INTEGRATED MANAGEMENT OF EXPERIMENTAL RESEARCH- AND META-DATA FOR FAN TEST RIGS

Nils PREUß, Peter F. PELZ

Technische Universität Darmstadt, Chair of Fluid Systems, Otto-Berndt-Straße 2, 64287 Darmstadt, Germany

SUMMARY

The availability of descriptive metadata is mandatory for the long term usability of experimental research data. This paper introduces a modular data model based on the HDF5 file format that is applicable for the typically heterogeneous and evolving experimental setups. It enables uniform storage of related measurement data in different processing stages and corresponding machine actionable metadata in the same file. On this basis a software design is proposed that streamlines data storage, access and processing by means of object oriented programming, limiting the need for customized data acquisition and processing software per test rig configuration. Furthermore, application examples for typical fan test rigs are presented and discussed.

INTRODUCTION

Experimental studies still remain the most reliable validation technique, as well as important tools during fan design and analysis, development of prediction methods and understanding of physical phenomena. A considerable amount of time, money and know-how is invested conducting such studies, generating and evaluating large amounts of experimental data in digital form. Alongside scientific publications, this research data is an important product of scientific research [1]. The context of this data needs to be comprehensibly documented for the long term to fulfil the quality requirements of good scientific practice and to nurture further research [2].

Like most experimental setups in mechanical engineering research, fan test rigs are typically evolving systems subject to continuous improvement. As a result, heterogeneous datasets are generated. The collective long-term usability of these heterogeneous datasets is largely dependent on the availability of descriptive metadata. As of 2014, only 22% of data is suitable for analysis and only 5% of data is ever analyzed because the information is not sufficiently characterized through metadata [3]. Without thorough documentation of this metadata, interpretation of the data itself is cumbersome and prone to errors [4] [5]. What is considered data or metadata depends on the point of view, scope and objective. Different persons involved with an experimental research project may look at a test rig

setup and its documentation from different perspectives [6]. This paper focuses on the viewpoint of measurement instrumentation and the data flow. Metadata is therefore mainly considered auxiliary information describing instrumentation setup, calibration, configuration or unit under test. Data directly yielded by instrumentation drivers is considered the most reliable. This raw data is further processed with respect to instrument calibration or physical models. These values of system state variables are used for subsequent analysis. To ensure the reliability of such analysis, any dataset must be traceable to its origin, i.e. the raw data.

Currently most researchers develop their own data schemas based on CSV or spreadsheet files [7]. Metadata, i.e. information about instrumentation setup, calibration, configuration or unit under test is included mostly inconsistent or not at all. Custom scripts are used for the processing of raw data or subsequent analysis. Information describing the processing actions implemented by these scripts is typically not well documented. The lack of metadata describing conversions of raw data to the measured quantity of the corresponding instruments leads to a proliferation of scripts that are highly coupled with specific data sets. This in turn leads to error prone data processing and analysis.

The heterogeneity of data and software further proliferates with the evolution of the setup, or when considering a larger scope of multiple test rigs [8]. Improvements of the experimental setup impose cumbersome and frequent adjustments of data storage schemes as well as data acquisition and processing software. As a result, expanding on data or results produced by predecessors is difficult, information is lost and redundant investigations might be conducted. To solve this, the development of a concept for standardized, traceable and interoperable measurement data storage is needed.

It is critical to disconnect data and code, i.e. moving information specific to a dataset out of any scripts and programs and into the data file. On this basis, longevity of the data can be achieved, and the maintenance cost for data management libraries can be significantly reduced. In turn, this facilitates innovation because researchers can spend more time on generating actual results and less time addressing relevant datasets individually by means of low-level data manipulation techniques.

This paper is structured as follows: (i) The first section outlines the author's general approach towards sustainable research data management. (ii) The second section introduces a modular, object oriented data model that enables the uniform representation of heterogeneous experimental setups. (iii) A software design is proposed that streamlines data storage, access and processing by means of object oriented programming, limiting the need for customized data acquisition and processing software per test rig configuration. The paper closes with (iv) a discussion of application examples for typical fan test rigs and (v) final conclusions.

APPROACH AND DESIGN PRINCIPLES

As the introduction describes, the main challenge is the documentation of multiple heterogeneous fan test rigs that evolve over time. More specifically, it is desired to avoid constant adjustment of measurement and control software as well as cumbersome versioning and matching of data to appropriate processing software. The goal is to generate structured, self-descriptive data packages that contain all descriptive information relevant for interpretation.

The approach of this work focuses on integrating the metadata creation and utilization into data acquisition and processing tools, to establish a solid basis of available metadata for further analysis. Capturing metadata right at the stage of data generation prevents possible information loss due to human error during delayed retroactive assignment [9]. This implies the machine-actionable documentation of utilized instrumentation, software configuration and unit under test, from the viewpoint of the data flow during operation of the test rig. Well-defined and machine actionable metadata being available means further pre- and post-processing can be streamlined and data access can be supported by search engine tools. The acquired data as well as any derived data and the metadata describing their respective origin are stored in one place.

The metadata of any dataset is concerned with the information describing what it represents, how it was generated and when. Being able to trace any dataset back to its origin is imperative for the thorough documentation of the end result. Special attention is given to the distinction of metadata and software. Any data processing action should be driven by the input data – all configurable information related to how to convert the data must be external to the software routine that implements the conversion. Any parameter is first defined for the target dataset, then given to the processing routine, together with the source dataset. No parameter may be hardcoded to the routine, unless the routine name clearly indicates its purpose and value. Therefore, the metadata of any dataset has to include the responsible software or routine as well as its corresponding configuration. This metadata can be leveraged for data driven processing routines by means of object oriented programming.

