery blading (as depicted in part B [10]) may be stored in an HDF5 file, which in turn may be attached to the blade row objects of the corresponding compressor data tree. Consequently, data affiliations can be expressed clearly.

As noted earlier, a single HDF5 file may become very complex and contain numerous branches with various datasets of heterogeneous data. Therefore, it is essential to allow the attachment of only specific substructures from an HDF5 file that are relevant for a specific application. This capability can be facilitated by employing shortcuts to nodes, where each shortcut originates from a node on a higher level in the HDF5 data tree and points to a node on a lower level. All node objects in between will be hidden, effectively removing irrelevant branches from the data tree without changing the file itself. As previously mentioned, this is particularly valuable for very large HDF5 files with many branches and a high level of complexity in the data structure. To continue the example of a CSM analysis of compressor blading: a single HDF5 file, as generated from a CSM workflow, may contain geometry and result data for the different blade rows of a compressor. Consequently, the same HDF5 file may be appended for each blade row once. In the case of a compressor with eight rotor blade rows, substructures of the file containing the desired information would be appended eight times in total. By employing node shortcuts, only the relevant geometries and result data are shown in the corresponding data

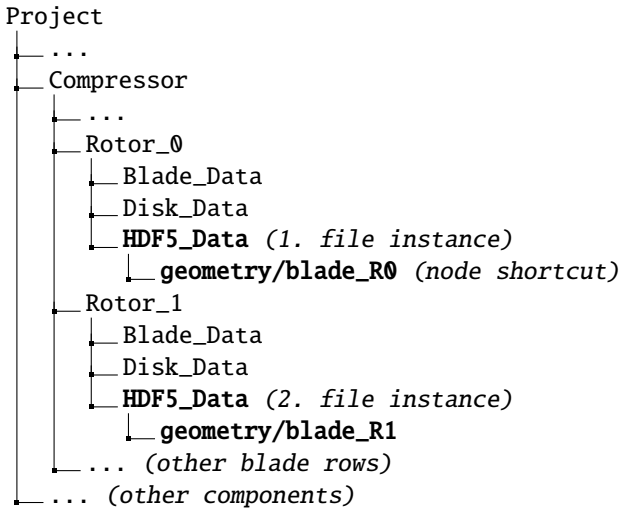
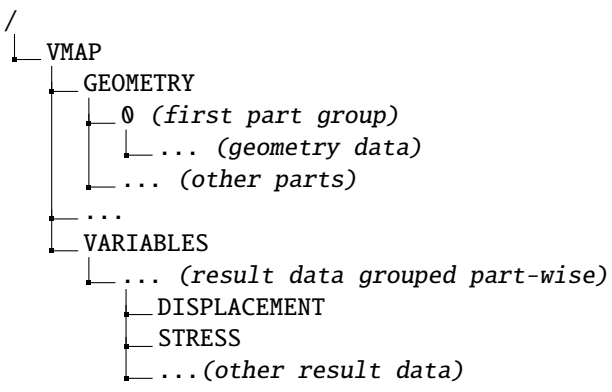


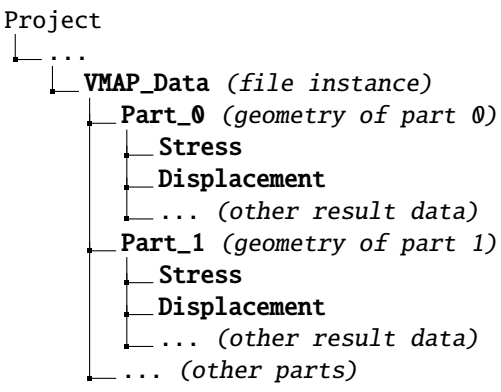
Fig. 3 Exemplary integration of an HDF5 file, containing multiple blade geometries. The file is appended multiple times (highlighted in bold). Node shortcuts are applied, linking only the relevant data for a blade object.

trees (Fig. 3). Thus, a high degree of flexibility is achieved in which data affiliations are expressed clearly.

