An architecture for CFD Workflow Management

Claudio Gargiulo
R&D - Aerothermal CFD
Fiat Group Automobiles, Italy
Email: claudio.gargiulo@fiat.com

Donato Pirozzi
ISISLab, Dip. di Informatica
Università di Salerno, Italy
Email: dpirozzi@unisa.it

Vittorio Scarano
ISISLab, Dip. di Informatica
Università di Salerno, Italy
Email: vitsca@dia.unisa.it

***Abstract*—Nowadays, to design product impacted by fluid flow, industries use Computational Fluid Dynamics (CFD) to get a better insight into product behaviour. In this paper, we present a system architecture design for CFD Workflow management.**

***Index Terms*—Architecture, CFD, CFD Workflow, management**

I. Introduction

Computational Fluid Dynamics (CFD) is computer numerical simulation able to solve and analyse problems that involve fluid flow, heat transfers and related physical phenomena. Actually, it is used in many industrial sectors such as automotive, aerospace, high tech, oil/gas and so on. Today, industries understand the advantages of using CFD simulations during the product development process especially when their products are impacted by fluid flow, heating, cooling and so on [1]. The manufacturers' goal is to bring high quality products to the market quickly, keeping the costs down. The top external pressure factors that lead the product development are: time-to-market, quality and cost [2]. These factors are often in conflict together [3]. Products are becoming more complex. They have a high number of components and configurations. Brands offer many options and the customers can combine the options together to get their configuration [4]. Engineers need to understand how the components interact together and to assess the performance of each configuration under many physics conditions. The prototypes and the infrastructure that are required to asses the prototype performances (e.g. the wind tunnel for the automotive sector) are expensive. So, it is often expensive and time-consuming to assess the product behaviour for every physics condition within the budget constraints, especially for the extreme conditions.

A benefit of using computer simulations is a reduction of the number of the physical prototypes that leads to a cost reduction. According to the market research "*Engineering Evolved*" [3] three or more different types of simulations are able to reduce the number of prototypes by 37% and CFD is one type of simulation [2]. CFD simulations are able to reduce the number of physical prototypes [3], to manage the overall product system complexity and to get a better insight into product behaviour since the initial development process stages [2].

Standard CFD Workflow phases are shown diagrammatically in Fig. 1: pre-processing, solving and post-processing phases. Industries are looking for an increasingly integration of the CFD Workflow into the business process [5]. This integration heavily depends on internal team organization, industry's best practices, the product development plan and industry's policies. Of course, CFD software are designed so that they focus mainly on the CFD simulations; business process integration is not their main goal and, it is often completely ignored. The main benefit of integrating CFD Workflow into processes is to promote collaboration between CFD engineers, design engineers and analysts [2]. In an industrial context, CFD simulations are just a part of a more wide and complex product development process, so engineers perform additional tasks to standard CFD Workflow. For some products, design and CFD tasks can sometimes be done by the same team, using integrated products (like SolidWorks and CATIA). In the automotive industry (and in other fields like aerospace) the teams are different and therefore the CFD workflow is distinct from design. Besides, in the automotive industry CFD engineers perform different types of CFD simulations on the same product (i.e. external aerodynamics, aeroacustics, air conditioning and engine thermal analysis), so they need to use different CFD software because each one is validated for its own area of application. We observed that many engineers' tasks are repetitive, error-prone and time-consuming (e.g. find and compare simulations results, document generation, parametric studies) and they can be automated. Another issue is the simulation data and results accessibility. All engineers, both CFD and design engineers, must have full and user-friendly access to centrally managed simulation results.

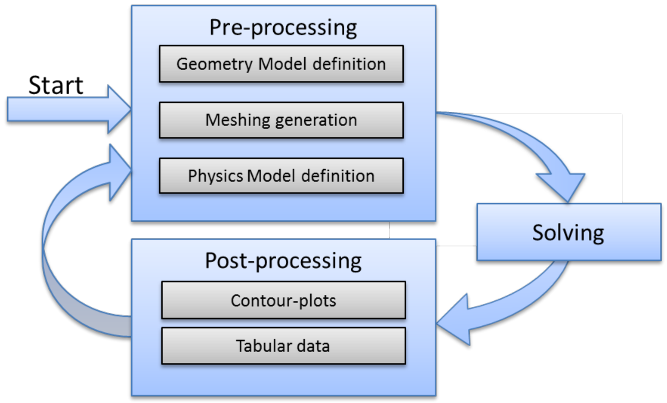


Fig. 1. CFD Workflow.

Our paper presents and validates a flexible architecture to manage the CFD Workflow. Our idea is to design and develop an architecture that enables the creation of tools to use within CFD Workflow. Architecture is the foundation for all future services upon existing CFD software.

The paper is organized as follows. Section II presents both functional and non-functional requirements for the system to design and it also reports some aspects of engineers' tasks to automate. Section III describes designed architecture to meet non-functional requirements. Section IV presents Floasys the prototype of the architecture that we are developing and some built tools.