The developed solution must be applicable to currently operational test rigs as well as already acquired data. This means the existing data acquisition and control software must be expanded to handle as much metadata as possible with as little extra effort as possible. Additionally, the capability to add currently separate metadata to relevant existing datasets and produce self-descriptive data packages is mandatory. The developed solution should be as non-intrusive to the active research as possible. Therefore, in addition to the flexibility to represent heterogeneous sets of data and metadata, other key requirements are simplicity and interoperability.

The main challenges outlined in the introduction and the approach proposed in this section can be summarized as follows:

(i) The information value of archived datasets decays quickly, depending on the quality of its documentation. To retain the usability of datasets, descriptive metadata is to be embedded in the datasets. Metadata should be stored as early, complete and automatic as possible; feasible by integrating the creation of metadata into data acquisition software.

(ii) Tracing processed or result data back to the original raw data and experimental setup is cumbersome and time consuming. To ensure the reliability of research results, the different processing stages of data must be relatable to each other, including information about the processing actions; feasible by bundling directly related data and metadata to a structured, self-descriptive data package.

(iii) Improvements of the instrumentation setup impose changes of static data acquisition and processing software. To prevent frequent adjustments of software, routines should be implemented metadata-driven, leveraging machine actionable metadata; feasible by utilizing object oriented programming.

THE PARENT FILE FORMAT

As the previous section states, no uniformly applicable tools are established for the documentation of the highly heterogeneous experimental studies for fan design and analysis. The outlined approach suggests to solve this by representing instrumentation and calibration data in the form of object-oriented sets of metadata. This metadata enables data driven data processing routines, which once developed require low maintenance effort and prevent frequent adjustments of software. For the development of a flexible enough data model that still can provide standardized interfaces, a suitable parent file format must be selected. The following paragraphs discuss the most important requirements.

(i) Ease of use is an important criterion, as the developed solution should interfere as little as possible with active research. A researcher must be able to utilize the file format in any programming language and operating system of choice through a simple interface, ideally with a moderate learning curve. This in turn implies a high degree of (ii) portability and or interoperability, as well as (iii) language and platform independence. Extraction of data subsets must be fast and uncomplicated to implement, thus (iv) random access of contents is desired. To ensure long term accessibility of contents as well as convenience, it is important that (v) the file format is self-descriptive.

(vi) Most importantly, to represent complex relations of data and metadata, the file data structure must supplement the representation of these relations. Currently used CSV or other spreadsheet based data models enforce a rigid schema specifying fields and fieldnames. This does not map well to the diverse data and metadata to be stored and provides no natural mechanism to relate datasets. While it is certainly possible to markup these relations from text strings following a special syntax, formats like XML or JSON are better suited for that approach. Another option would be to utilize the concept of primary and foreign keys employed by relational database systems (RDBMS). This approach is cumbersome to implement in the evolving data acquisition and processing software and implies an enormous maintenance cost. Therefore a file format implementing a graph based data model is preferred, which allows relating data or metadata records directly.

HDF5 is an open source, general purpose file format that provides the foundation to structure data on disk as a directed acyclic graph (DAG) and to attach lists of attributes to each data object. The following section gives a brief overview of its structure and capabilities. The Hierarchical Data Format (HDF) open standard was originally developed at the National Center for Supercomputing Applications and maintained by a non-profit organization, the HDF Group. The File format is independent of storage medium, programming environment and operating system. Open source platform independent bindings to read and write HDF5 files are maintained by the HDF group for C, C++, CLI (.NET), FORTRAN and Java. As HDF5 has been adopted by a large community over a broad range of disciplines [10], third party libraries to operate on HDF5 files exist for all programming or scripting languages relevant in the scientific community. Examples include MATALB, LabVIEW, Python, R, Mathematica, IDL, Julia and Octave [11].

Figure 1 shows the abstract base model of HDF5 that allows users to form hierarchic or DAG structures from a set of base objects [12]. These include the file root, groups and datasets, all of which can be further described by a set of attributes as key-value pairs. Each object is uniquely identifiable and accessible via file internal URLs, allowing for random access of file contents. The stored representation is self-describing, i.e. the format includes all necessary information about data types and dimensions to read and reconstruct the contents of a file. The user, application or program does not need to know the implemented structure beforehand, it can instead be programmatically explored. This allows to view and edit any file content in the available graphical editor [13], independent of actual stored data or structure. Additionally, tools for the conversion of HDF5 to data exchange formats like XML or JSON are available [14] [15].

