Center of Excellence in Combustion

D6.2 Definition of simulation API, post-processing functions and database schema

Version 1.0

Document Information

Contract Number	952181
Project Website	http://www.coec-project.eu
Contractual Deadline	30/09/2021 (M12)
Dissemination Level	Public
Nature	Report
Author	Jens Henrik Göbbert
Contributors	Antoine Dauptain, Christos Emmanouil Frouzakis, Jimmy-John Hoste, Christian Witzler, Daniel Mira Martinez
Reviewers	Eleonore Riber, Salvatore Iavarone

Center of Excellence in Combustion

Change log

Version	Author	Description of Change
V0.5	Jens Henrik Göbbert	Initial draft for review
V0.6	Antoine Dauptain	Document reviewed
V0.7	Christos Emmanouil Frouzakis	Document reviewed
V0.8	Christian Witzler	Document reviewed
V0.9	Jens Henrik Göbbert	Update of minor issues
V1.0	Eleonore Riber	Document reviewed

The CoEC project has received funding from the European Union's Horizon 2020 research and innovation programmed under grant agreement No 952181.

Center of Excellence in Combustion

Outline

The CoEC project has received funding from the European Union's Horizon 2020 research and innovation programmed under grant agreement No 952181.

1. Introduction

The Center of Excellence in Combustion (CoEC) grew out of the European Union's need to decarbonize power generation and transportation to achieve net-zero greenhouse gas (GHG) emissions by 2050 [1].

In fact, global energy demand is increasing every year, and most of this energy is generated by burning fossil fuels. To overcome this drawback, an increased use of sustainable fuels such as biofuels, hydrogen or power-to-X (P2X) based on "e-fuels" is planned in the near future. Therefore, there is a clear need for the development of advanced simulation software to support this transition.

To this end, the CoEC project is investing in eleven outstanding high-performance codes, each of which has long proven to be an important building block in research for the combustion community. The simulation codes allow questions in combustion to be answered, providing important insights in research and development. Each individual code comes with special capabilities and is invaluable in its own right. Developed over many years, the simulation codes are already being used extensively in fundamental and applied large scale simulations performed on HPC systems. However, even more is possible if the simulation tools and the obtained results can be exploited by the different groups. We are convinced that from a technical point of view, common standards such as simulation APIs, post-processing functions and database schemas are needed.

This deliverable summarizes the results of discussions between the project partners during the first 12 months of WP6 and is intended to provide further directions.

1.1 General Challenge

Within the CoEC, eleven HPC simulation codes are further developed and used to address various scientific questions. The knowledge and time of the numerous developers that have gone into the development of the codes over the years make the simulation programs, with their more than **1.5 million**[1] lines of source code, extremely feature-rich and at the same time irreplaceable. The speed of development is high and unstoppable.

Each individual code is independent in itself and is driven continuously by the specific research priorities of the individual scientists. The codes use different models, mathematical solution approaches, and numerical methods. For example, they differ in discretization methods, grid representation, parallelization scheme, and turbulence and combustion models - just to name a few of the most obvious differences. The codes are written in different programming languages and support or are optimized for different hardware (primarily with GPU support or without). From an HPC perspective, there are different challenges in performance, scalability, and I/O. So far, only the technical differences have been listed here. There are of course further challenges - independent of the actual code - in the communication and decision-making structure among the developers, the software management or the details of the individual licenses. In addition,

[1] CoEC proposal, table 1.3.a

The CoEC project has received funding from the European Union's Horizon 2020 research and innovation programmed under grant agreement No 952181.

the codes run on a wide variety of supercomputers at different computing centers with different software stacks and side-specific services.

All in all, the previous paragraph should make it clear that under these circumstances it is very difficult to define and develop **common**, **uniform** and **forward-looking standards** that also find their way into the practice of the individual developers and users of the simulation codes.

