DEVELOPMENT OF A SOFTWARE LIBRARY FOR PERFORMANT AND CONSISTENT CPACS DATA PROCESSING

M. Alder*, A. Skopnik†

*German Aerospace Center (DLR), Institute of System Architectures in Aeronautics, 21129 Hamburg, Germany
†TU Chemnitz, 09111 Chemnitz, Germany

Abstract

The Common Parametric Aircraft Configuration Schema (CPACS) is increasingly used as a standard for data exchange in collaborative aircraft design projects involving many heterogeneous disciplines and expert knowledge [1,2]. CPACS provides a hierarchical parametrization of fixed-wing aircraft and rotorcraft spanning from a detailed component level up to the interaction of the vehicle and its peripheral aviation system such as airline operations. Following the CPACS paradigm - data must be unique and explicit - all parameters are uniquely specified using an XML Schema Definition (XSD) which allows for a robust syntactic interpretation of the data. The transformation of this data (e.g., unit transformations of physical quantities) or the inference of additional information (e.g., interpolation of aerodynamic coefficients) usually requires rules and assumptions which are specified in a human interpretable format within the CPACS documentation. However, practical experience shows that even whilst having comprehensive documentation available, there is a potential source for inconsistency if such rules and assumptions are not available as standardized software implementations. While the TiGL geometry library serves as common software library for geometry data, other disciplines are lacking standardized approaches on how to infer knowledge from CPACS data. The present paper therefore introduces a software library called cpacsLibrary aiming to ensure consistent data handling in large collaborative aircraft design projects, and to enable an easier entrance into connecting models to CPACS as well as re-using implementations across disciplines. As it is tightly coupled to the development of CPACS itself and closely aligned with TiGL, cpacsLibrary intends to complete the CPACS eco-system by providing standardized methods for non-geometric data handling. A detailed description of the software architecture comprises the implementation of low- and high-level methodologies in C++, test-driven development, version control, bindings to Python as well as data visualization strategies. The practical application of the library is demonstrated by the interpolation of irregular, multidimensional performance maps (aerodynamic and engine) stemming from disciplinary analysis tools in automated aircraft design processes applying Radial Basis Functions (RBF). The paper concludes by describing the future development roadmap, including opportunities for collaboration.

Keywords

CPACS; Software Library; RBF interpolation; Aerodynamic Performance Maps

1. BRIEF INTRODUCTION TO CPACS

The Common Parametric Aircraft Configuration Schema CPACS is a standardized data model for air transportation systems. CPACS has been developed since 2005 to meet the challenges of collaborative design projects [1]. One is to reduce the amount of possible interfaces between N disciplines from $N(N-1)$ to $2N$ by introducing a central data source (see Fig. 1). A second challenge is to ensure a consistent data transfer between the very heterogeneous expert domains. CPACS tries to tackle these challenges by establishing a common language for aircraft design via the Extensible Markup Language (XML), making it both human and machine readable. The underlying XML Schema Definition (XSD) allows to model complex structural and semantic rules. Strengths and weaknesses of this approach will be discussed in Sec. 1.1.

Starting as a development within the German Aerospace Center e.V. (DLR), CPACS has become a community

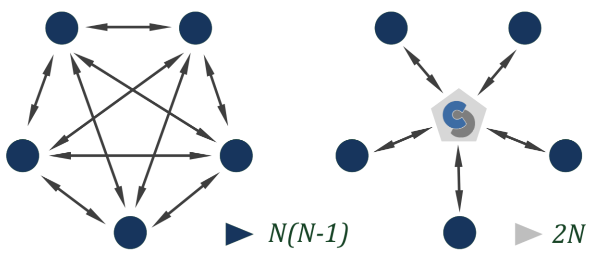
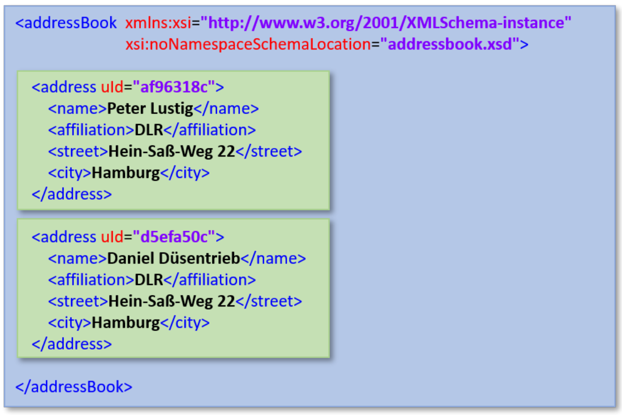


FIG 1. The amount of possible data interfaces reduces from $N(N-1)$ to $2N$ by using a central data source.



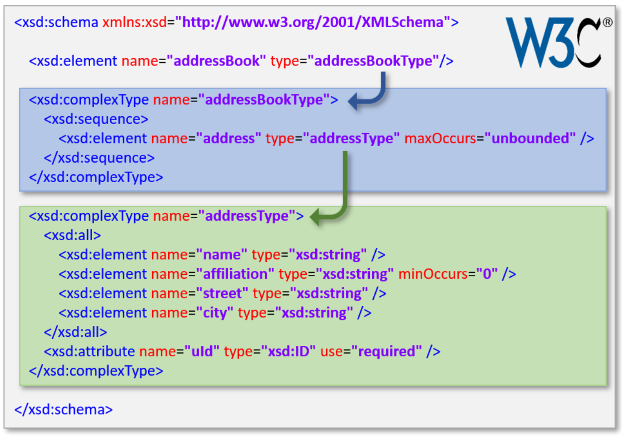
```
<addressBook xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:noNamespaceSchemaLocation="addressbook.xsd">

  <address uId="af96318c">
    <name>Peter Lustig</name>
    <affiliation>DLR</affiliation>
    <street>Hein-Saß-Weg 22</street>
    <city>Hamburg</city>
  </address>

  <address uId="d5efa50c">
    <name>Daniel Düsentrieb</name>
    <affiliation>DLR</affiliation>
    <street>Hein-Saß-Weg 22</street>
    <city>Hamburg</city>
  </address>

</addressBook>
```

FIG 2. Example of XML file containing address data.



```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="addressBook" type="addressBookType"/>

  <xsd:complexType name="addressBookType">
    <xsd:sequence>
      <xsd:element name="address" type="addressType" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="addressType">
    <xsd:all>
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="affiliation" type="xsd:string" minOccurs="0" />
      <xsd:element name="street" type="xsd:string" />
      <xsd:element name="city" type="xsd:string" />
    </xsd:all>
    <xsd:attribute name="uId" type="xsd:ID" use="required" />
  </xsd:complexType>

</xsd:schema>
```

FIG 3. Example of a data model for an address book in XSD (*Note: If not specified,* `minOccurs` *and* `maxOccurs` *equals 1.*)

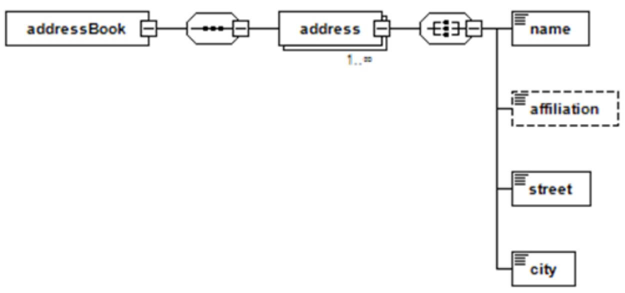


FIG 4. XSD diagram representation of an address book data model.

project involving universities, research institutes and industry, which drives the continuous development, testing and application of the data model in practice [2]. Section 1.2 presents some application examples within current research projects. Based on this, requirements and strategies for consistent handling of data are derived in Sec. 1.3.

1.1. Data Modeling with XSD

First, the terminology used in the following discussions should be clarified. In computer science, a distinction is made between data models and data formats. A data model is an abstract model that organizes elements of data and standardizes how they relate to one another and to the properties of real-world entities. A data format specifies the representation of data in terms of bits and bytes [3]. The actual development of CPACS itself is about developing an explicit and unique data structure to describe air transportation systems in all its facets (e.g. geometry and material properties, analysis results of flight characteristics, etc.). CPACS therefore constitutes a data model. For the practical realization XML is chosen as data format.
XML provides XML Schema Definition (XSD) to describe the data model. XSD itself is an XML document specifying data in terms of elements, their relation to each other, their data type and occurrence. In the following, only the basic foundations which are required for the understanding of the present papers' content will be discussed. An an extensive literature on XML/XSD is available for further reading [4–6].
A simple representative example shall illustrate how to develop a data model like CPACS from a technical perspective. The goal is to store addresses in an address book, each specifying `name`, `affiliation`, `street` and `city` of a person (see Fig. 2). In XML it is common practice to use the so called camel case style capitalizing the first letter of the second and all following words, thus spelling `addressBook`. So, the data model shall describe an `addressBook` with multiple `addresses` each containing the above mentioned address data. Figure 2 constitutes an instance of this model, i.e. the actual data stored in an XML file.