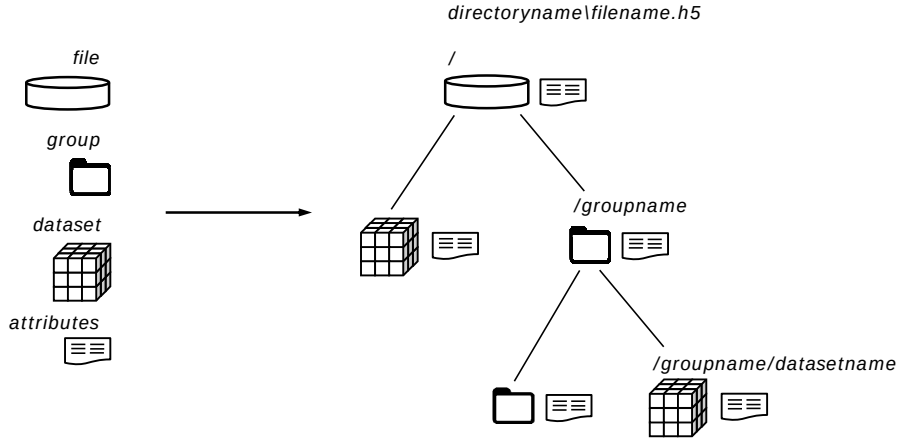

Figure 1: Overview of HDF5 base model objects and exemplary hierarchic data structure.

These above described features lead to a high level of interoperability, which motivated the decision to use HDF5 over other formats designed to store experimental data [16]. Various projects and communities similarly use HDF5 as the foundation for more domain specific data models [17] [18] [19] [20]. The utilization of HDF5 enables the development of a flexible data model at an excellent cost-benefit-ratio, while still serving as a stepping-stone to other file formats or database solutions.

DATA MODEL DESIGN

A suitable data model must be able to represent a multitude of possible test rig setups and configurations as well as units under test and research objectives. It must be applicable for the various experimental setups that are currently operational or have produced measurement data in the past that is still relevant to the scope of research. Therefore, an object oriented design method is applied to develop a data model that mandates generic classes of relevant information objects and is extensible via introduction of more specific subclasses thereof. The first prototype of an interface for this data model design is expected to be implemented in form of by-value classes in the programming environments MATLAB and LabVIEW. As a result, the object structure is designed purely hierarchical in this first iteration of the development process. The developed design pieces and their relationships are visualized using the Unified Modeling Language (UML).

In many experimental setups generic I/O-hardware is used to measure or generate electric signals. Figure 2 shows exemplary setups for measurement and control instruments. The right-hand side illustrates a separated setup of the actual temperature probe, measuring bridge and AD-converter. The measured electric signal is translated back to the physical quantity of interest using nominal instrument characteristics provided by the respective vendor. The left-hand side illustrates a setup of a variable-frequency-drive system, controlled via an analog signal provided by a DA-converter. Again, the actual physical quantity of interest must be translated to an electric signal of the appropriate value. In both cases, the range of physical quantity and electric signal is dependent on the characteristic of the utilized instruments. Therefore, each separate instrument is mirrored by a separate software representation. For each signal conversion in the real world, an inverse conversion is necessary in the software system.

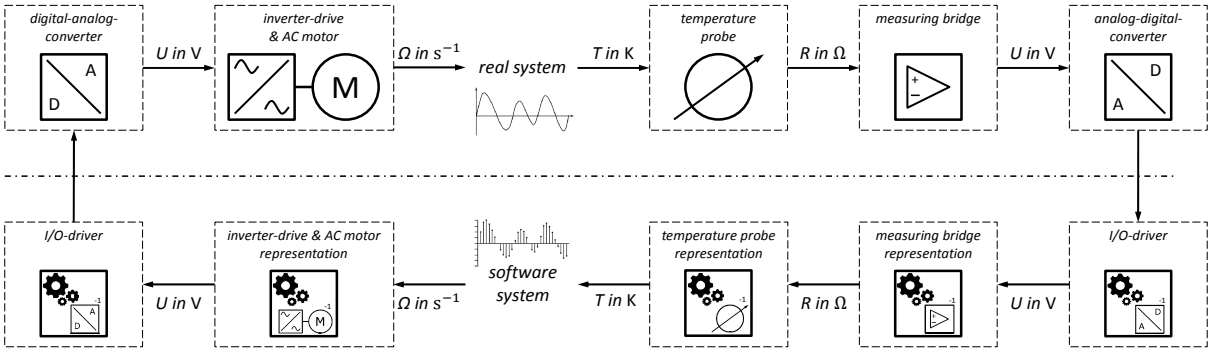

Figure 2: Exemplary arrangement of instrumentation for controlling the rotation frequency of an AC motor (left-hand side) and measuring a temperature (right-hand side).

To avoid frequent changes of the software system, the processing actions implementing the corresponding instrument characteristic should be data driven. The data model in turn must be able to store information describing arbitrary sets of instruments and their configuration. Most importantly, it needs to include the utilized calibration data, i.e. the computational models that implement each instruments characteristic.

This is achieved by introducing the class of *datachannels*, extending the concept of I/O-channels, which typically represent the configuration of the I/O-hardware per signal as well as the corresponding data. Each *datachannel* relates a set of *instruments* to the data it produces. Each object of class *instrument* is in turn described by a set of informational attributes and a *model*. An object of class *model* represents the configuration of the software routine that implements the data conversion of the corresponding instrument (see section ‘Software design’).

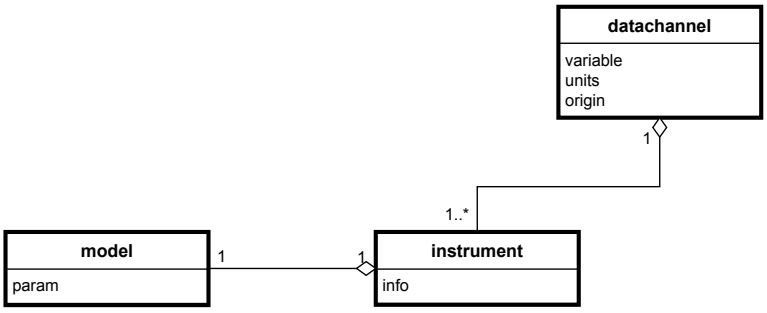

Figure 3: UML diagram showing the relationship of the introduced classes datachannel, instrument and model.