1.2 Strategy

For the reasons mentioned above, our focus is primarily on finding, defining, developing and establishing common standards that take these challenges into account and are of practical use. For this to happen, they must be supported, desired, or better yet, craved by both developers and users. We believe it is important, therefore, that they can be incorporated into individual simulation programs and established workflows with minor changes. This goal has the highest priority.

Nevertheless, it is important to us that even modern forward-looking solution methods do not have to be discarded because they require a major rethink on the part of the code developers or users of the simulation programs. The constantly growing complexity of research questions, collaborations, workflows and, of course, hardware and software can only be overcome by consistently exploiting new technology. Therefore, we keep the possibility open to test and introduce methods that require a strong rethinking on the part of code developers and users in practice.

Programming interfaces are always caught between continuity and innovation. On the one hand, there is the desire that the interface does not change over time (continuity) and on the other hand, one would like to continuously add new functionality (innovation). Especially in the early development phase, it is important not to fix the programming interface too early, but to keep the process to the final definition open.

For the planned standards, we therefore distinguish between a **practical approach** that integrates without major changes to existing software and a **disruptive approach** that is implemented more elegantly but expects major changes to existing software in return.

1.3 Connections with other deliverables and work packages

The CoEC aims to achieve a high level of excellence in science and technology. This is largely supported by the track record of the scientists involved in the project and also by the ambitious goals of the proposed eight Exascale Application Challenges (EACs). The CoEC identifies the defined EACs as critical to the implementation of low-carbon technologies.

Therefore, the common EACs form the central link of this deliverable 6.2 to the work packages **WP4, WP5** and **WP7**. D6.2 lays the **technical foundation** necessary for **efficient collaboration** between the project partners with their different simulation codes and data sets.

The CoEC project has received funding from the European Union's Horizon 2020 research and innovation programmed under grant agreement No 952181.

2. Simulation APIs

The term "API" (Application Programming Interface) generally refers to a programming interface. It is the part that is made available by a simulation program to other programs for connection. In contrast to an Application Binary Interface (ABI), an API defines only the program connection at source code level.

The actual goal of an API is to enable different codes to communicate with each other. In which way this is achived is for the time being open. Roughly one can divide therefore programming interfaces into function-oriented, file-oriented, object-oriented and protocol-oriented classes. Depending upon the respective use case their advantages and disadvantages result.

2.1 Aims

Based on the very general definition of an API, the present simulation codes of course already build on numerous APIs in various ways and define quite a few more in addition. We would like to focus here specifically on the interfaces for runtime information and simulation control.

Runtime information becomes of ever increasing importance as the size, complexity and heterogeneity of HPC systems and simulations grow. With a view to future exascale supercomputers, our goal is to ensure that the best possible insight into the current state is available at every point in time of a simulation. This is the only way to ensure that the expensive resources are used wisely and that running simulations are heading for the desired results using the planned compute time.

Simulation control provides the ability to manually intervene in an otherwise autonomous simulation process to change its outcome. The term is often used to specifically describe interactive control of a computer experiment aimed at moving it into a particular region of interest - e.g., by changing geometry at runtime or making changes to flow parameters. However, this is *not* the type of simulation control we are talking about here. The **goal** of an interface for simulation control should be to provide additional control commands at runtime, which do not change the simulation progress itself, but e.g. ask the program for additional information, trigger the execution of dedicated analysis functions, or start I/O to the storage system or to external services. In this case, the actual simulation progress is not changed.

2.2 Strategy

In the introduction of this document, the general challenges in defining new standards have been discussed. Of course, we also have to face these when defining and developing simulation APIs. They must have the high flexibility to continuously adapt to future changes in the different simulation codes and the different sites.

Similar challenges have to be mastered in cloud computing today. Therefore, our **strategy** is to adopt the basic principle of **micro-services** to some extent here. We implement the

The CoEC project has received funding from the European Union's Horizon 2020 research and innovation programmed under grant agreement No 952181.

interface for runtime information and simulation control as independent programs that can be started in parallel to the simulation programs (presumably at rank 0).