Figure 3 shows an implementation the corresponding data model with XSD. The syntax follows a standard from the World Wide Web Consortium (W3C) [7] which is included by the `XMLSchema` namespace and in this example referred to via the prefix `xsd`. Each data is represented by an `element` and a corresponding `type`. Such type is called *simple* if it describes elementary data, for example `string` or `double` values. If an element consists of child elements, then its type is *complex* (`xsd:complexType`). In this way, a hierarchical structure can be easily set up by defining elements which either have complex or simple types. Another important property of elements is how often they may occur in a certain position of the data tree, defined by minimum and maximum occurence (`minOccurs` and `maxOccurs`) with a value of `1` being the default setting. The order of the elements can be set arbitrarily (`xsd:all`) or strictly predefined (`xsd:sequence`). Finally, elements can be enhanced by attributes (`xsd:attribute`), which can only be defined as simple data types. Attributes are usually used to describe meta information, such as unique identification keys (`xsd:ID`) which might be used as a reference for linking (via `xsd:IDREF`).
For easier communication of data models, for example during discussions with stakeholders or to illustrate new developments, an abstraction of the XSD syntax via so-called XSD diagrams can be used. An XSD diagram of the above address book example is shown in Fig. 4. Elements are represented by rectangles in a hierarchically tree-view. The border style indicates their occurence:

A solid line corresponds to single occurring elements, dashed lines represent optional elements and staggered borders indicate element repetitions.
These are the basics which are needed to understand the implementation of the CPACS data model in XSD and how the development of corresponding software libraries can benefit from this. For additional information on XSD techniques used for CPACS, such as inference, extensions or restrictions of types, the authors refer to the available literature [4–6].

1.2. CPACS Current Status and Applications

A comprehensive overview on recent developments in CPACS is given in a previous publication by the author [2]. Currently CPACS is available in version 3.4. An an excerpt from the corresponding XSD diagram is shown in Fig. 11. Among others, current developments focus on the different facets of aircraft systems, including the geometric, material, functional and physical properties of the various sub-systems and components and how they interact. Also topics like weight and balance descriptions for vehicles with in-flight configurational changes (e.g., military airplanes or rescue helicopters) require extensions of the CPACS data model.
This is reflected in current research projects where CPACS is employed as a central data exchange model. A small selection of projects illustrates that CPACS is currently strongly driven by topics concerning environmental sustainability and safety:

EXACT

Starting in 2020, forty-five researchers from twenty DLR institutes have been working together on the Exploration of Electric Aircraft Concepts and Technologies (EXACT) project, which is developing new technological components for an environmentally friendly commercial aircraft [8, 9]. Here, CPACS plays a crucial role as the central data exchange model and is extended with regard to novel aircraft system architectures, e.g. power/energy breakdowns, enhancements of the mass breakdown, definition of liquid hydrogen tanks or electric propulsion architectures.

IMOTHEP

The core of a project on the Investigation and Maturation of Technologies for Hybrid Electric Propulsion (IMOTHEP) is an integrated end-to-end investigation of hybrid-electric power trains for commercial aircraft, performed in close connection with the propulsion system and aircraft architecture. This EU granted project (Horizon 2020) is running four years and is supported by seven research institutes, eleven industries (from aviation and electric systems), a service SME and seven universities from nine EU countries [10]. CPACS is used for the design of regional aircraft configurations and recent development work was devoted to the question of how to specify complex propulsion system architectures and corresponding assumptions.

AGILE 4.0

AGILE 4.0 (Aircraft 3rd Generation MDO for Innovative Collaboration of Heterogeneous Teams of Experts) targets the digital transformation of the aeronautical supply-chain: design, production and certification and manufacturing. The three-year project involves sixteen partners from eight countries, involving universities, research institutions and industry [11–13]. AGILE project members developed concepts for the geometric and mass description of system components, among others for space allocation analyses.

DIABOLO

The DLR-project Diabolo investigates the multidisciplinary design of both an unmanned military vehicle (MULDICON) and the DLR Future Fighter Demonstrator (FFD) to demonstrate DLR's design capability and to further-develop the required key technologies and design procedures. The project involves eleven DLR institutes and facilities as well as the industrial partners Airbus Defence and Space, MTU Aero Engines and German-Dutch Wind Tunnels (DNW) [14]. In this context, CPACS faces the challenge of representing in-flight configurational changes, such as air-to-air refueling or military store releases, in terms of weight and balance changes and the resulting flight characteristics.

CHASER

The three-year CHASER project involves nine DLR institutes and investigates the design process of highly flexible and fast helicopter configurations, such as those used in medical rescue [15]. One of the challenges for CPACS here, similar to the Diabolo project, is how to map the configurational flexibility of rotorcraft to the weight and balance characteristics, for example in mission definitions that require the repeated delivery of emergency physicians to accident sites and the evacuation of patients.

1.3. Challenges of Consistent Data Processing

The main philosophy behind CPACS is that data is explicit and unique. The latter means that information must be unambiguous in order to ensure the consistency of a data set, for example after modifications or additions to the data. Explicitness is comparatively easy to achieve through well-chosen element names, but from practical experience it is a continuous search for the appropriate balance between unambiguous description of data and flexibility of use. The following question illustrates this: Should the wing of an aircraft be referred to as `mainWing`, `horizontalTailplane`, `verticalTailplane`, and so on, or just as generic `wing`? The latter allows high flexibility in the interpretation and therefore enables unconventional configurations (e.g., boxed-wing airplanes), but at the same time poses the risk of misinterpretation when evaluating data sets. A rule that the main wing is always the largest one is implicit and might not always apply. Modern knowledge-based engineering techniques attempt to enrich data with such information to ensure correct

mutual understanding of knowledge, but the above example shows that the problem is more fundamental and not easily solvable when using a central data model in an interdisciplinary environment.
Another challenge is how to derive additional data from existing CPACS parameters. Again, the wing provides a good example where individual wing sections are explicitly defined in the data set by airfoil point coordinates. In the next step, robust geometry operations are required to determine the three-dimensional lofting of the wing skin. CPACS does not specify which mathematical methods are to be used for lofting. In multidisciplinary projects, therefore, a common approach to process CPACS data should be agreed upon. A software library called *TiGL Geometry Library* (TiGL) is typically used to translate the CPACS parameters into three-dimensional surfaces and thus ensures that all project partners are working on the base of the same geometry [16].
To summarize, the two main challenges when applying CPACS in multidisciplinary design activities are: (1) the communication and consistent application of implicit rules on how to process data and (2) the derivation of additional information from existing data, e.g. through interpolation.

2. CPACSLIBRARY: DESIGN AND IMPLEMENTATION

While TiGL has established as the primary software library for geometric operations on CPACS data, an equivalent solution is missing for other disciplines, such as aerodynamics or flight performance analysis. The following sections therefore introduce a new software library called cpacsLibrary. The focus of cpacsLibrary is to address the challenges when working with CPACS as outlined in Sec. 1.2, i.e. implement rules on how to handle CPACS data (which are currently only written in the documentation) and to provide interpolation algorithms for data other than geometry.
The present study focuses particularly on the conceptual design of a robust and sustainable software architecture. Therefore, the Software Engineering Guidelines of the DLR Institute for Software Technology are applied, providing recommendations for requirements management, software architecture, design and implementation, change management, software test, release management and more [17].

2.1. Software Requirements

The requirements of the cpacsLibrary are derived from a stakeholder analysis and their intentions to use the software. Stakeholders will primarily be actual CPACS users who need robust, performant and simple access to the data and the information derived from it. Practical experience with similar use-cases has already been gained with TiGL. From this, it is known that typical users need to integrate the library into different programming languages, of which Python is most frequently used. Furthermore, C, C++ as well as Matlab are actively used by stakeholders. CpacsLibrary should therefore provide interfaces to different programming languages. A typical user-story would for example be:

> [**User Story 1**] As a tool developer I want to integrate cpacsLibrary into my C++ tool to receive aerodynamic lift and drag coefficients for a given set of Mach number, altitude and angle of attack to analyze flight performance.

or

> [**User Story 2**] As a data analyst I want to use cpacsLibrary in Python to extract and evaluate the flight trajectories, e.g. by plotting the flight altitude versus the flight distance.

While the first user story is a request on a higher level of data interpretation (i.e., aerodynamic coefficients must be interpolated from existing data, which requires knowledge beyond what is defined in XSD), the second user story is a request on a lower level of interpretation as only the existence of values needs to be checked and then returned without further manipulation. In this case the interface must also be accessible via Python.
Another requirement stemming from CPACS users is that methods must be available to create or modify CPACS data:

> [**User Story 3**] As a CPACS user I want to create a new CPACS file from scratch.

> [**User Story 4**] As a CPACS user I want to change the aspect ratio of an existing wing definition.

The third user story requires that cpacsLibrary is able to write XML data according to the CPACS specification (i.e., XSD). The fourth user story aims at modifying existing CPACS data, as the aspect ratio is not directly available in CPACS but must be derived from the wing parametrization. This requires explicit rules on how to change the corresponding wing sections and how to treat interdependent data, such as internal structures. The user must be informed on what data will be affected by the changes.
Furthermore, an important use-case often experienced in practice is represented by the following user story:

> [**User Story 5**] As a CPACS user I received a CPACS file and I want to quickly inspect its content without scripting or programming.

In summary, the following requirements are stemming from the above user-stories:

1) The cpacsLibrary must be able to extract data from a given CPACS file.
2) The cpacsLibrary must be able to modify existing CPACS data.
3) The cpacsLibrary must be able generate new CPACS data.
4) The cpacsLibrary must provide interpolation routines for data in multi-dimensional parameter spaces.
5) The cpacsLibrary must allow for easy inspection of CPACS data.