The data model therefore must be able to describe raw data acquired via the I/O-driver and any processed data representing the measured quantity of connected instruments separately. This is achieved by introducing the *origin* attribute for *datachannels*. In the case of a *datachannel* carrying raw data, this attribute value simply states this fact. In the case of a *datachannel* carrying processed data, the attribute value holds a reference (URL or URI) to the *datachannel* that represents the origin data. The processed data can then be recreated by re-evaluating the computation *model* of the related *instruments*, using the data provided by the *origin datachannel*.

As stated, the object structure resulting from the data model design is required to be purely hierarchical. Therefore, all *datachannels* representing any relevant system variable regardless of processing state are bundled in the scope of one *measurement run*. It is reasonably implied that the setup of instrumentation and the corresponding object structure does not change over the course of a single *measurement run*. As a result, one *datachannel* can be related to an arbitrary number of *datasets.* Each object of class *dataset* is further described by attributes, including at least a timestamp. Any object of class *measurement run* is further described by a set of optional informational attributes. Additionally, *parameters* describing environment conditions or the unit under test can be attached to the *measurement run*. Objects of class *parameter* can in turn be further described by attributes of any data type and still provide easily machine actionable metadata of the *measurement run* to data processing tools.

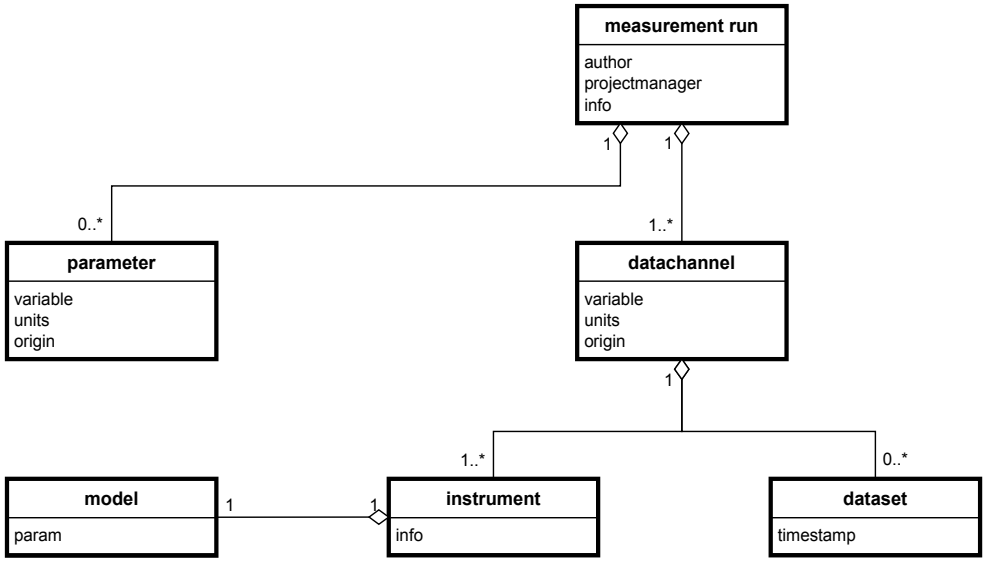

Figure 4: UML diagram showing the data model, forming a hierarchic structure of generic classes.

Based on this generic base model, highly diverse experimental setups can be represented. This covers different test rigs of different projects as well as adjustments of one and the same test rig over time. Based on the embedded information, data processing actions corresponding to utilized instruments and their calibration can be configured and re-run. Therefore, scripts and programs deployed for data processing do not need to be adjusted for every change of the hardware setup. Arbitrary stages of raw or processed data can be stored and related to their origin in the scope of one measurement run. The measurement run can be further described by parameters representing nominal operational state, test environment conditions, and the unit under test.

SOFTWARE DESIGN

The model proposed by the previous section must be made actionable by implementing software interfaces for measurement and control software as well as processing software. At the Chair of Fluid Systems at the TU Darmstadt, the primarily used software environments for data acquisition and processing are LabVIEW and MATLAB. Since the data model was designed as a hierarchic structure of objects, it can be implemented directly as a composition of by-value classes, which both environments support. This section describes the most important aspects of the software interface design, including the mechanisms for automatic execution of appropriate data processing routines, using embedded calibration data.

To minimize code duplication and to keep the data model extensible, the interface implementation introduces the abstract class *data item*. This class implements methods for the organization of HDF5 URLs as well as reading and writing of attributes and nested objects. All classes presented in the previous section inherit these capabilities, their implementation and future extensions therefore require mostly adjusted specifications, rarely implementation of methods. Every *data item* object stored to a HDF5 file is accessible by a URL that consists of the base URL determined by objects further up in the hierarchy and the objects name. To enable the reconstruction of the object structure serialized to the HDF5 file, the class of each object is designated by attributes stating class, subclass and version. Additional attributes are either enforced by class definition, or arbitrarily customized by the user. Figure 5 shows the relation of the abstract class data item to the generic classes specified by the data model as well as an exemplary mapping of objects to a HDF5 file.