In this way, they can be developed independently of the HPC codes and better exploit the capabilities of the supercomputing center or compute node without introducing new dependencies from the perspective of the simulation programs.

Their task is to collect runtime information from a wide variety of sources and make it available to the outside world via a "**Representational State Transfer" (REST) API.** In the other direction, they accept instructions for simulation control via a REST API and forward them to the simulation tool. The information exchange is to be done using the **JSON**[2] format. In combination with **SCHEMA**[3] the data can be validated and understood.

2.3 Expected outcomes

The provision of the simulation APIs includes the detailed documentation of the interface functions with their parameters as an electronic document. This document is kept as a **live document**, which allows to continuously extend and improve the standard at runtime of CoEC and beyond.

It describes the functionality of the simulation APIs for runtime information and simulation control. The **implementations** are initially done with a practical approach as two independent programs, which are started together with the simulation tool in the same job script and communicate externally via a REST API using JSON.

Runtime information is collected and provided from various sources. These are both static settings of the job (queried via the scheduler) and the computing resource (e.g. via /proc) as well as dynamic measured variables such as performance, scalability, load balancing. These are either queried from the system (e.g. performance counter) or the simulation tool, or are measured variables of the simulation itself, which describe the combustion, turbulence or other dynamic parameters.

The **simulation control** is strongly dependent on the capabilities of the respective simulation tool. The simulation controls task is to communicate these possibilities to the outside via a uniform REST interface and to pass on incoming instructions to the simulation tool. Which instructions are possible is defined solely by the simulation tool itself. Therefore, the focus here is on hiding this inconsistency of the simulation tools as much as possible.

The simulation APIs are essential for the development of **CoEC dashboards**.

3. Post-processing functions

The term post-processing refers to the **after-treatment and preparation** of simulation results. Post-processing is often carried out with a separate program that has special

[2] http://json-schema.org/understanding-json-schema
[3] https://cerfacs.fr/coop/json-schema-for-sci-apps

The CoEC project has received funding from the European Union's Horizon 2020 research and innovation programmed under grant agreement No 952181.

functions and procedures for data analysis. The large amounts of data are prepared in such a way that insights can be derived from the calculated results. For this purpose, it may often be necessary to combine post-processing functions.

As the name **"post-processing"** suggests, post-processing functions are executed after "processing". What exactly is called "processing" remains open for the time being. In CoEC the processing is the simulation and the result of the processing is the simulation data. In the classical case, this simulation data is available at the end of the simulation. However, since the calculation is usually evolving and develops along a simulation time, intermediate results (e.g. at the end of an a time step) can also be important for gaining knowledge. Therefore, in the following we also refer to the evaluation of these intermediate results as "post-processing", although the simulation itself has not yet reached its end.

Where the post-processing will be done is still open. Usually, it is performed on separate compute nodes on the result data stored on the storage system at the end of the simulation. At simulation runtime, however, it can also be performed on the compute nodes used by the simulation tool itself - in this case we can speak of "in situ analysis". If the simulation data is transferred to other computational nodes at runtime of the simulation to be analyzed there, this is usually called "in transit analysis". In any case, we include these special forms of data analysis under the term "post-processing".

3.1 Aims

There are a large number of post-processing functions that remain unchanged by the developers for a relatively long time (e.g. to determine the quality of simulation data based on certain parameters) and are available in different realizations for different simulation tools.

However, we do not want to re-implement existing functionality in CoEC, but primarily promote scientific creativity. With special post-processing functions, new insights can be gained from the simulation data. Along the way, the functions have to be changed often and quickly because they look a little different for each problem. It is therefore our **goal** that **new post-processing functions can be developed faster, with more functionality and in higher quality**.