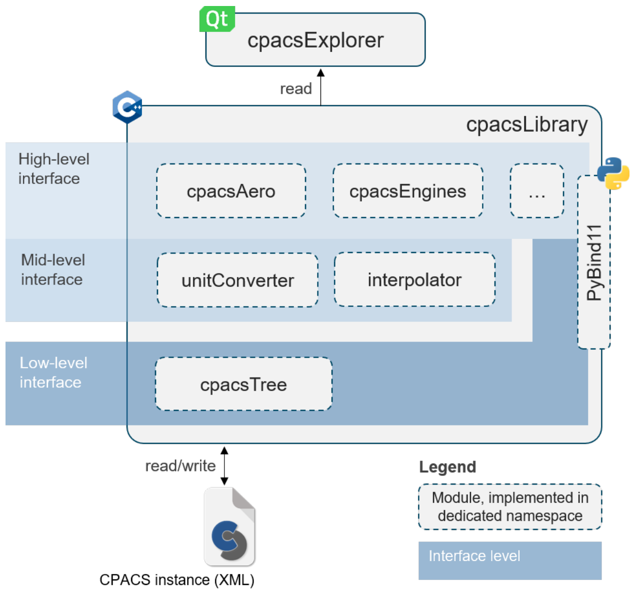


FIG 5. Schematic representation of the software architecture of cpacsLibrary.

6) The cpacsLibrary must provide interfaces to different programming languages

2.2. Software Architecture

Considering the requirements derived in the previous section, a software architecture has been developed as shown in Fig. 5. It is composed of different interface levels:

- A low-level interface, called *cpacsTree*
- A mid-level interface, called *cpacsEvaluator*
- A high-level interface containing different disciplinary modules (e.g., *cpacsAero* or *cpacsMissions*)

The purpose of the **low-level interface** is to convert CPACS data, which is imported from an XML file, into an object oriented structure which can be accessed in-memory. At this level, no additional knowledge is applied to the data than what is defined in XSD.

The **high-level interface** uses data from the low-level interface and adds CPACS-specific knowledge. Such knowledge is usually discipline-specific and therefore en-capsuled into disciplinary modules. One example here is that CPACS defines aerodynamic maps in terms of dimensionless coefficients in the aerodynamic coordinate system. Currently, this information is not specified in XSD, but only in the human-readable documentation. To avoid the risk of errors during user-specific data processing, the knowledge must be available in the corresponding cpacsAero module. This allows for consistent unit conversions, coordinate transformations and other disciplinary operations. The goal of the disciplinary modules is thus to enable a view on CPACS data through the glasses of disciplinary experts.

In addition to the previously mentioned interfaces, a **mid-level interface** is implemented. Its task is to evaluate data via generic algorithms that cannot be assigned to specific disciplines, but must contain knowledge that goes beyond the low-level interface. A typical example is interpolation algorithms, which are employed for both aerodynamic and engine performance maps.

An additional Graphical User Interface (GUI), called *cpacsExplorer*, uses data from the low- and high-level interface for easy data inspection and visualization. An essential role of this component is extensive testing and debugging in practice.

Finally, the low- and high-level interfaces should be accessible via an *Application Programming Interface* (API) for different programming languages. This is not intended to be the case for the mid-level interface, as this is only for internal usage.

2.3. Implementation

The programming language C++ (version 11) has been chosen for the implementation of cpacsLibrary. This choice is based on balancing the following pro- and contra-arguments:

Arguments for using C++:

- Enables a close collaboration with TiGL development team (e.g., lessons learned, mutual support, interoperability of both software libraries)
- Existing third-party software and code-base already available in C/C++, such as the XML Interface *TiXI* [18], the *CPACSGen* software for automatic generation of C++ classes from XSD [19] and Exception handling methods [20]
- Good performance of compiled C++ code
- Established programming language with a large community
- No technical limitations expected
- Possibility to provide interfaces for many other programming languages (e.g., via pybind11 [21] or Swig [22])

Arguments against using C++:

- Entry barrier for new developers is higher than, for example, with Python or Matlab. This could slow down the development process and make community building challenging.
- Maintenance is rather difficult due to the complexity of C++ and availability of experienced developers compared to, e.g., Python.

The advantages of using C++ outweigh the contra-arguments here, which is why this programming language is chosen for the implementation of cpacsLibrary.

2.3.1. Low-Level Interface

The task of the low-level interface is primarily to import data from a CPACS file and make it accessible as in-memory data for efficient further processing. Section 1.1 has shown that each data object is defined via XSD in terms of element or attribute names, its occurrence, order of appearance and the corresponding data type. This can be used to automatically generate C++ code providing dedicated methods for the various data types according to the schema. Thus, an individual class could be created for each complex XSD type with corresponding getter and setter functions for its child elements and attributes. For this purpose, *CPACS generator* (CPACSGen), which is developed by RISC Software GmbH [19] to provide the code base for TiGL, is adopted for cpacsLibrary. The generated code make use of a TiXI instance to read and write data from and to a CPACS file. The generated code is compliant with C++ 11 and 14 and depends on Boost libraries. The automatic generation of C++ code from XSD furthermore lowers the effort to update cpacsLibrary when new CPACS versions are released.

An often-used feature in CPACS is that elements of type `xsd:idref` can reference other elements via `uID` attributes of type `xsd:id`. TiGL manages these dependencies via a so-called `uID`-manager. This concept has been adopted for cpacsLibrary, although it slightly differs as some geometric properties are not applicable for all CPACS elements. An idea resulting from discussions between developers of CPACSGen, TiGL and cpacsLibrary is to implement a rather general `uID`-manager as part of CPACSGen. This demonstrates well the advantage of aligning the developments of cpacsLibrary and TiGL.

2.3.2. Mid-Level Interface

The mid-level interface is internally referred to as cpacsEvaluator. Here, for example, interpolation routines are implemented, which contain rules and knowledge about the processing of CPACS data that go beyond the XSD schema, but cannot and should not be assigned to any single discipline. Section 3 introduces in detail the implementation of scattered data interpolation methods for multidimensional maps, which can be employed for aerodynamic performance maps as well as engine maps. Another use-case for cpacsEvaluator is performing unit or coordinate transformations.

2.3.3. High-Level interface

Finally, the high-level interface represents an enrichment of the pure CPACS data with domain-specific knowledge for consistent and robust processing of such. Two use-cases are selected for the implementation to demonstrate the concept. The first example refers to multidimensional maps of aerodynamic coefficients in CPACS, which are called `aeroPerformanceMaps` (see Fig. 12). While the CPACS structure must be followed exactly using the low-level interface (see Listing 1), the high-level interface can deviate from this and directly return the lift coefficient parameter `cl` as shown in Listing 2. This is a conceptually different handling on the same data and decouples the complex structure of the data model from the interface design. Adding domain-specific knowledge to cpacsLibrary therefore allows for both a more user-friendly way of working with CPACS and for a higher flexibility to the design of the data model itself. The first implementation furthermore contains unit conversion and the possibility to filter values with respect to `altitude`, `machNumber`, `angleOfAttack` and `angleOfSideSlip`.

Another implementation concerns flight performance analysis in order to investigate how to visualize time-dependent data and how to treat redundancy, which is avoided as far as possible in CPACS, but still exists for some cases. Part of flight performance analysis is flight trajectories resulting from mission analysis tools. The overall flight distance can both be drawn from the time-dependent `flightPoints/groundDistance` vector or from the `global/distance` parameter. It is of course convenient to directly find such evaluated parameters in CPACS, but this example is an obvious source of inconsistencies. Furthermore, the user must study the CPACS documentation thoroughly to find out whether `global/distance` refers to the flight distance or ground distance. Both approaches are implemented and will in future be used to further investigate strategies on how to remove such redundancy from CPACS while providing evaluation methodologies with cpacsLibrary.

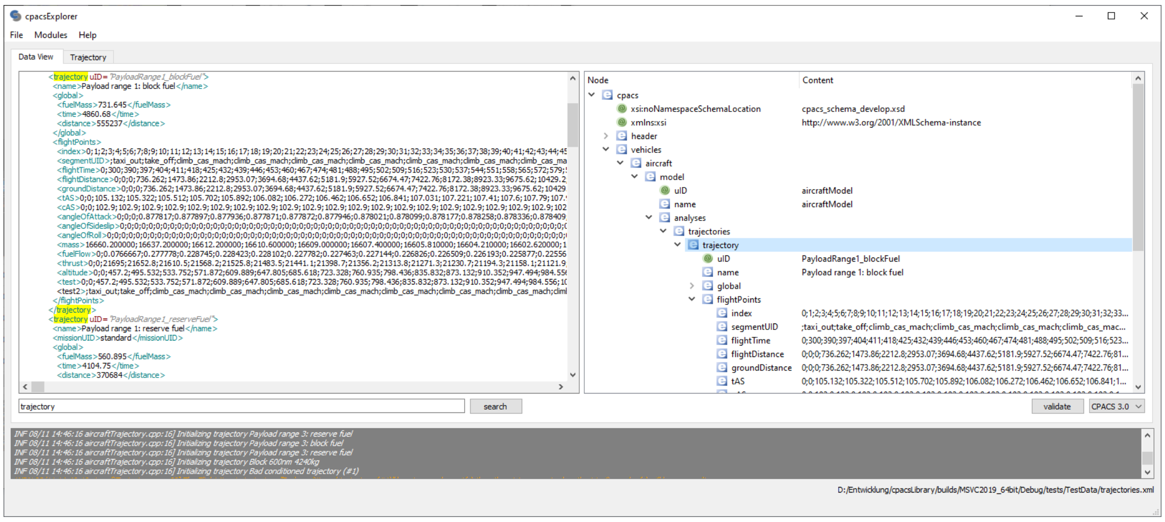
Implementing the above examples is a good starting point to test the concept of cpacsLibrary in practice. According to the different disciplines it is subdivided into dedicated modules defined in individual name-spaces (i.e., declarative sections of code which are used to avoid name collisions). This increases flexibility in the development as, for example, students or disciplinary research groups can contribute with their own code en-capsuled in modules. Common evaluation routines, such as interpolation algorithms, are imported from the mid-level interface. This concept is further illustrated in Sec. 3.