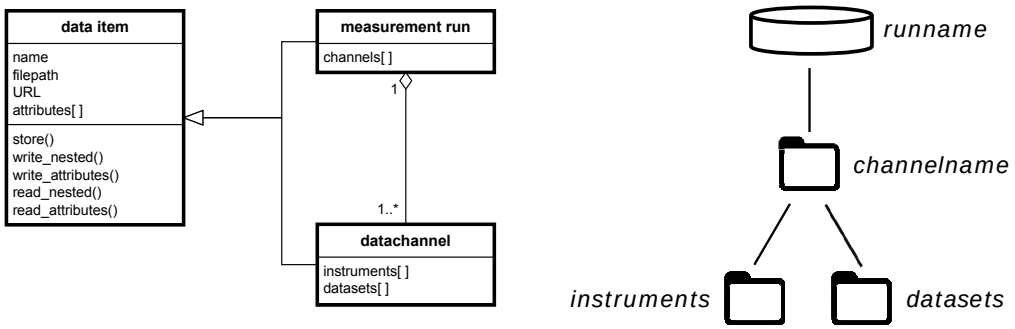

Figure 5: relation of the data item abstract class to generic data model classes (left-hand side) mapping of objects to the HDF5 file structure (right-hand side)

The mechanisms for automatic execution of appropriate data processing routines are implemented by the classes *datachannel, instrument* and *model* using the strategy pattern [21]. The class *datachannel* implements the method *processdata*, which obtains the input data from the *datachannels* specified in the origin attribute and delegates the actual data processing to the nested *instrument* objects. The class *instrument* in turn implements the *processdata* method using the strategy pattern once again: It iterates over the input *datasets* and calls the *apply* method of the nested *model* object.

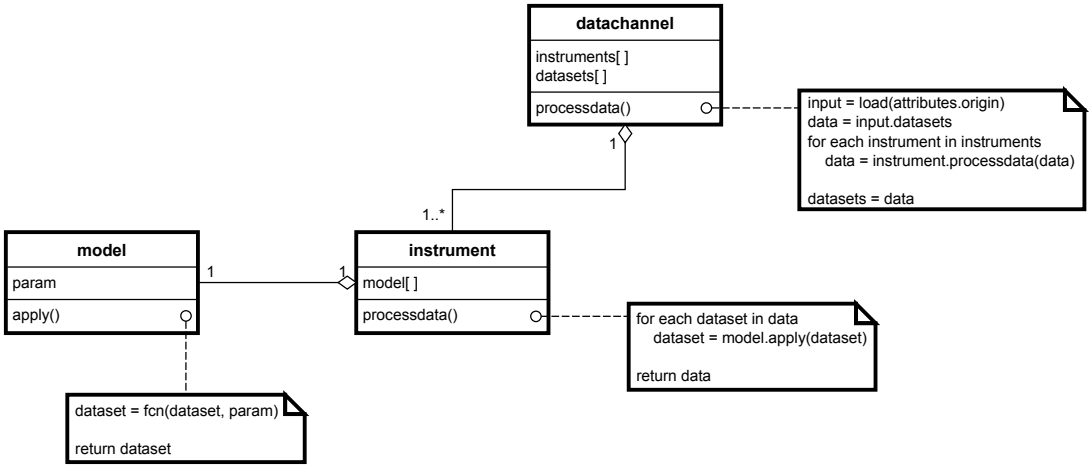

Figure 6: Mechanisms for automatic data processing with respect to corresponding instruments calibration.

This design solves the need for conditional statements, as *datasets* are bundled with the correct processing actions as *datachannels.* Furthermore, it minimizes the need for extensive subclassing of the class *datachannel* by delegating the processing logic to the nested *instrument* and *model* objects. The complete routine to process data originating from any combination of instruments can be composed as a set of smaller routines, encapsulated by appropriately configured objects. As a result, data driven evaluation of measurements is enabled by storing this configuration together with the data. This prevents frequent adjustments of software due to a change in hardware setup. Since the first prototype implementation focuses on the viewpoint of data storage and processing, no extensive subclassing of the generic classes defined by the data model is required. A future goal will be to fully integrate the data storage mechanisms with I/O-hardware communication, which will require an additional subclass, or delegate class, per I/O-instrument.

IMPLEMENTATION AND USAGE EXAMPLES

As stated, a considerable amount of previously acquired measurement data is still relevant for current research projects at the Chair of Fluid Systems at the TU Darmstadt. To leverage the capabilities of the proposed data model and file format for data driven processing and programmatic exploration of content this legacy data has to be migrated. The appropriate environment for this migration is MATLAB, where currently all legacy data is imported and handled in some form. Measurement and control software for operational test rigs is written in either MATLAB/SIMULINK or LabVIEW, which provide the capability to call MATLAB code directly. A first prototype of the software interface is therefore implemented in MATLAB. To represent the hardware setup of any test rig, researchers can compose a structure of appropriately configured objects via a MATLAB script or routine. Legacy data is mapped to a corresponding object structure, currently missing metadata is assigned manually. Likewise, the representation of an operational test rig is built as a HDF5 file with the appropriate structure and metadata. The actual measurement data is then fed into the file, each at the corresponding location, leveraging the random access capabilities of HDF5.