Post-processing is basically independent of the simulation tool used and is per se based on the simulation data alone. However, simulation tool and simulation data are not easy to separate from each other. The most obvious is the data format used, but there are plenty of other reasons why post-processing functions are often tightly linked to specific simulation tools. The scientific question itself is clearly open to be investigated with another simulation tool as well. It is therefore our **goal** to create possibilities for **post-processing functions to be developed independently of the simulation tool**.

3.2 Strategy

The structure of a post-processing function usually consists of three steps: reading the data, evaluating the data, and saving or passing on the results. The actual scientific

The CoEC project has received funding from the European Union's Horizon 2020 research and innovation programmed under grant agreement No 952181.

question is examined in the 2nd step, the evaluation. Also the storing or passing on of the analysis result can be generally regarded as independent of the simulation tool. However, the first step, the reading of the data, is not. In order to achieve the set goals, we see above all great potential for simplification in the provision of **functions for reading the simulation data**.

However, since each simulation tool has its own historically developed or methodologically/numerically justified **data and grid structures**, simulation data cannot be used directly by another CoEC simulation program. In practical terms, the different file formats are also a major technical obstacle.

Our **strategy** is therefore to focus on the practical benefits of the solution planned and to transform the simulation data (if necessary) into data and grid structures that can be read by the "Visualization Toolkit" (VTK)[4]. We try to avoid possible numerical errors during the transformation, but accept them if they are unavoidable and small enough.

Post-processing functions can thus build on data and grid structures that are clearly defined by VTK. We hope to be able to **achieve both of our goals** in this way. Post-processing functions can be developed faster, with more functionality and in higher quality and are at the same time significantly more independent of the respective simulation tool.

We are aware that there are still plenty of good reasons for many analysis functions to be developed specifically for a particular simulation tool. It therefore remains a matter for the developer to weigh up which path to use for a new post-processing function.

3.3 Expected outcomes

Post-processing functions can access the simulation data via VTK data structures in memory. This way they can do their calculations on the data without a deep knowledge of the specific HPC application used for the simulation.

Preferably, the data is received at runtime so that the detour via the storage system can be avoided. The possibilities of **ParaView/CatalystV2** are promising and are to be used here particularly, since in this way data structures can be transferred by the simulation program in many cases 1:1 without having to be converted. Nevertheless, this preferred method requires changes to the code of the simulation programs.

However, we still expect that loading the simulation data from the **storage system** will remain the preferred method. Therefore, I/O wrappers shall be developed for this purpose as well, which can operate independently of a running simulation code. In this case, too, post-processing functions make the simulation data available as VTK data structures in the working memory.

We are aware that I/O wrappers are usually difficult to maintain because the code they enclose is constantly changing and often does not conform to a strict non-regression strategy. This therefore remains a challenge that needs to be addressed.

[4] https://vtk.org

The CoEC project has received funding from the European Union's Horizon 2020 research and innovation programmed under grant agreement No 952181.

With the availability of I/O wrappers, it will be possible for scientists to check their knowledge gained with one simulation tool much more easily on the simulation data from other simulation tools. We therefore expect that this will make **collaborations** between different research groups easier and lead to solutions more quickly. More data will be available to each individual and post-processing functions can be shared.

It is important that the I/O wrappers are not only technically well implemented, but also so easy to use that they can be used in **everyday research** without much prior knowledge.

4. Database Schema

This section is largely based on the content of the live document "Towards a common understanding of Combustion solvers data formats"[5], written and maintained for CoEC by Antoine Dauptain and Jimmy-John Hoste.

Scientific applications often use large and very complex inputs without any standard. The previous section primarily discussed how CoEC will make it possible for different post-processing functions to read in simulation data from different simulation codes. However, it did not discuss how they will be informed about the meaning of the data. This aspect will be done through an understandable database schema.

SCHEMA is a standard data structure notation system originally used for JSON formatted data. However, it can easily be used for other serialization formats as well. It is now a central part of data exchange on the web and defines what data is needed for an application and how it can be modified.