2.4. Data Inspection with cpacsExplorer

On top of cpacsLibrary a GUI enables fast inspection of CPACS data (see User Story 5; requirement 6). It furthermore simplifies testing the implementations in practice and therefore supports the identification and debugging of errors. The GUI has been named cpacsExplorer. The prototype implementation is based on the Qt framework. This has the advantage of developing both the library and the GUI in the same programming language and providing them as a cohesive package. Thus, errors can be traced directly from the interface to the corresponding methods in cpacsLibrary.

A basic feature of cpacsExplorer is the visualization of the CPACS data in the form of a typical text-editor representation as well as a graphical representation of the tree structure. This is already familiar to many CPACS users from the visualization of CPACS data in the process integration framework RCE [23]. Both views are linked via event listeners to facilitate navigation through a CPACS file (see Fig. 6).

Additional tabs can be added to the main view to present domain-specific data. In the Mission module, trajectories can be evaluated and visualized as line charts and in tabular form (see Fig. 7). Another example is the visualization of multidimensional aerodynamic maps. By selecting two input variables (two out of `machNumber`, `altitude`, `angleOfAttack` and `angleOfSideslip`), the aerodynamic coefficients are projected into a two-dimensional scatter plot and also shown in tabular form. Accessing the filter algorithms of the high-level interface further supports data inspection.



```
<trajectory uID="PayloadRange1_blockFuel">
 <name>Payload range 1: block fuel</name>
 <global>
  <fuelMass>731.645</fuelMass>
  <time>4860.68</time>
  <distance>555237</distance>
 </global>
 <flightPoints>
  <index>0;1;2;3;4;5;6;7;8;9;10;11;12;13;14;15;16;17;18;19;20;21;22;23;24;25;26;27;28;29;30;31;32;33;34;35;36;37;38;39;40;41;42;43;44;45
  <segmentUID>;taxi_out;take_off;climb_cas_mach;climb_cas_mach;climb_cas_mach;climb_cas_mach;climb_cas_mach;climb_cas_mach;climb_cas_ma
  <flightTime>0;300;390;397;404;411;418;425;432;439;446;453;460;467;474;481;488;495;502;509;516;523;530;537;544;551;558;565;572;579;
  <flightDistance>0;0;0;736.262;1473.86;2212.8;2953.07;3694.68;4437.62;5181.9;5927.52;6674.47;7422.76;8172.38;8923.33;9675.62;10429.2
  <groundDistance>0;0;0;736.262;1473.86;2212.8;2953.07;3694.68;4437.62;5181.9;5927.52;6674.47;7422.76;8172.38;8923.33;9675.62;10429
  <tAS>0;0;105.132;105.322;105.512;105.702;105.892;106.082;106.272;106.462;106.652;106.841;107.031;107.221;107.41;107.6;107.79;107.
  <cAS>0;0;102.9;102.9;102.9;102.9;102.9;102.9;102.9;102.9;102.9;102.9;102.9;102.9;102.9;102.9;102.9;102.9;102.9;102.9;102
  <angleOfAttack>0;0;0;0.877817;0.877897;0.877936;0.877871;0.877872;0.877946;0.878021;0.878099;0.878177;0.878258;0.878336;0.878409
  <angleOfSideslip>0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
  <angleOfRoll>0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
  <mass>16660.200000;16637.200000;16612.200000;16610.600000;16609.000000;16607.400000;16605.810000;16604.210000;16602.620000;1
  <fuelFlow>0;0.0766667;0.277778;0.228745;0.228423;0.228102;0.227782;0.227463;0.227144;0.226826;0.226509;0.226193;0.225877;0.22556
  <thrust>0;0;21695;21652.8;21610.5;21568.2;21525.8;21483.5;21441.1;21398.7;21356.2;21313.8;21271.3;21230.7;21194.3;21158.1;21121.9
  <altitude>0;0;457.2;495.532;533.752;571.872;609.889;647.805;685.618;723.328;760.935;798.436;835.832;873.132;910.352;947.494;984.55
  <test>0;0;457.2;495.532;533.752;571.872;609.889;647.805;685.618;723.328;760.935;798.436;835.832;873.132;910.352;947.494;984.556;1
  <test2>;taxi_out;take_off;climb_cas_mach;climb_cas_mach;climb_cas_mach;climb_cas_mach;climb_cas_mach;climb_cas_mach;climb_cas_mach;climl
 </flightPoints>
</trajectory>
<trajectory uID="PayloadRange1_reserveFuel">
 <name>Payload range 1: reserve fuel</name>
 <missionUID>standard</missionUID>
 <global>
  <fuelMass>560.895</fuelMass>
  <time>4104.75</time>
  <distance>370684</distance>
```

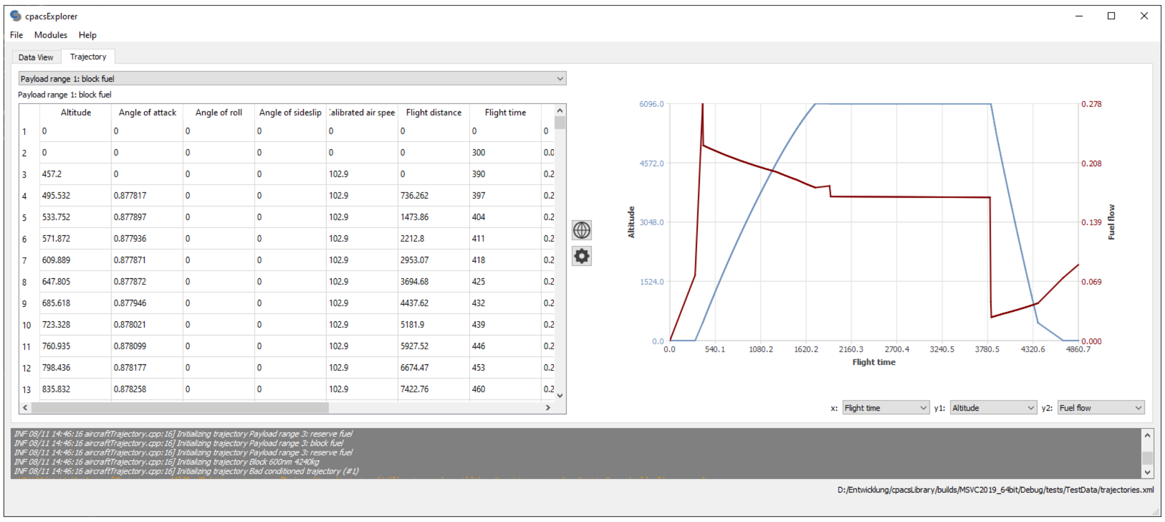
trajectory

Node	Content
cpacs	
xsi:noNamespaceSchemaLocation	cpacs_schema_develop.xsd
xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
header	
vehicles	
aircraft	
model	
uID	aircraftModel
name	aircraftModel
analyses	
trajectories	
trajectory	
uID	PayloadRange1_blockFuel
name	Payload range 1: block fuel
global	
flightPoints	
index	0;1;2;3;4;5;6;7;8;9;10;11;12;13;14;15;16;17;18;19;20;21;22;23;24;25;26;27;28;29;30;31;32;33...
segmentUID	;taxi_out;take_off;climb_cas_mach;climb_cas_mach;climb_cas_mach;climb_cas_mac...
flightTime	0;300;390;397;404;411;418;425;432;439;446;453;460;467;474;481;488;495;502;509;516;523...
flightDistance	0;0;0;736.262;1473.86;2212.8;2953.07;3694.68;4437.62;5181.9;5927.52;6674.47;7422.76;81...
groundDistance	0;0;0;736.262;1473.86;2212.8;2953.07;3694.68;4437.62;5181.9;5927.52;6674.47;7422.76;81...
tAS	0;0;105.132;105.322;105.512;105.702;105.892;106.082;106.272;106.462;106.652;106.841;1...

```
INF 08/11 14:46:16 aircraftTrajectory.cpp:16] Initializing trajectory Payload range 3: reserve fuel
INF 08/11 14:46:16 aircraftTrajectory.cpp:16] Initializing trajectory Payload range 3: block fuel
INF 08/11 14:46:16 aircraftTrajectory.cpp:16] Initializing trajectory Payload range 3: reserve fuel
INF 08/11 14:46:16 aircraftTrajectory.cpp:16] Initializing trajectory Block 600nm 4240kg
INF 08/11 14:46:16 aircraftTrajectory.cpp:16] Initializing trajectory Bad conditioned trajectory (#1)
```

D:/Entwicklung/cpacsLibrary/builds/MSVC2019_64bit/Debug/tests/TestData/trajectories.xml

FIG 6. Tree-view of CPACS data in cpacsExplorer.