As application examples, this section describes two aspects of the data management in the scope of measurements conducted for the validation of efficiency scaling methods. At the Chair of Fluid Systems at TU Darmstadt, Hess [22], Stonjek [23] and Saul [24] have utilized various different experimental setups to generate data that is relevant for analysis. While the general objective of these measurements has remained the same, their setup, utilized equipment and method of data management has changed over time. As a result, accessing and re-evaluating the raw data from a new perspective is cumbersome and imposes a great effort. The proposed data management approach aims to enable uniform documentation and utilization interfaces for this heterogeneously generated data.

For simplified comprehension, the following sections discuss the core principles on application examples in the smaller scope of a fan test rig currently utilized at TU Darmstadt. Figure 7 shows a schematic of the DIN 24163 compliant test rig, which deploys the following instrumentation to measure relevant system state variables. The aerodynamic torque and rotation frequency of the impeller are measured using variants of a common torquemeter type, which yields electric signals proportional to the measured quantities. The various system pressures are acquired by a pressure scanner which directly provides pressure values via a serial interface. Different channels of the pressure scanner are used for different experiment configurations. To measure the system temperatures, the respective resistance of various Pt100 probes is measured by means of a measurement bridge module. All instruments are calibrated in specified maintenance intervals.

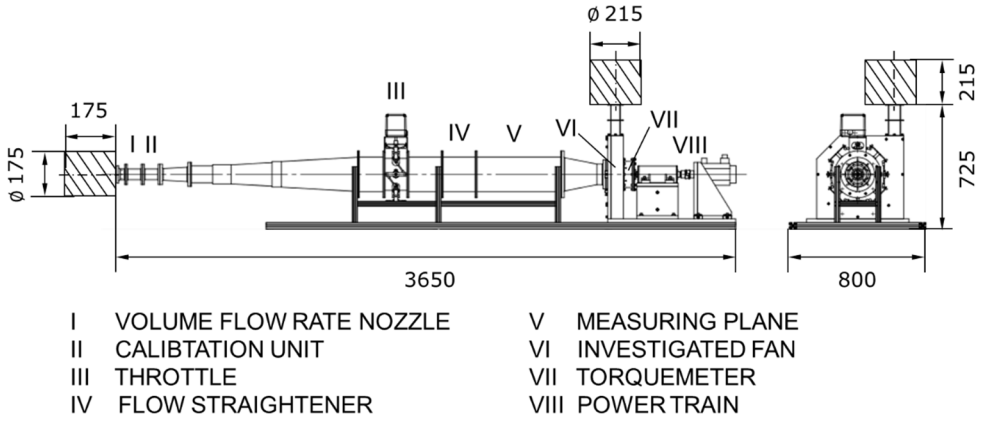

Figure 7: DIN 24163 compliant fan test rig utilized at TU Darmstadt for validation of efficiency scaling methods.

Application example 1: general file and metadata management

Studies are conducted in sequences of measurement runs, consisting of multiple single measurements. The data management fits the general current situation described in the introduction: For each measurement run a new directory is created, storing multiple CSV files carrying parts of the data. One CSV file per instrument per single measurement is stored. The experiments parameters and setup, i.e. unit under test, operational parameters, instruments and calibration are represented by a string of abbreviations in the file name and directory. The use of abbreviations is motivated by the exact denomination of parameters being verbose and the limited length for file paths on windows systems. As a result these denominations and layouts of CSV data form a rigid schema that has to be parsed and mapped to the actual descriptive information. Evolving test rigs and objectives as well as differing units under test make the maintenance the resulting multitude of schemas problematic.

When processing these data files using MATLAB scripts or routines, specific script versions are selected according to the parsed abbreviations via conditional statements. These include operational parameters and characteristics of utilized instrumentation. The processed and aggregated data is saved to separate MATLAB files. As a result, most of the metadata is obscured by being buried in file paths, code or comments therein. Raw and processed data is spread across different files and formats.

The introduced data model in conjunction with HDF5 as the storage backend solves this proliferation of files and denomination schemas by enabling flexible storage of measurement data in different processing stages and corresponding metadata in the same file. Metadata is available in a structured form and thus machine actionable. This allows exploring or crawling measurement data files by means of iterating over the content objects.

Apart from metadata critical for data processing, which is handled by the proposed software interface, the regular HDF5 interface can be used to add customized descriptive information. To re-iterate: A data file should contain all descriptive information relevant for interpretation. For example, the top-level section of a file should contain comprehensive schematic drawings of the test setup and fan under test with specifications of the measuring planes and important dimensions. Furthermore,

photographs of the fan under test or close-up sections of critical setup parts should be included. Values or comments and descriptions should be made available to software interfaces by adding them as attributes or datasets. Figure 8 shows a comparison of the different data management approaches.