In the comprehensive dataset of a collaborative application, reorganising the data presented in HDF5 format is often essential. This restructuring is intended to align the data more effectively with the specific needs and goals of the collaborative project, and to ensure that the dataset is optimally configured for collaborative analysis, processing, and decision making. Instead alternative hierarchies of data objects could be employed that are more suitable for a specific purpose, without changing the actual internal layout of the file. This is especially relevant for standardized formats that are based on HDF5, such as CGNS or VMAP. As the name implies, a standardized format follows the same structure in which data is stored. Whereas the layout of the data is optimized on functional and logic level, displaying the entire structure as is may be too technical or low-level for a specific workflow or application. A more fitting representation may be created that hides irrelevant information or condenses branches. This feature is not only relevant for an entire HDF5 file but also substructures may benefit from an altered hierarchy. Providing the ability to override the representation of a HDF5 file without changing the actual layout enables a high degree of flexibility. For instance, this might involve reformatting a VMAP file from the CSM analysis, as depicted in part B [10], which may contain numerous geometries and result data such as stress or displacement information (Fig. 4a). These datasets are each spread over different branches and are deeply nested. By providing an alternative view on the



(a) Simplified hierarchy of a VMAP file as generated from CSM analysis [10]. Geometry and result data are separated.



(b) Integration of a VMAP file using a specialized hierarchy. Here result data is directly affiliated with geometries.

Fig. 4 Exemplary integration of a VMAP file.

VMAP file, in which only the geometries and the corresponding result data are organized in a straightforward hierarchy, a more intuitive and less technical representation can be achieved from a user's point of view. Still, the same VMAP file may be appended multiple times within the same project tree. For each applications, a varying hierarchy may be employed without changing the file in any way and providing the same features of data handling. Figure 4 depicts how a raw VMAP file may be transformed into an alternative hierarchy in the framework to provide abstraction.

C. Efficient Access to HDF5 Data

In the approach outlined to this point, the framework only transforms the structure of an HDF5 file at the metadata level, avoiding to load the actual data into memory. This strategy leads to significant savings in essential system resources.

The approach to handling HDF5 data in this framework is designed for efficiency and resource management. Data from the HDF5 files is only fetched when it is actually accessed, a method similar to lazy evaluation. This ensures that only the data required, whether it is a specific record or a subset, is loaded into the framework – a process referred to as "data internalization". When access to the data is no longer required, the data is removed from memory. Any changes to the internalized data are stored during this unload process, known as "data externalization".

A smart resource-handling approach is employed to avoid fetching data multiple times. Once data access is required, only the requested data is loaded into memory. By leveraging a reference-counting mechanism, subsequent fetching calls are not required. Instead, the reference count of the data handle is incremented for each access. Thus, all actors with access to the handle work with the same data; changes made by one actor are also available to the other actors. Essential safeguards are implemented to avoid simultaneous access to the data in a shared memory environment. Once an actor no longer requires access to the data, the reference count is decremented. When no access to the data is required anymore, and thus, the reference count reaches zero, the data is externalized. If the internalized data has been changed by the framework, the corresponding nodes in the HDF5 file are overwritten. Otherwise, the data in memory can be discarded. To further mitigate repeated data fetching, an additional buffering is employed, which keeps the recently accessed n number of datasets in memory. A schematic representation of how a process accesses the data of an HDF5 object within the framework is depicted in Fig. 5. Accessing the HDF5 data yields a data handle, which internalizes the actual HDF5 data. After processing the data and releasing the data handle; the HDF5 data object is externalized.

As mentioned in section IV the framework GTlab provides a compare algorithm to find the difference between two datasets. This is especially important in collaborative applications, to track changes and modifications of data and to ensure data consistency. Consequently, it is essential to also enable the same level of data handling for HDF5 data within the framework. To elaborate: it should not matter whether the actual data object is located within an HDF5 file, thus external to the project or whether the data object is deeply integrated in the framework. Since the data is internalized when access is required, the existing data handling is already made available. A snapshot of an internalized datset can be taken and compared to another snapshot at a later point. Determining the difference between two HDF5 datasets is thus efficient from a resource perspective, since only the data is fetched and compared that must be examined.