SCHEMA is a standard for "documenting the footprint" of a data tree in a format-independent approach (tree structure, values/branches, documentation). It can help validate inputs, add precise documentation, automatically fill in the missing part and create interfaces.

A typical documentation of the temperature field in SCHEMA would look like this, for example:

```
temperature:
  type: object
  description: Static temperature (Kelvins)
  properties:
    dtype:
      type: string
    value:
      type: array
      items:
      - type: float
```

The SCHEMA information is a detailed description of the data set. The following figure shows how similar data is stored for different solvers, namely AVBP, YALES2 and Nek5000.

[5] https://cerfacs.fr/coop/combustion-schema-wip

The CoEC project has received funding from the European Union's Horizon 2020 research and innovation programmed under grant agreement No 952181.

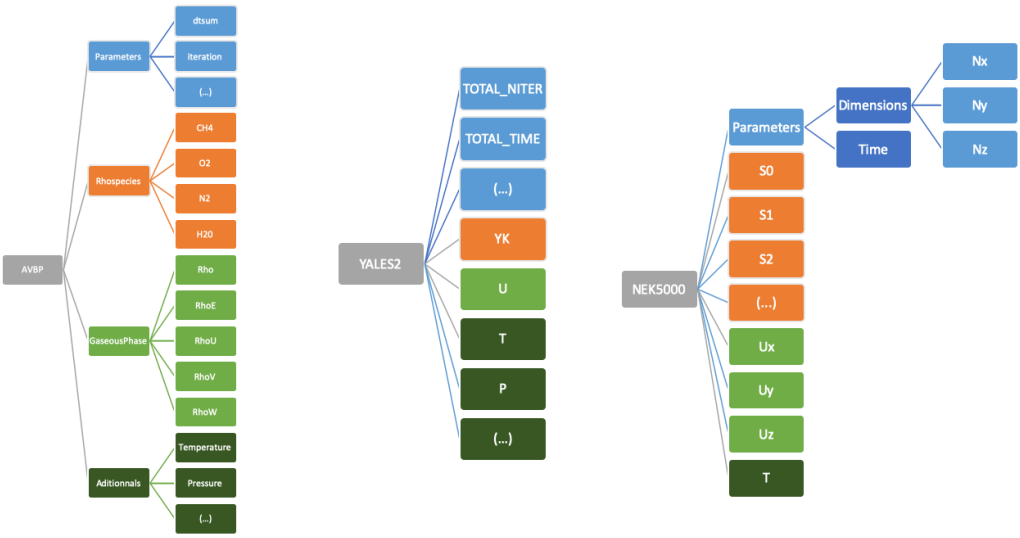

Fig. 1: Similar data is stored for different solvers. Colors for parameters, species, velocity components, variables.

4.1 Aims

While scientists usually understand each other theoretically thanks to the publications, deciphering how other teams technically store their data is always a problem. By contrasting their structures, a common basis for understanding is thus created within CoEC.

Our **goal**, instead of the impossible approach of trying to find a standard for data formats/data structures, is to agree on a **consistent way to describe our various data sets**.

In this way, the data of the different simulation programs will not only be both readable and understandable by for example the **post-processing functions**.

4.2 Strategy

To achieve this goal, the first step is to analyze and understand the data structure of the output data of each CoEC simulation program and document it using SCHEMA in a format-independent way. This then describes *where*, *what* and *how* the data is stored.

The CoEC project has received funding from the European Union's Horizon 2020 research and innovation programmed under grant agreement No 952181.

Schema blueprints are available so far for AVBP[6], Nek5000[7], YALES2[8], OpenFOAM[9] and ALYA[10]. They have been developed using sample datasets from the various simulation tools collected in a Git repository[11]. Commonalities between the data trees were also searched for, making the resulting SCHEMA even more widely applicable.