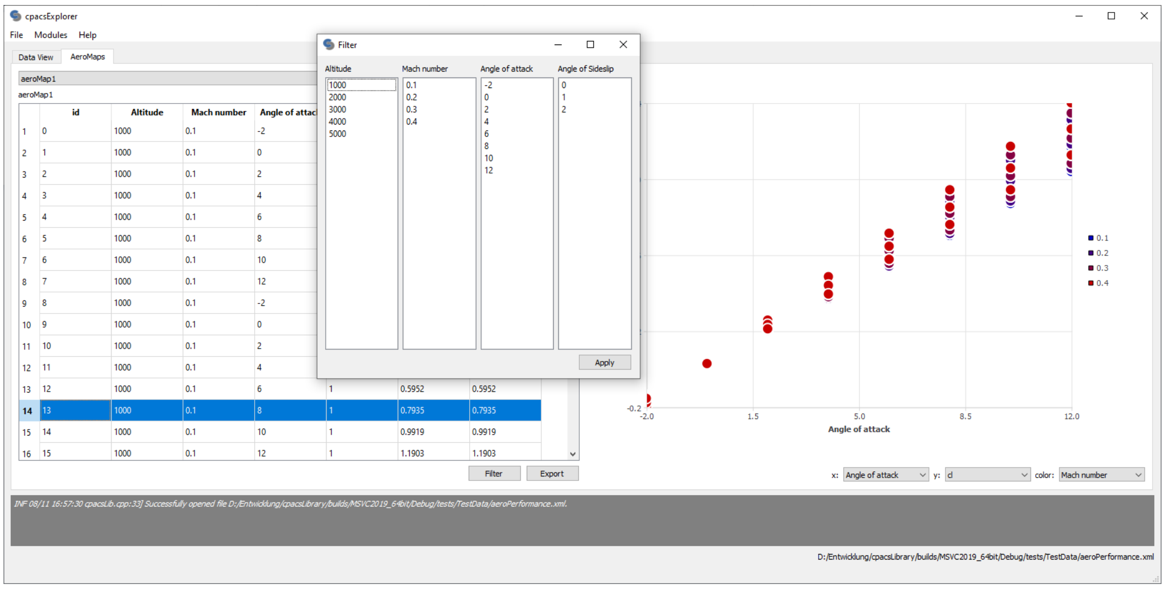
Payload range 1: block fuel

	Altitude	Angle of attack	Angle of roll	Angle of sideslip	:alibrated air spee	Flight distance	Flight time	
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	300	0.0
3	457.2	0	0	0	102.9	0	390	0.2
4	495.532	0.877817	0	0	102.9	736.262	397	0.2
5	533.752	0.877897	0	0	102.9	1473.86	404	0.2
6	571.872	0.877936	0	0	102.9	2212.8	411	0.2
7	609.889	0.877871	0	0	102.9	2953.07	418	0.2
8	647.805	0.877872	0	0	102.9	3694.68	425	0.2
9	685.618	0.877946	0	0	102.9	4437.62	432	0.2
10	723.328	0.878021	0	0	102.9	5181.9	439	0.2
11	760.935	0.878099	0	0	102.9	5927.52	446	0.2
12	798.436	0.878177	0	0	102.9	6674.47	453	0.2
13	835.832	0.878258	0	0	102.9	7422.76	460	0.2

```
INF 08/11 14:46:16 aircraftTrajectory.cpp:16] Initializing trajectory Payload range 3: reserve fuel
INF 08/11 14:46:16 aircraftTrajectory.cpp:16] Initializing trajectory Payload range 3: block fuel
INF 08/11 14:46:16 aircraftTrajectory.cpp:16] Initializing trajectory Payload range 3: reserve fuel
INF 08/11 14:46:16 aircraftTrajectory.cpp:16] Initializing trajectory Block 600nm 4240kg
INF 08/11 14:46:16 aircraftTrajectory.cpp:16] Initializing trajectory Bad conditioned trajectory (#1)
```

D:/Entwicklung/cpacsLibrary/builds/MSVC2019_64bit/Debug/tests/TestData/trajectories.xml

FIG 7. Trajectory view in cpacsExplorer.



aeroMap1

	id	Altitude	Mach number	Angle of attac			
1	0	1000	0.1	-2			
2	1	1000	0.1	0			
3	2	1000	0.1	2			
4	3	1000	0.1	4			
5	4	1000	0.1	6			
6	5	1000	0.1	8			
7	6	1000	0.1	10			
8	7	1000	0.1	12			
9	8	1000	0.1	-2			
10	9	1000	0.1	0			
11	10	1000	0.1	2			
12	11	1000	0.1	4			
13	12	1000	0.1	6	1	0.5952	0.5952
14	13	1000	0.1	8	1	0.7935	0.7935
15	14	1000	0.1	10	1	0.9919	0.9919
16	15	1000	0.1	12	1	1.1903	1.1903

Altitude	Mach number	Angle of attack	Angle of Sideslip
1000	0.1	-2	0
2000	0.2	0	1
3000	0.3	2	2
4000	0.4	4	
5000		6	
		8	
		10	
		12	

```
INF 08/11 16:57:30 cpacsLib.cpp:33] Successfully opened file D:/Entwicklung/cpacsLibrary/builds/MSVC2019_64bit/Debug/tests/TestData/aeroPerformance.xml.
```

D:/Entwicklung/cpacsLibrary/builds/MSVC2019_64bit/Debug/tests/TestData/aeroPerformance.xml

FIG 8. Aerodynamic map view in cpacsExplorer.

2.5. Python interface

The Python interface is generated via pybind11 [21]. This continued development of the boost.python library [24] provides a flexible way to translate complex C++ methods to Python, including strategies to handle overloaded constructors [25].
Pybind11 is used to export methods from both the low-level and high-level interfaces. For the current prototype, pyhbin11 code is implemented manually. Future work could comprise the automatic translation of the low-level interface to pybind11 syntax.

2.6. Software configuration and dependency management

The software configuration of cpacsLibrary is managed via CMake [26]. This allows for a flexible selection or de-selection of the various features of cpacsLibrary before compilation, such as the export of the Python interface or the inclusion of cpacsExplorer. Furthermore the third-party modules can be managed conveniently. In its current state cpacsLibrary depends on the following third-party software (license in brackets):

- TiXI [18]: CPACS XML-Interface (Apache-2.0 license)
- Boost [27]: Various components supporting C++ programming (individual Open-Source license)
- Eigen [28]: Template library for linear algebra, used for RBF interpolation (Mozilla Public License 2)
- pybind11 [21]: Creates Python interfaces from C++ code (individual Open-Source license)
- Qt Framework [29]: C++ library used for cpacsExplorer (Commercial or LGPL)
- QtCharts [30]: Extension for Qt Framework providing various visualization modules, used for chart plots in cpacsExplorer (Commercial or GPLv3)

All settings are en-capsuled in a set of CMake configuration files serving as input for a compiler-independent build of cpacsLibrary on Windows and Linux operating systems.

2.7. Exception Handling and Logging

CpacsLibrary is equipped with exception handling classes identifying errors during execution via adjustable error categories and verbose levels. The Google Logging Library (glog) [31] is used for logging. Thanks to the close alignment of the software architecture with TiGL, the exception handling and logging methods could be adopted with small modifications from TiGL, speeding up the development of a first prototype.

2.8. Testing and Change Management

All components of cpacsLibrary are extensively tested via unit tests. For this, the GoogleTest framework [32] is applied for the C++ code covering the low-, mid- and high-fidelity interfaces. Furthermore, the Python `unittest` library [33] is used to test the pybind11 exports.
The source code is maintained at a DLR GitLab repository. Common software development tools, such as issue tracking, milestone management as well as the provision of release versions with compiled code are employed.
Next steps involve the implementation of appropriate license models to make cpacsLibrary available via public GitLab [34] or GitHub [35].

3. USE-CASE: INTERPOLATION OF MULTIDIMENSIONAL MAPS

3.1. Interpolation Requirements

With the introduction of new multi-dimensional aero- and engine-performance maps in CPACS v3.3, it is no longer a requirement that the data in the d-dimensional parameter space is generated via full factorial experiment design, i.e. the data is not necessarily mapped to a uniform or regular grid. This is to avoid unrealistic parameter combinations, such as flight conditions characterized by large Mach numbers at small altitudes or vice versa.
Therefore, interpolation on scattered data in d dimensions is necessary. In this study, *radial basis function* (RBF) interpolation is chosen due to good performance characteristics for large sets of given data points $X := \{\vec{x}_1, \dots, \vec{x}_n\}$ in high-dimensional space $\mathbb{R}^d$, compared to, for example, triangulation [36]. The following section gives a brief introduction to the basic theory underlying the present study.