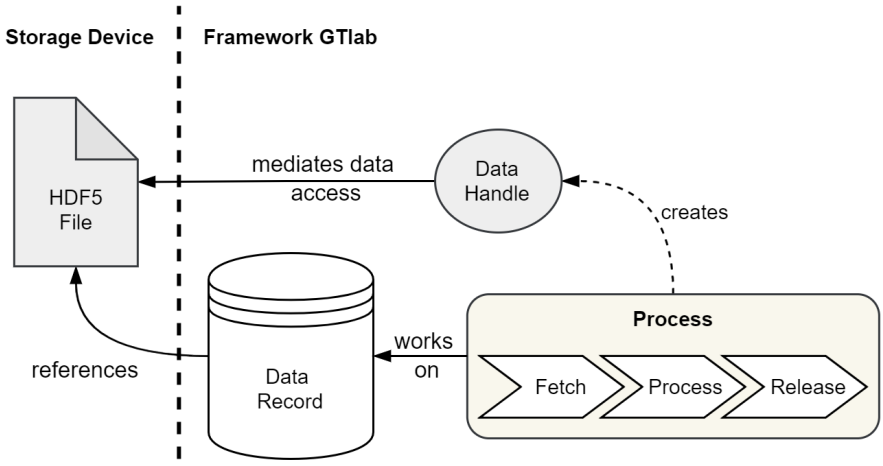


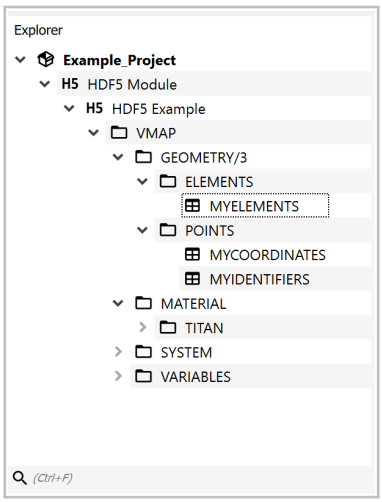
Fig. 5 Schematic representation of how a process accesses the data of an HDF5 object within the framework.

D. User Interface

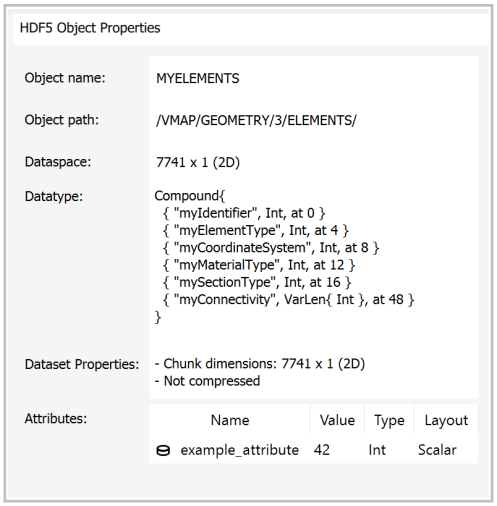
To provide access to HDF5 data through the user interface of the framework, the plugin enhances GTlab's GUI with multiple graphical tools and features. The graphical integration is crucial to achieve a seamless integration of HDF5 data into the user interface. This ensures that HDF5 data objects behave like any other native data object within the framework. The key extensions for graphical integration include: (1) a HDF5 property widget to display high-level information about HDF5 nodes, (2) an embedded HDF5 data viewer, and (3) numerous context menu entries – so-called *quick actions* – for each data object it implements. These utilities are presented in the following:

(1) By attaching an HDF5 file to a GTlab project, its data tree is displayed in the project's explorer (Fig. 6a). The property widget displays the most relevant metadata of the HDF5 object that is currently selected. The metadata includes the HDF5 node name and the full path to the node inside the linked HDF5 file. Detailed information about the datatype and its dataspace are displayed for dataset objects, which also entails whether the dataset is chunked and compressed. Finally, attributes of the referenced node are listed in a table, which displays the names of the attributes, their value, datatype, and dataspace. The value is displayed in a single cell for scalar attributes to provide convenience for the user. Nonetheless, all attributes regardless of complexity can be viewed in a dedicated viewer as depicted in (2). The property widget is shown in Fig. 6b, displaying properties of an exemplary dataset.