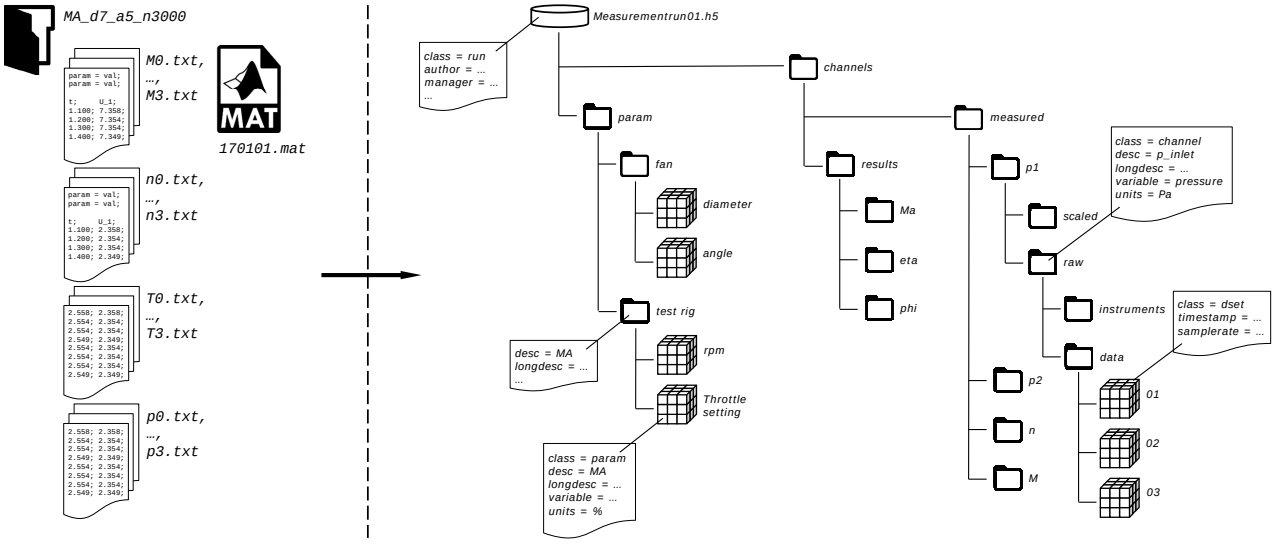

Figure 8: Comparison of the current data management approach based on directories and files to the proposed approach based on a modular data model and the HDF5 file format.

Application example 2: management of different instruments and calibrations

Each torquemeter utilized in the test rig yields voltage signals proportional to both the torque and the rotational frequency. Six different variants of the torquemeter are utilized, depending on the size of the investigated fan. Each variant yields electric signals in the range of 0-10 volts that map to different measurement ranges. The characteristic correlation of measurement range and signal range is calibrated in specified time intervals. For each torquemeter variant, multiple calibration protocols exist, that apply to specific data sets acquired at different points in time. This proliferation of different calibration and measurement data is especially aggravated when datasets acquired outside of the scope of this example are considered for analysis.

As the previous section indicates, the appropriate processing actions for the conversion of raw data (values of electric signals) to processed data (values of physical quantity corresponding to the instrument) are currently selected by means of a multitude of conditional statements. Each different torquemeter, or rather calibration, requires its own case in a conditional statement based on a parameter. The parameter is in turn depending on the conditional statement handling the parsed abbreviation. Hence, extending this system requires designating a new torquemeter or calibration with a new abbreviation, extending the conditional statements and implementing the transmission behavior via a new routine. All of this is boilerplate code, which is prone to errors, difficult to trace for a non-familiar user and is not transparent when viewing any data without digging in scripts and functions.

The introduced data model and software design solve this in the following way: As the previous section demonstrates, each different torquemeter and specific calibration can be represented by appropriately configured *instrument* and *model* objects. The information contained in these objects is then stored in an HDF5 file according to the data model specification. Data processing is now simply implied in a data driven way, when calling the read method on the channel object that hosts the instrument. A mathematical model for the processing routine is specified by the concrete *model* subclass and the routine is parameterized by the stored calibration data. Of course, this means each new different configuration is handled like a different instrument, despite actually representing the same hardware. To reflect the latter, each instrument should include at least one attribute that uniquely identifies the represented hardware, for example the serial number.

This implementation is extensible for arbitrary new torquemeter or other instruments and their configuration or calibration simply by adjusting the calibration data that parameterize an existing processing action. If no implementation of the mathematical model for the required processing action exists yet, it can be added via a new subclass of the interface class *model*.

CONCLUSIONS

Fan test rigs are evolving systems subject to continuous improvements. A lack of uniform but flexible data structures and machine actionable metadata imposes cumbersome and frequent adjustments of data storage schemes as well as data acquisition and processing software. The presented data model and software design reduce the proliferation of resulting scheme and software versions that typically accompanies the improvements of the instrumentation setup.

Measurement data can be linked to extensive descriptive information with relatively low effort, facilitating the collective long-term usability of heterogeneous datasets. The software interface allows integrating the creation of appropriate data structures and metadata into the data acquisition software of operational test rigs. Likewise, migrating existing data sets is possible. Processing of raw data is streamlined by means of object oriented programming, leveraging the stored machine actionable metadata to configure a modular object structure. The derived data is linked to the origin raw data, providing full traceability of data processing actions. As a result, the need for customized post-processing software per test configuration can be limited. Most individual computational analysis can thus be built upon the process variables.

Ease of use is achieved as a result of the pure hierarchic data model and its file representation, keeping the complexity low. HDF5 as the underlying file format and the corresponding libraries provide the capabilities to explore the file contents programmatically as well as random access to the file contents. A user can browse a file using the available editor or write a reader for the file format in any programming language and operating system of choice. The hierarchic model has proven suitable for an interface implementation based on by-value classes. The resulting object structures can be handled like usual data structures in MATLAB or LabVIEW and thus integrate very well with existing data acquisition and processing software.

Future work will focus on the integration of the data management functionality with data acquisition driver interfaces, further increasing the potential for code re-use. A by-reference implementation will open up the potential for more advanced functionality, which is closely interlinked with expanding the hierarchic data model to a directed acyclic graph structure. To unify data management across different research projects, a standard vocabulary and basic information requirements must be established.