3.2. RBF Interpolation

An unknown function $u(\vec{x})$ fitting given data at $\vec{x}_k \in \mathbb{R}^d$, the so-called interpolant, can be constructed by a linear combination of functions $\phi(\vec{r})$ depending solely on the radial distance $r = \|\vec{x} - \vec{x}_k\|_2$ from the center $\vec{x}_k$:

$$u(\vec{x}) = \sum_{k=1}^{n} w_k \phi\left(\|\vec{x} - \vec{x}_k\|_2\right), \tag{1}$$

where $\|\cdot\|_2$ donates the standard Euclidean norm. The weights $w_k \in \mathbb{R}$ can be determined by employing the given data $u(\vec{x}_i) = f_i$ at point $\vec{x}_i, i = 1, 2, \dots, n$:

$$f_i = \sum_{k=1}^{n} w_k \phi\left(\|\vec{x}_i - \vec{x}_k\|_2\right) \tag{2}$$

Combining Eq. 2 and 1 yields a system of linear equations:

$$\vec{f} = \underline{M}\vec{w} \tag{3}$$

with the symmetric coefficient matrix (also distance matrix) $\underline{M}$:

$$\underline{M} := \begin{bmatrix} \phi(\|\vec{x}_1 - \vec{x}_1\|_2) & \cdots & \phi(\|\vec{x}_1 - \vec{x}_n\|_2) \\ \vdots & \ddots & \vdots \\ \phi(\|\vec{x}_n - \vec{x}_1\|_2) & \cdots & \phi(\|\vec{x}_n - \vec{x}_n\|_2) \end{bmatrix} \tag{4}$$

Typical approaches to solve such linear systems are LU decomposition, Cholesky Decomposition or Pseudoinverse. In literature Eq. 3 is often expanded by a matrix of polynomials to improve condition when using conditionally positive definite radial functions. As several

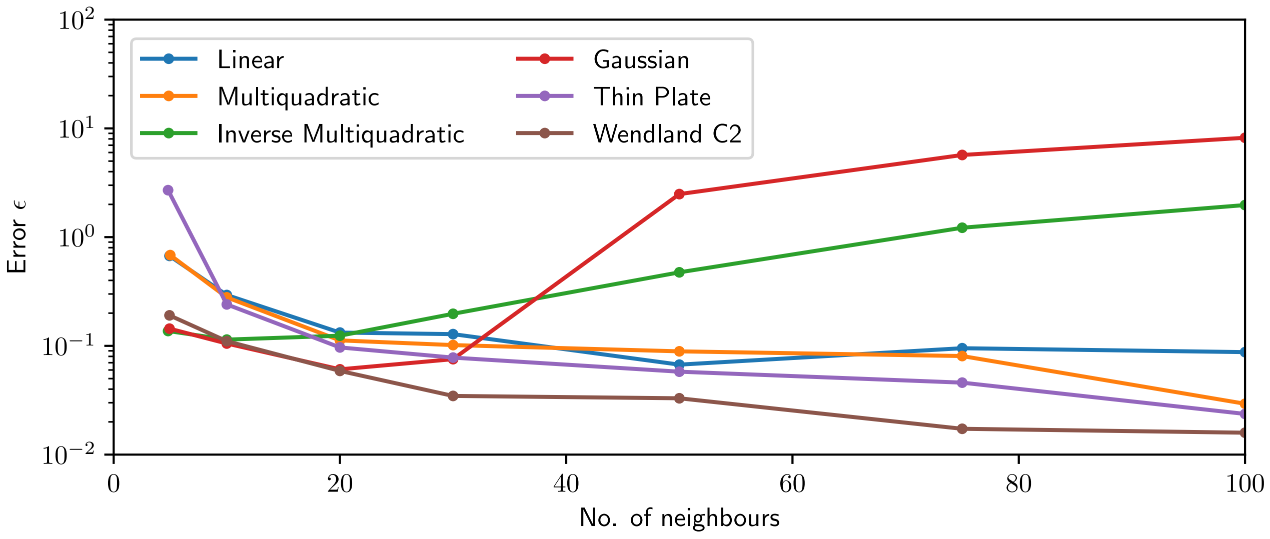


FIG 9. RBF interpolation of `aeroPerformanceMap` lift coefficient `cl` with nearest neighbor search (KNN).

standard RBF are positive definite, such as Gaussian or Inverse Multiquadratic (see below), expansion with polynomials is not considered in the present study but will be part of upcoming research to also improve results with conditionally positive definite RBFs.
RBFs can be extended by a scaling parameter ε to adjust its influence strength. The following RBFs (also referred to as *kernel*) are implemented in cpacsLibrary:

- Linear:

$$\phi(x) = r \tag{5}$$

- Multiquadratic:

$$\phi(x) = \left(\varepsilon^2 + r^2\right)^{\frac{1}{2}} \tag{6}$$

- Inverse Multiquadratic:

$$\phi(x) = \left(\varepsilon^2 + r^2\right)^{-\frac{1}{2}} \tag{7}$$

- Gaussian:

$$\phi(x) = \exp\left(-\frac{1}{2}r^2\varepsilon^2\right) \tag{8}$$

- Thin Plate:

$$\phi(x) = r^2 \log\left(\frac{r}{\varepsilon}\right) \tag{9}$$

- Wendland C2:

$$\phi(x) = \left(1 - \frac{r}{\varepsilon}\right)^4 \left(4\frac{r}{\varepsilon} + 1\right) \tag{10}$$

3.3. Implementation

Due to its generic mathematical character, the interpolation algorithms are implemented in the mid-level interface. The implementation is twofold: First, the distance matrix $\underline{M}$ is determined from the known data and then used to determine the weighting vector $\vec{w}$. This operation is numerically expensive, but has to be done only once, because the known data sets do not change. Algorithm 1 illustrates the implementation in cpacsLibrary. The second step comprises to computation of the unknown quantity f_{new} at a new point $\vec{x}_{\text{new}}$ as shown in Alg. 2.

Algorithm 1 RBF Setup

```
Input: Known data $\mathbb{D}$ at $\vec{x}_i$ with corresponding values $f_i$, $\mathbb{D} = \{(\vec{x}_i, f_i), i = 1,\dots,n | \vec{x}_i \in \mathbb{R}^d, f_i \in \mathbb{R}\}$
Input: Choice of interpolation kernel
1: for $i = 1 \dots n$ do
2:     for $k \dots n$ do
3:         Compute dist $\leftarrow \|\vec{x}_i - \vec{x}_k\|_2$ to all $\vec{x}_k$
4:         Fill distance matrix $\underline{M}_{ik} \leftarrow$ dist
5:     end for
6: end for
7: Normalize $\underline{M}$ along columns
8: Solve $\vec{f} = \underline{M}\vec{w}$ for $\vec{w}$ with LU decomposition using PartialPivLu from Eigen library [37]
9: return $\vec{w}$
```

Algorithm 2 RBF Interpolation

```
Input: New data point $\vec{x}_{\text{new}}$ for which $f_{\text{new}}$ is unknown
1: Normalize $\vec{x}_{\text{new}}$ according to $\underline{M}$
2: for all existing data points $\vec{x}_k$ do
3:     Compute distance dist $\leftarrow \|\vec{x}_{\text{new}} - \vec{x}_k\|_2$
4:     Fill distance vector $\vec{d}_k \leftarrow$ dist
5: end for
6: $f_{\text{new}} \leftarrow \vec{w} \cdot \vec{d}$
7: return $f_{\text{new}}$
```

Algorithms 1 and 2 represent the basic approach. In order to improve the computational performance of the interpolation and to avoid overfitting, it is extended by a `K-D-Tree` algorithm [38] to filter just the k nearest neighbors (KNN) used for the interpolation.

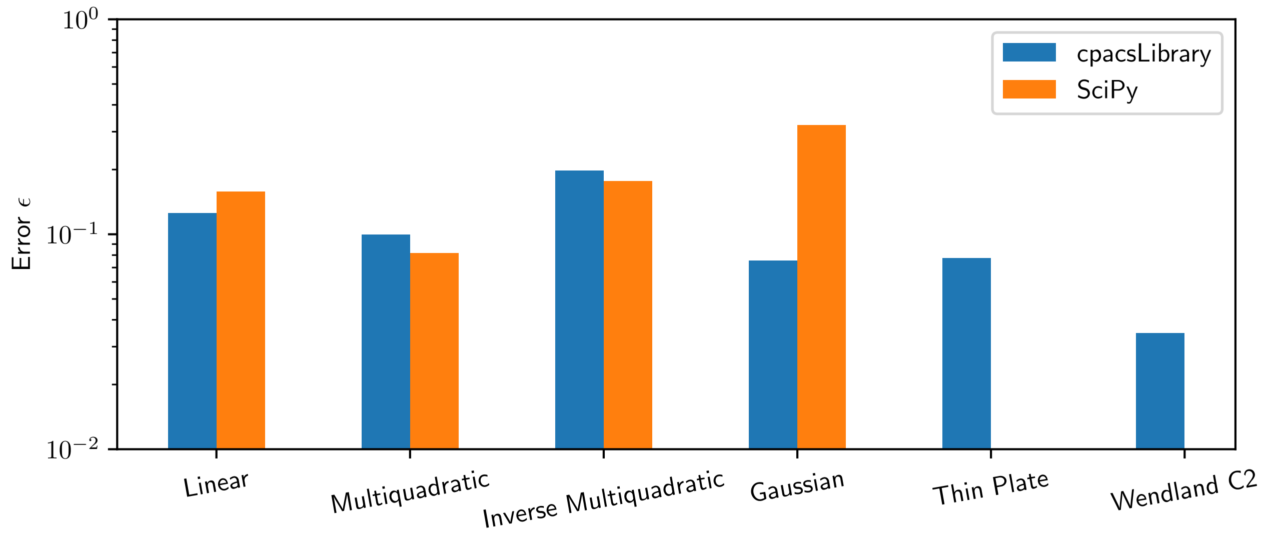


FIG 10. Comparison of interpolation errors for various RBFs and 30 neighbors.