(2) The HDF5 plugin implements an HDF5 data viewer, which allows to view datasets and attributes directly within GTlab in a tabular layout. It can be invoked using quick actions. It supports all common datatypes, including variable length, deeply nested, and complex compound types. For multidimensional data with dimensions exceeding two, two-dimensional (2D) slicing is employed. The HDF5 viewer is shown in Fig. 7 in which a subset of an exemplary dataset is displayed. The HDF5 viewer supports different modes for displaying/fetching the data of a dataset. Depending on the process used to generate the data, a single HDF5 dataset may often contain thousands of entries. In such a case, fetching and displaying all entries in a viewer may require substantial computational effort and system resources. However, in the context of GTlab, practice has shown that users are rarely interested in the entire dataset if it contains numerous entries. Instead, they are mostly interested in a portion of the data to check for errors or estimate the data's trend. This requirement can be met efficiently with HDF5 as it also allows the query of custom selections and subsets instead of loading the entire dataset. Using this feature, the HDF5 viewer reads and displays only the n first and last data rows if the number of rows is too large to display. The user can also turn off this behaviour when all data should be displayed.

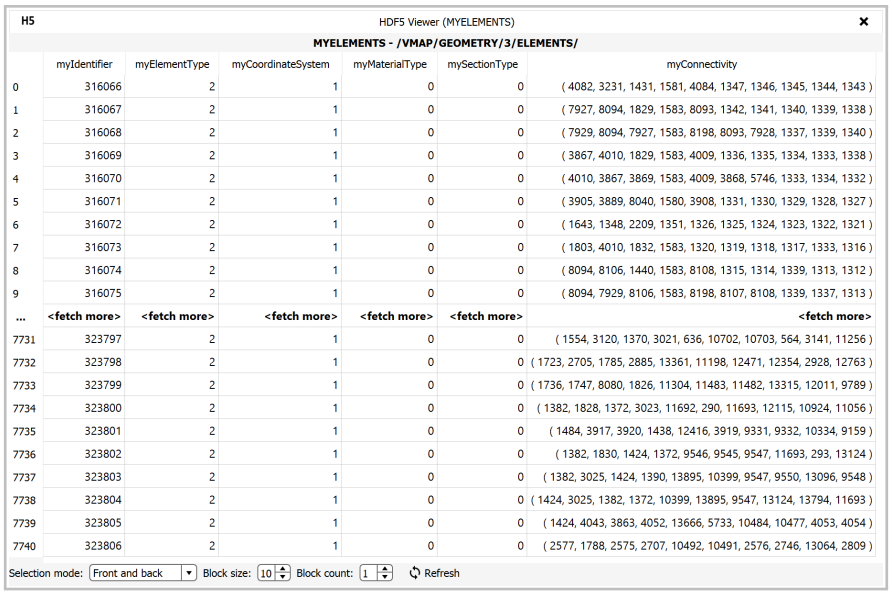


(a) The project explorer in GTlab. Here an exemplary HDF5 file is displayed in which the node `MYELEMENTS` is selected.



(b) HDF5 property widget displays the metadata and attributes of the selected node.

Fig. 6 Graphical integration of an HDF5 file.


HDF5 Viewer (MYELEMENTS)

MYELEMENTS - /VMAP/GEOMETRY/3/ELEMENTS/

	myIdentifier	myElementType	myCoordinateSystem	myMaterialType	mySectionType	myConnectivity
0	316066	2	1	0	0	(4082, 3231, 1431, 1581, 4084, 1347, 1346, 1345, 1344, 1343)
1	316067	2	1	0	0	(7927, 8094, 1829, 1583, 8093, 1342, 1341, 1340, 1339, 1338)
2	316068	2	1	0	0	(7929, 8094, 7927, 1583, 8198, 8093, 7928, 1337, 1339, 1340)
3	316069	2	1	0	0	(3867, 4010, 1829, 1583, 4009, 1336, 1335, 1334, 1333, 1338)
4	316070	2	1	0	0	(4010, 3867, 3869, 1583, 4009, 3868, 5746, 1333, 1334, 1332)
5	316071	2	1	0	0	(3905, 3889, 8040, 1580, 3908, 1331, 1330, 1329, 1328, 1327)
6	316072	2	1	0	0	(1643, 1348, 2209, 1351, 1326, 1325, 1324, 1323, 1322, 1321)
7	316073	2	1	0	0	(1803, 4010, 1832, 1583, 1320, 1319, 1318, 1317, 1333, 1316)
8	316074	2	1	0	0	(8094, 8106, 1440, 1583, 8108, 1315, 1314, 1339, 1313, 1312)
9	316075	2	1	0	0	(8094, 7929, 8106, 1583, 8198, 8107, 8108, 1339, 1337, 1313)
...	<fetch more>	<fetch more>	<fetch more>	<fetch more>	<fetch more>	<fetch more>
7731	323797	2	1	0	0	(1554, 3120, 1370, 3021, 636, 10702, 10703, 564, 3141, 11256)
7732	323798	2	1	0	0	(1723, 2705, 1785, 2885, 13361, 11198, 12471, 12354, 2928, 12763)
7733	323799	2	1	0	0	(1736, 1747, 8080, 1826, 11304, 11483, 11482, 13315, 12011, 9789)
7734	323800	2	1	0	0	(1382, 1828, 1372, 3023, 11692, 290, 11693, 12115, 10924, 11056)
7735	323801	2	1	0	0	(1484, 3917, 3920, 1438, 12416, 3919, 9331, 9332, 10334, 9159)
7736	323802	2	1	0	0	(1382, 1830, 1424, 1372, 9546, 9545, 9547, 11693, 293, 13124)
7737	323803	2	1	0	0	(1382, 3025, 1424, 1390, 13895, 10399, 9547, 9550, 13096, 9548)
7738	323804	2	1	0	0	(1424, 3025, 1382, 1372, 10399, 13895, 9547, 13124, 13794, 11693)
7739	323805	2	1	0	0	(1424, 4043, 3863, 4052, 13666, 5733, 10484, 10477, 4053, 4054)
7740	323806	2	1	0	0	(2577, 1788, 2575, 2707, 10492, 10491, 2576, 2746, 13064, 2809)