ACKNOWLEDGEMENTS

The authors would like to thank the German Research Foundation (DFG) for funding this research within the Collaborative Research Centre (SFB) 805 ‘Control of uncertainty in load carrying structures in mechanical engineering’ (TU Darmstadt), speaker Prof. Dr.-Ing. Peter F. Pelz).

BIBLIOGRAPHY

[1] J. Ludwig, *Leitfaden zum Forschungsdaten-Management*. **2013**.
[2] Deutsche Forschungsgemeinschaft e.v, "Leitlinien zum Umgang mit Forschungsdaten," **2015**.
[3] V. Turner and J. F. Gantz. (2014, November 17). *The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things*. Available: https://www.emc.com/leadership/digital-universe/2014iview/digital-universe-of-opportunities-vernon-turner.htm

[4] W. K. Michener, J. W. Brunt, J. J. Helly, T. B. Kirchner, and S. G. Stafford, "NONGEOSPATIAL METADATA FOR THE ECOLOGICAL SCIENCES," *Ecological Applications,* vol. 7, no. 1, pp. 330-342, **1997**.

[5] W. Benger, "On Safari in the File Format Jungle-Why Can't You Visualize My Data?," (in English), *Computing in Science & Engineering,* vol. 11, no. 6, pp. 98-102, **2009**.

[6] R. McClatchey, Z. Kovacs, F. Estrella, J. M. L. Goff, L. Varga, and M. Zsenei, "The role of meta-objects and self-description in an engineering data warehouse," in *Database Engineering and Applications, 1999. IDEAS '99. International Symposium Proceedings*, pp. 342-350, **1999**.

[7] Digital preservation services development group, "Research Data File Formats and Digital Preservation - Final Report," The Open Science and Research Initiative (ATT), **2017**.

[8] Z. Chen, D. Wu, J. Lu, and Y. Chen, "Metadata-based Information Resource Integration for Research Management," *Procedia Computer Science,* vol. 17, pp. 54-61, **2013**.

[9] M. Greenwald, T. Fredian, D. Schissel, and J. Stillerman, "A metadata catalog for organization and systemization of fusion simulation data," (in English), *Fusion Engineering and Design,* vol. 87, no. 12, pp. 2205-2208, **2012**.

[10] The HDF Group. (2017, October 30). *HDF5 Users*. Available: https://support.hdfgroup.org/HDF5/users5.html

[11] The HDF Group. (2017, October 30). *Summary of Software Using HDF5*. Available: https://support.hdfgroup.org/products/hdf5_tools/SWSummarybyName.htm

[12] The HDF Group. (2017, October 30). *HDF5 User's Guide HDF5 Release 1.10.0*. Available: https://support.hdfgroup.org/HDF5/doc/UG/HDF5_Users_Guide-Responsive HTML5/index.html

[13] The HDF Group. (2017, October 30). *HDFView Homepage*. Available: https://support.hdfgroup.org/products/java/hdfview/

[14] The HDF Group. (2017, November 17). *HDF5 XML Information Page*. Available: https://support.hdfgroup.org/HDF5/XML/

[15] The HDF Group. (2017, November 17). *HDF5/JSON documentation*. Available: http://hdf5-json.readthedocs.io/en/latest/

[16] A. Pfeiffer, I. Bausch-Gall, and M. Otter, "Proposal for a Standard Time Series File Format in HDF5," presented at the 9th International Modelica Conference, Munich, Germany, **2012**.

[17] R. E. Ullman, "HDF-EOS, NASA's standard data product distribution format for the Earth Observing System data information system," in *Geoscience and Remote Sensing Symposium, 1999. IGARSS '99 Proceedings. IEEE 1999 International*, vol. 1, pp. 276-278 vol.1, **1999**.

[18] A. Ingargiola, T. Laurence, R. Boutelle, S. Weiss, and X. Michalet, "Photon-HDF5: An Open File Format for Timestamp-Based Single-Molecule Fluorescence Experiments," *Biophysical Journal,* vol. 110, no. 1, pp. 26-33, **2016**.

[19] M. T. Dougherty *et al.*, "Unifying Biological Image Formats with HDF5," *Communications of the ACM,* vol. 52, no. 10, pp. 42-47, **2009**.

[20] D. C. Price, B. R. Barsdell, and L. J. Greenhill, "HDFITS: Porting the FITS data model to HDF5," *Astronomy and Computing,* vol. 12, pp. 212-220, **2015**.

[21] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*. **1994**.

[22] M. Heß, "Aufwertung bei Axialventilatoren – Einfluss von Reynolds-Zahl, Rauheit, Spalt und Betriebspunkt auf Wirkungsgrad und Druckziffer," Dissertation (PhD), Institut für Fluidsystemtechnik, TU Darmstadt, **2010**.

[23] S. Stonjek, "Wirkungsgradaufwertung bei Ventiolatoren," Dissertation (PhD), Institut für Fluidsystemtechnik, TU Darmstadt, Aachen, **2016**.

[24] S. Saul and P. F. Pelz, "Influence of Compressibility on Fan Efficiency and Fan Efficiency Scaling," presented at the 3rd International Rotating Equipment Conference (IREC) Pumps, Compressors and Vacuum Technology, Duesseldorf, Germany, **2016**.