II. Managing CFD Workflow

During the last year we have worked closely with a team of professionals that extensively use CFD, analysing their needs and supporting many of the findings and suggestions from literature and recent survey in the field. The professional team is composed by four highly skilled engineers with a strong CFD experience. They work in the automotive industry and they are not computer scientists. The issues that we are facing within automotive sector seem to be very common issues also in other sectors. Aberdeen Group market research [2], through a survey and interviews, has studied the experiences of 704 companies about the use of CFD to design products. Companies belong to industrial equipment, automotive, aerospace and defense, high tech, oil/gas and military/public sectors. In the analysed automotive context we observed issues that are almost similar to what has been outlined at the end of Aberdeen Group study. So, we hope that our analysis, that is based on a limited context (i.e. automotive), can be successfully translated into other similar contexts, as well.

Our initial approach is based on an empirical study through direct observations about how engineers work on simulations and discussions with them. At beginning we have tried to identify error-prone, repetitive and time-consuming tasks. So, we have focused on single team member tasks and we have identified initial functional requirements. In this step we have realized that many tasks (e.g. simulation finding, document generation) are not covered by CFD software, because they depend on internal organization and we have pointed out the need for simulation data sharing.

Our claim is that industries require many services upon and correlated to CFD simulations. These services are important in order to reduce costs, time and to improve engineers' tasks. Besides, there is a strong need to increase the integration of CFD Workflow into the overall product development process. So, our idea is to design an architecture for CFD Workflow management that supports the creation of services within the CFD Workflow, independently by and upon CFD software. This section describes some gathered functional requirements and it covers non-functional requirements that have led the architecture design.

A. Functional Requirements

The following are the identified actions to perform in order to improve industrial daily CFD engineers' work:

- centrally manage simulation data;
- identify and automate tasks through software tools;
- take advantage of wizards and templates.

1) Centrally manage simulation data: The aim is to improve the simulation data accessibility and to promote knowledge sharing among engineers.

We observed that this action is already performed by storing all the simulations data in a proprietary file format within one or more shared network folders. Engineers usually need to find the old simulation data stored in the repository to compare the results. It is interesting for the engineers to search the simulations stored in the central repository by the simulation revision name, the physic properties and other tags defined by the end-users. The simulation files are often binary files, so the engineers can not use the search tools based on text content.

We think that simulation data tagging, linking and searching are interesting features. The file system has a hierarchical structure and it is not enough to meet these described criteria.

According to Aberdeen Group study [2], in order to improve their performances, companies should "*Centrally manage previous simulation results*". It is not useful for engineers to store centrally only the simulation results: they need both the simulation results and the configuration case data. So, we decided to centrally manage the configuration case with all settings (e.g. boundary conditions, physical properties, the max number of iterations), the solving logs, the convergence charts and results (i.e. contour-plots and tabular data). Over the years industries perform a high quantity of simulations and each simulation file takes up over ten gigabytes due the geometry model details. Engineers usually need to find simulation data to compare the results.

We think that a central repository for all simulation data provides an historical view for a given project and a strategic competitive advantage for the future. Simulation central repository is the knowledge base on which apply metrics, perform statistics and make strategic decisions.

In order to centrally manage simulation data, we have identified the following functional requirements: (1) *centrally manage configuration case, solving logs, convergence charts and results*; (2) *simulation data tagging*; (3) *search based on simulation data and simulation tags*; (4) *automatic documents generation from simulation data* and (5) *version control*.

2) Tasks automation: Here we report some examples of tasks to automate, considering that this paper focus on the architecture design rather than on a particular tool. CFD Workflow needs a better integration in the product development process. For example, engineers create many documents based on the simulation experiments and results. They always use the same document structure but with different data. So, the first task to be automate is the automatic document generation. The first time engineers will prepare n document templates, each one with its basic structure, and will store them in the system. In order to generate documents, the engineer then selects one or more simulations and chooses a document template; the system merges the simulations data and results with the chosen template. The CFD team is highly skilled and has a lot of experience with command line tools, but as reported in literature this requires high training costs. A common CFD engineer task is to run simulations using the HPC systems.

Usually, engineers use a small set of commands through an ssh connection to submit/kill a job and to monitor the running jobs. Engineers need to do some other operations in order to monitor the simulation convergence data. Our idea is to create a Monitoring Tool: a workbench that provides in one view the job queues and the convergence charts for the running simulations.

3) Take advantage of wizards and templates: Wizards and templates guarantee that all team members work in the same way. Besides, wizards incorporate the best practices and support less experienced users. Both wizards and templates are a good way to support engineers in the repetitive and error-prone tasks.

B. Non-Functional Requirements

Our goals are to identify leading non-functional requirements and to find a trade-off among them. Non-functional requirements gathered from our analysis are: *extensibility, modularity, open, portability, Intranet-based* and *deployment*.