Selection mode: Front and back Block size: 10 Block count: 1 Refresh

Fig. 7 HDF5 data viewer in GTlab. Here a 2D dataset is shown. Only the first and last 10 elements are displayed. More rows may be fetched on demand.

(3) The HDF5 plugin implements multiple quick actions for the different HDF5 object types, allowing the user to perform high-level operations on HDF5 data. For the file object, the most common file operations, like renaming and deletion from disk, have been implemented. It should be noted that the HDF5 hierarchy of a file object in a data record is just a snapshot of the actual file on disk at a given time. Changes to the file throughout external processes will not be reflected directly in GTlab to maintain data consistency. Instead, the file object will be marked as outdated. A quick action is implemented to synchronize changes of the HDF5 data objects in memory with the files on disk on demand. However, HDF5 objects that are part in the snapshot before a change, which are no longer present in a sequential snapshot, are not deleted directly. Instead, these objects will be marked as missing. Added data objects will also be highlighted accordingly – this way, a user can browse through the top-level modifications of an HDF5 file. Quick actions are also available to rename nodes and attributes in the linked HDF5 file. Similarly, a node and attribute may be removed entirely from the HDF5 file. Finally, for all HDF5 nodes attributes can be appended using quick actions. As presented in subsection V.B, shortcuts to nodes are employed to hide irrelevant information and branches in an HDF5 data tree. The shortcuts are implemented using the quick actions system. When activated, the user is prompted via a dialog to select the node at a higher level. Once created, the nodes and branches in between are hidden. A node shortcut may also be expanded once again.

VI. Conclusion

This paper presented an approach for integrating large binary data based on HDF5, produced by simulation processes such as CFD and CSM, in a collaborative architecture. Previously, the data was considered opaque since the software framework had no insights into the internals of HDF5 files.

To make the internal structure of an HDF5 file accessible and searchable, the file structure was explored and transformed into a related hierarchy in the GTlab framework, in which each data object represents a corresponding node of the HDF5 file. By embedding only the relevant data structures and employing shortcuts to nodes or displaying alternative representations for a data tree, irrelevant information is hidden, and a more fitting integration is achieved – without modifying the actual HDF5 file. Since the same file can be appended multiple times to the project and to any other data structures in the framework, data affiliations are expressed clearly. All data objects in the framework GTlab share a common object type. As a result, a uniform handling of HDF5 data is ensured. All necessary changes are implemented in the form of an HDF5 plugin. Only the file structure is appended to the data tree of a project, whereas the actual data is accessed by employing a reference-counting mechanism. This mechanism ensures that only the required data is fetched for the minimum duration needed. Thus, a lightweight integration can be achieved, in which essential system

resources are conserved. Additional graphical utilities have been implemented, which provide a seamless integration from a user's point of view. The utilities include the graphical representation of metadata associated with HDF5 nodes in the GTlab framework. A dedicated HDF5 viewer allows efficient access to the raw data of a dataset or attribute.