Often there are parameters (Fig. 1, blue) that provide global information about the data set, such as the physical time and iteration number associated with the data. In the case of gas flow, these are the velocity components (Fig. 1, green). AVBP stores the conservative variables and puts temperature and pressure in a separate additional group, while YALES2 and Nek5000 directly include pressure and temperature in addition to velocity. The species data set (Fig. 1, orange) gives the composition of the reactive gas. It can be stored as a vector (Yk for YALES2) and sometimes requires a correspondence table (Nek5000). The species concentrations seem to be mass fractions in all cases (it could also be volume fraction). Although, the values do not always translate so easily: AVBP for example stores RhoYk in accordance with its conservative storage of flux components.

In the overall **strategy**, SCHEMA is a **crucial building block** on which further CoEC tools are to be based.

4.3 Expected outcomes

We consider it difficult to impossible to introduce and enforce a uniform new data-structure in the historically grown and highly dynamic scientific combustion community. With the strategy of using SCHEMA, however, this is not necessary. In this way, the scientific community can retain its flexibility and still have a method at hand that simplifies collaboration among the different groups and scientists and makes data validation and data conversion possible with significantly less effort.

The first steps on this path have been taken and blueprint schemas for AVBP, Nek5000, YALES2, OpenFOAM and ALYA have been developed using sample datasets. The second step is now to make use of them.

With a **clear understanding** of the data, software tools can take on new tasks that until now could only be performed by humans. As a result, we expect to see for example tools for automatically extracting meta-information from simulation datasets, tools for validating datasets, and for semi- to fully-automatically converting data for one simulation tool to another.

[6] http://www.cerfacs.fr/avbp7x
[7] https://nek5000.mcs.anl.gov
[8] https://www.coria-cfd.fr/index.php/YALES2
[9] https://www.openfoam.com
[10] https://www.bsc.es/research-development/research-areas/engineering-simulations/alya-high-performance-computational
[11] https://gitlab.com/cfrouzak/coec_wp6

The CoEC project has received funding from the European Union's Horizon 2020 research and innovation programmed under grant agreement No 952181.

5. Summary & Conclusion

On a technical level, we will simplify the access to the results of the eleven high-performance CoEC simulation codes in order to accelerate the knowledge gain and to counteract the increasing complexity of the overall systems. We are convinced that from a technical point of view, common standards such as simulation APIs, post-processing functions and database schemas are needed.

Defining common, uniform and forward-looking standards is usually not a problem. However, making them practical is a challenge. Our highest priority is therefore the practical benefit, even if we will also test disruptive approaches in parallel. The practical benefit will be measured on the basis of the CoEC Exascale Application Challenges (EAC) by the efficiency of the collaboration between the project partners with their different simulation codes and data sets.

As we move towards exascale supercomputers, we aim to improve access to runtime information and simulation control. It is our goal to provide the best possible insight into the current state at any point in time of a simulation. In addition, despite different codes, it is intended to develop a unified interface for simulation control that can be used to send additional control commands at simulation runtime. Both will also pave the way to a CoEC dashboard.

Post-processing is basically independent of the simulation tool used and is per se based on the simulation data alone. However, simulation tool and simulation data cannot be easily separated from each other. Therefore, post-processing functions are often developed specifically to the simulation code. The developments in CoEC will allow a certain amount of post-processing functions to be developed independently of the simulation tool and thus to arrive at a new evaluation of data faster, with more functionality and with higher code quality.

However, this requires not only the technical possibility to read simulation data of the different CoEC simulation tools, but the data must also be understood by the evaluation program. Due to the differences in the data trees, this cannot be taken for granted. With SCHEMA, the different data sets are described in a uniform way so that tools can gain an understanding of their meaning.

All together, these building blocks add parts to develop the necessary technical solutions that, despite the increasing complexity of systems and scientific questions, help shape the transition to sustainable fuels by use of advanced simulation software.

Bibliography

[1] EU Commission, COM, 2018, p. 773.

The CoEC project has received funding from the European Union's Horizon 2020 research and innovation programmed under grant agreement No 952181.