3.4. Verification

To verify the implementation, an `aeroPerformanceMap` from CPACS with 9200 entries for the lift coefficient `cl` over the parameters `altitude`,`machNumber`, `angleOfAttack` and `angleOfSideslip` is chosen as test case. First, a uniformly distributed set of $N = 60$ samples is removed from the given data, for which the values are thus known. These are then reconstructed via the RBF interpolation, varying both the kernel and the number of nearest neighbors. The average error is determined as:

$$\epsilon = \frac{1}{N}\sum_{i}^{N}\frac{f_i - f_{\text{exact},i}}{f_{\text{exact},i}} \cdot 100\% \qquad (11)$$

Figure 9 shows the result of this study. After an initial reduction, both the Gaussian and Inverse Multiquadratic kernels increase the interpolation error again as the number of neighbors increases, which could be attributed to overfitting (i.e, the interpolant corresponds too closely or exactly to the given data, and may therefore fail to reliably predict additional data [39]). Nevertheless, it can be concluded from the results that the test points were reconstructed with satisfactory accuracy. Furthermore, a value of thirty neighbors seems well suited as a default setting for typical CPACS datasets.
Furthermore, the implementation is verified by comparison with the RBF interpolation in SciPy. SciPy is an established software library dedicated to scientific data analysis, providing verified interpolation algorithms [40]. Unfortunately, SciPy is only available in Python. Figure 10 shows the interpolation errors for thirty neighbors computed with different kernel functions. As the errors are in the same order of magnitude, it can be concluded that the RBF algorithm is properly implemented.

4. SUMMARY AND OUTLOOK

This paper presents a software called cpacsLibrary for performant and consistent processing of parametric aircraft data in the CPACS data model. An overview of current research projects in which CPACS is used as the central data exchange format illustrates that the multidisciplinary design of novel aircraft system architectures not only benefits from using a central source of data, but also places new requirements on CPACS due to the increasing system complexity and level of detail. Among others, driving topics are the design of sustainable propulsion architectures as well as configurational changes of military aircraft and helicopters during mission simulation.
Although XML Schema Definition (XSD) allows to create advanced data models, complex rules and knowledge to further interpret the data can currently only be captured via the CPACS documentation or, in the case of geometric operations, with the TiGL geometry library. CpacsLibrary therefore complements the CPACS data model with an object-oriented low-level interface, as well as disciplinary high-level interfaces providing consistent data processing methods. A first prototype demonstrates a flexible software architecture with disciplinary knowledge en-capsuled in dedicated modules and serves as proof-of-concept. Implemented in C++ the software not only allows for performant data processing through compiled code, but also provides the possibility to provide interfaces for other programming languages, such as Python. The close alignment of cpacsLibrary with the development of TiGL allows to benefit from best-practice and lessons-learned, to use existing third party software like CPACSGen, as well as to foster future collaboration between cpacsLibrary and TiGL development teams.
The concept of cpacsLibrary has been demonstrated through processing and visualization of flight trajectories as well as aerodynamic data sets. The implementation of a scattered data interpolation based on Radial Basis

Functions (RBF) underlines the need for robust interpolation methods in order to guarantee a consistent interpretation of the data in multidisciplinary projects. The prototypical implementation was tested using four-dimensional aerodynamic maps and the results were compared with the Python-based open-source software SciPy. The accuracy of interpolation is comparable to SciPy and therefore verifies the correct implementation of the RBF interpolation algorithms.

Currently cpacsLibrary is still in an early alpha development phase. Next steps include extensions of the aerodynamic module to take increment maps for control surface deflections into account. Based on the same approach, an engine module will be developed employing the same interpolation routines. First tests indicated good performance when applied to irregularly distributed multi-dimensional engine performance maps. However, further studies are needed to identify the most robust RBF settings for the evaluation of typical aerodynamic and propulsion performance maps which will then be implemented as default values.

Next developments will aim at handling the complexity of various system components and their complex interactions. Further research on the software architecture comprises finding an alternative solution for cpacsExplorer based on modern web technologies to avoid GPLv3 licensing from Qt-Framework modules and to provide a state-of-the-art user experience through a web interface with more interactive visualizations.

The above developments are currently maintained on a DLR-internal version management infrastructure (GitLab). A publication strategy is under development. Nevertheless, any support is expressly welcome and by contacting the authors, individual solutions for early collaboration with external partners will be found. All developments aim to establish cpacsLibrary as a fundamental component in the CPACS ecosystem and as such, largely enhance both the easiness and consistency in using CPACS as a central data exchange model in multi-disciplinary research.

References

[1] Björn Nagel, Daniel Böhnke, Volker Gollnick, Peter Schmollgruber, Arthur Rizzi, Gianfranco La Rocca, and Juan J. Alonso. Communication in Aircraft Design: Can we establish a Common Language? In *28th International Congress of the Aeronautical Sciences*, 2012.

[2] Marko Alder, Erwin Moerland, Jonas Jepsen, and Björn Nagel. Recent Advances in Establishing a Common Language for Aircraft Design with CPACS. In *Aerospace Europe Conference 2020*, 2020.

[3] Jason Edelman, Scott Lowe, and Matt Oswalt. *Network Programmability and Automation*. O'Reilly Media, Sebastopol, CA, March 2018.

[4] W3C. XML Schema Part 0: Primer Second Edition. www.w3.org/TR/xmlschema-0, 2004. [Online; accessed 20-August-2022].

[5] Margit Becher. *XML: DTD, XML-Schema, XPath, XQuery, XSL-FO, SAX, DOM*. Springer Vieweg, 2 edition, February 2022.

[6] Eric van der Vlist. *XML Schema*. O'Reilly, Heidelberg, Germany, 1 edition, February 2003.

[7] W3C. XML Schema Part 1: Structures Second Edition. www.w3.org/TR/xmlschema-1, 2004. [Online; accessed 20-August-2022].

[8] Hartmann, Johannes and Nagel, Björn. Eliminating Climate Impact From Aviation - A system level approach as applied in the framework of the DLR-internal project EXACT. Presentation at the DLRK 2021 Web Conference, 2021.

[9] Daniel Silberhorn, Katrin Dahlmann, Alexander Görtz, Florian Linke, Jan Zanger, Bastian Rauch, Torsten Methling, Corina Janzer, and Johannes Hartmann. Climate impact reduction potentials of synthetic kerosene and green hydrogen powered mid-range aircraft concepts. *Applied Sciences*, 12(12):5950, June 2022.

[10] IMOTHEP. Investigation and Maturation of Technologies for Hybrid Electric Propulsion. www.imothep-project.eu, 2022. [Online; accessed 20-August-2022].

[11] Agile 4.0. Towards cyber-physical collaborative aircraft development. www.agile4.eu, 2022. [Online; accessed 20-August-2022].

[12] Luca Boggero, Pier Davide Ciampa, and Björn Nagel. An MBSE architectural framework for the agile definition of complex system architectures. In *AIAA AVIATION 2022 Forum*. American Institute of Aeronautics and Astronautics, June 2022.

[13] Jasper H Bussemaker, Pier Davide Ciampa, Jasveer Singh, Marco Fioriti, Carlos Cabaleiro De La Hoz, Zhijun Wang, Daniël Peeters, Philipp Hansmann, Pierluigi Della Vecchia, and Massimo Mandorino. Collaborative design of a business jet family using the AGILE 4.0 MBSE environment. In *AIAA AVIATION 2022 Forum*, Reston, Virginia, June 2022. American Institute of Aeronautics and Astronautics.

[14] Diabolo. Technologies and design of next generation fighter aircraft. www.dlr.de/as/en/desktopdefault.aspx/tabid-15880/25737_read-66160. [Online; accessed 20-August-2022].

[15] CHASER. Conceptual Handling Assessment Simulation and Engineering of Rotorcraft. www.dlr.de/as/en/desktopdefault.aspx/tabid-18135/28809_read-74794. [Online; accessed 20-August-2022].

[16] Martin Siggel, Jan Kleinert, Tobias Stollenwerk, and Reinhold Maierl. TiGL: An Open Source Computational Geometry Library for Parametric Aircraft Design. *Mathematics in Computer Science*, 7(1):23, 2019.

[17] Tobias Schlauch. Framework Directive Software Engineering. Technical Report QMH-DLR-VA004, DLR, 2022.

[18] DLR Institute for Simulation and Software Technology. TiXI: fast and simple xml interface library. http://tixi.sourceforge.net/Doc/index.html, 2019. [Online; accessed 18-August-2022].

[19] RISC Software GmbH. CPACSGen: generates CPACS schema based classes for TiGL. www.github.com/RISCSoftware/cpacs_tigl_gen, 2018. [Online; accessed 18-August-2022].

[20] DLR Institute for Simulation and Software Technology. TiGL. www.github.com/DLR-SC/tigl, 2018. [Online; accessed 18-August-2022].

[21] Pybind 11. www.github.com/pybind/pybind11, 2022. [Online; accessed 18-August-2022].

[22] Swig. www.swig.org, 2019. [Online; accessed 18-August-2022].

[23] DLR Institute for Simulation and Software Technology. RCE - Remote Component Environment. www.rcenvironment.de, 2022. [Online; accessed 19-August-2022].

[24] Boost.Python. Building Hybrid Systems with Boost.Python. www.boost.org/doc/libs/1_63_0/libs/python/doc/html/article.html. [Online; accessed 19-August-2022].

[25] Pybind 11. Overloaded methods. http://pybind11.readthedocs.io/en/stable/classes.html#overloaded-methods, 2022. [Online; accessed 19-August-2022].

[26] CMake. www.cmake.org. [Online; accessed 19-August-2022].

[27] Boost. www.boost.org. [Online; accessed 19-August-2022].

[28] Eigen v3. www.eigen.tuxfamily.org, 2010. [Online; accessed 19-August-2022].

[29] Qt. www.qt.io, 2022. [Online; accessed 19-August-2022].