Whereas this paper presents a viable approach for integrating large opaque data in a collaborative framework, certain challenges regarding large data remain. A critical challenge is the exchange of large amounts of data, which requires significant bandwidth and computation effort in a collaborative environment. Nonetheless, the approach presented may lay the foundation for an efficient solution. One approach that is currently investigated is to relocate the processing of large data to the data storage location by means of a remote processing unit, in the spirit of edge computing [14]. For HDF5 files, only the structure could be exchanged via a network. Engineers and designers can then examine the data tree within the software framework and browse the high-level representation. Access to individual datasets would be exchanged efficiently by only transmitting the data that is needed, as presented in this paper.

The application and evaluation of the presented methodology is carried out by a detailed CSM analysis described in part B [10].

References

[1] Kunc, O., and Broecker, M., "Integration of large data based on HDF5 in a collaborative and multidisciplinary design environment, Part B: Application to Structural Mechanics of Gas Turbine Engines," *AIAA Scitech 2024 Forum*, 2024.

[2] Reitenbach, S., Krumme, A., Behrendt, T., Schnös, M., Schmidt, T., Hönig, S., Mischke, R., and Mörland, E., "Design and Application of a Multidisciplinary Predesign Process for Novel Engine Concepts," *Journal of Engineering for Gas Turbines and Power*, Vol. 141, No. 1, 2019.

[3] Vieweg, M., Hollmann, C., Reitenbach, S., Schnoes, M., Behrendt, T., and Krumme, A., "Collaborative Aircraft Engine Preliminary Design using a Virtual Engine Platform, Part B: Application," *AIAA SciTech Forum*, 2020.

[4] Nasser, T., and Tariq, R., "Big data challenges," *J Comput Eng Inf Technol 4: 3. doi: http://dx. doi. org/10.4172/2324*, Vol. 9307, No. 2, 2015.

[5] Poirier, D., Allmaras, S., McCarthy, D., Smith, M., and Enomoto, F., "The CGNS system," *29th AIAA, Fluid Dynamics Conference*, 1998, p. 3007.

[6] Reitenbach, S., Vieweg, M., Becker, R., Hollmann, C., Wolters, F., Schmeink, J., Otten, T., and Siggel, M., "Collaborative Aircraft Engine Preliminary Design using a Virtual Engine Platform, Part A: Architecture and Methodology," *AIAA Scitech 2020 Forum*, American Institute of Aeronautics and Astronautics, 2020. doi:10.2514/6.2020-0867.

[7] Reitenbach, S., Hollmann, C., Schmeink, J., Vieweg, M., Otten, T., Haessy, J., and Siggel, M., "Parametric datamodel for collaborative preliminary aircraft engine design," *AIAA Scitech 2021 Forum*, 2021, p. 1419. doi:10.2514/6.2021-1419.

[8] Reitenbach, S., Vieweg, M., Hollmann, C., and Becker, R. G., "Usage of Data Provenance Models in Collaborative Multidisciplinary Aero-Engine Design," *Journal of Engineering for Gas Turbines and Power*, Vol. 142, No. 10, 2020. doi:10.1115/1.4048436.

[9] Cui, Y., and Widom, J., "Lineage Tracing for General Data Warehouse Transformations," *The VLDB Journal—The International Journal on Very Large Data Bases*, Vol. 12, No. 1, 2003, pp. 41–58. doi:10.1007/s00778-002-0083-8.

[10] O. Kunc, M. B., "Integration of large data based on HDF5 in a collaborative and multidisciplinary design environment, Part B: Application to Structural Mechanics of Gas Turbine Engines," *AIAA SciTech Forum, submitted for publication, session "Meshing, Visualization, and Computational Environments"*, 2024.

[11] Folk, M., Heber, G., Koziol, Q., Pourmal, E., and Robinson, D., "An Overview of the HDF5 Technology Suite and Its Applications," *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, Association for Computing Machinery, New York, NY, USA, 2011, p. 36–47. doi:10.1145/1966895.1966900.

[12] "VMAP Standard Specifications," `https://vmap-standard.org`, 2020. Accessed Dec 1, 2023.

[13] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.

[14] Garcia Lopez, P., Montresor, A., Epema, D., Datta, A., Higashino, T., Iamnitchi, A., Barcellos, M., Felber, P., and Riviere, E., "Edge-centric computing: Vision and challenges," , 2015.