[30] Qt Documentation. Qt Charts. https://doc.qt.io/qt-6/qtcharts-index.html, 2022. [Online; accessed 19-August-2022].

[31] Google. Google Logging Library. www.github.com/google/glog, 2022. [Online; accessed 19-August-2022].

[32] Google. GoogleTest - Google Testing and Mocking Framework. www.github.com/google/googletest, 2022. [Online; accessed 19-August-2022].

[33] Python Software Foundation. unittest - Unit testing framework. docs.python.org/3/library/unittest.html, 2022. [Online; accessed 19-August-2022].

[34] GitLab. www.gitlab.com. [Online; accessed 19-August-2022].

[35] GitHub. www.github.com. [Online; accessed 19-August-2022].

[36] William H. Press, Brian P Flannery, Saul A Teukolsky, and William T Vetterling. Numerical recipes in C: The art of scientific computing, February 2002.

[37] Eigen. Online Documentation. www.eigen.tuxfamily.org/dox/classEigen_1_1PartialPivLU.html, 2018. [Online; accessed 18-August-2022].

[38] Rosettacode.org. K-d tree. www.rosettacode.org/wiki/K-d_tree, 2022. [Online; accessed 18-August-2022].

[39] English: Oxford Living Dictionaries. Definition of "overfitting". https://web.archive.org/web/20171107014257/https://en.oxforddictionaries.com/definition/overfitting. [Online; accessed 22-September-2022].

[40] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. DOI: 10.1038/s41592-019-0686-2.

Contact address:

marko.alder@dlr.de

A. CPACS SCHEMA

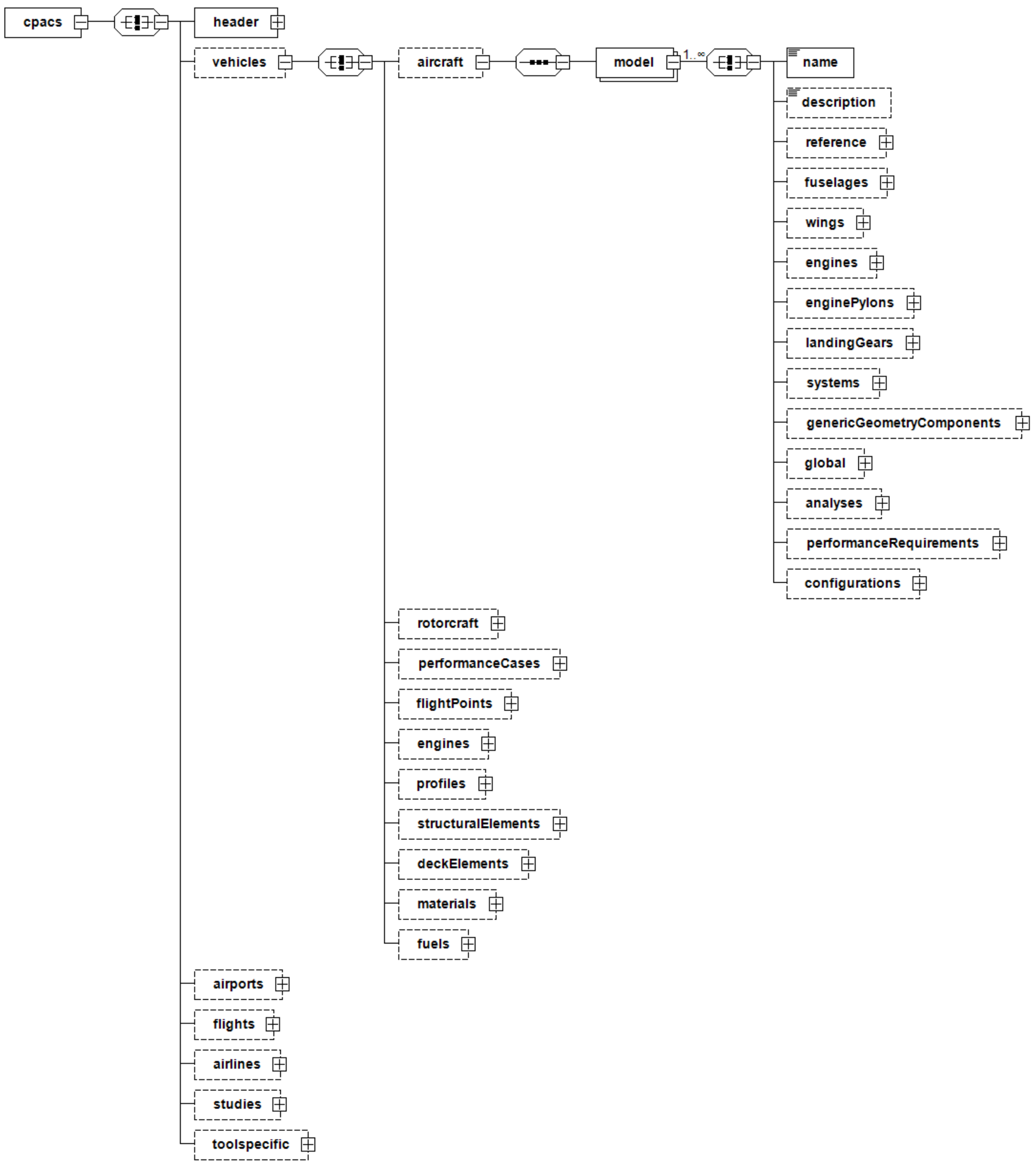


FIG 11. Tree structure diagram of the CPACS v3.4 XML Schema Definition (XSD).

B. CODE EXAMPLES

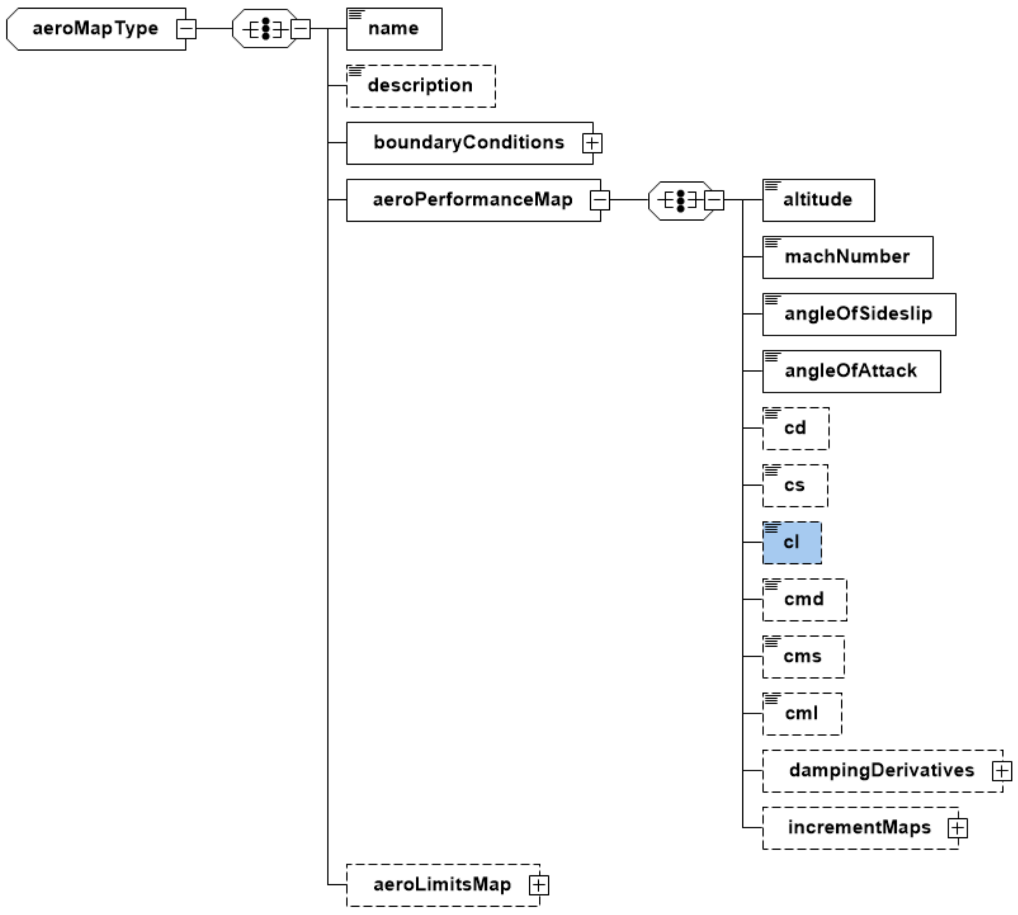


FIG 12. CPACS XSD type for aeroMaps.

```
const char* filename = "TestData/aeroPerformance.xml";
m_cpacs.openCPACS(filename);
m_cpacsTree = m_cpacs.cpacsTree();
const auto& aeroMap = m_cpacsTree->GetVehicles()->GetAircraft()\
   ->GetModels().at(0)->GetAnalyses()->GetAeroPerformance()\
   ->GetAeroMaps().at(0);
const std::string cl = aeroMap->GetAeroPerformanceMap().GetCl()\
   ->GetSimpleContent();
```

Listing 1. Low-Level Interface Example

```
const char* filename = "TestData/aeroPerformance.xml";
m_cpacs.openCPACS(filename);
m_aero = m_cpacs.cpacsAero();
aeroMap1 = m_aero->aircraftAeroMaps().at(0);
std::vector<double> cl = aeroMap1->cl()->values();
```

Listing 2. High-Level Interface Example