1) Extensibility: Extensibility is the ability of a software system to allow and accept significant extension of its capabilities without major rewriting of code [6]. Extensibility answers to the question: how easy is it to add new functionality to the system [7]? In order to add future functionalities that depend by industry's product development process and by particular CFD team, the system should be open for future extensions. Industries want functionalities tailored to their needs and they want to develop their in-house tools. Industries initially use the base set of functionalities, then in order to accommodate future needs they can develop new functionalities. For example, each industry uses its internal document template; the system must be extensible in order to support future format and future changes in document templates.

2) Modularity: Modularity is the degree to which a system or computer program is composed by discrete components such that a change to one component has minimal impact on other components [6]. A module is a logically separable part of a program [6]. The software system is made by modules that fit together to create the overall system. Modularity advantages are: module replacement, creation of new modules and a well-structured system. With modularity, it is possible to identify which services are provided by each module. The system is tailored to customer needs: each customer can compose a system loading just the need modules.

3) Open: The system must use open protocols and open formats to avoid vendor lock-in and to be interoperable with other systems. Vendor lock-in is the phenomenon that causes customer dependency on given vendor with regard to specific good or service [8]. We observed a potential CFD software vendor lock-in data format that occurs when end-user stores data in a proprietary format and the proprietary software does not have import/export functions to an open format. CFD Workflow requires the use of many different software: CAD/CAE, CFD and post-processing software. Before choosing a software, industries estimate the benefits in using open formats. Nowadays, industries use open formats to store and to exchange geometry data between CAD/CAE software and pre-processing tools. IGES and STEP are well-established open formats for geometry data exchange. Industries usually do not store CFD simulation data in an open format. CFD General Notation System (CGNS) is a standard for CFD input and output, grid, flow solution and boundary conditions [9]. CGNS concerns with CFD data representation and can be used also for data exchange [10]. A software must have import and export options [11] to open formats, but not all software vendors offer these functionalities. In our case study, simulation data such as geometry mesh, physical model, convergence data and results are stored into proprietary file format and the proprietary software does not have the export option to CGNS format. This practice is due to proprietary software adoption. Only simulation results are exported in a neutral format.

4) Portability: Portability is how easily a system or component can be transferred from one hardware or software environment to another [6]. Our goal is to build tools and automated procedures that run independently by CFD software. We can say that tools are portable across CFD simulators.

Many CFD software provide a script language which enables engineers to extend software functionalities. Over the years, engineers have written their own scripts to automate the execution of tasks. Scripts are very important for the manufacturers because they contains the experience of the engineers, the internal procedures and the business process practices. Scripts can increase productivity because they automate tasks, but they can also create a potential vendor lock-in problem: scripts are fitted on particular CFD software features and are not portable across CFD software. Our aim is to create automated procedures that run over any CFD software. A solution to vendor lock-in anti-pattern is to design the system with an isolation layer [12].

5) Intranet-Based: Engineers access to services, tools and resources within the company Intranet. In using the system, engineers expect to have Intranet performances, such as high bandwidth, low latency and good response time.

6) Deployment: Deployment into an existing industry can be expensive especially when it must be done on each client workstation. System design must take in account the future deployment costs. The system should reduce costs and time for deploying and updating the system on all clients.

III. System Architecture Design

This section reports the design decisions made to achieve the requirements presented in Section II. Here, we intentionally do not mention any particular software technology, and we speak about patterns, architectures and protocols to guarantee a future reproducibility of our architecture. In the first part we describe a client/server system architecture taking in account both hardware and software, then we give more details about the server-side software architecture.

A. System architecture

Our system has a client/server architecture (Fig. 2). The clients are web-based, so the end-users use their web browser

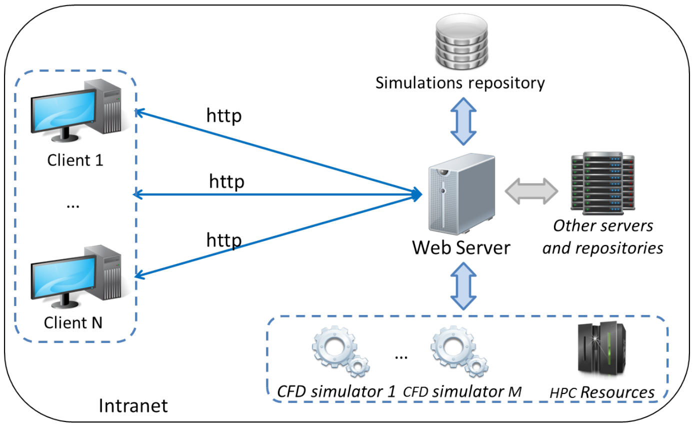


Fig. 2. System Architecture.

to access to the simulation data, HPC resources and CFD functionalities provided by the central web server. The server exchanges data with the simulation repository that provides the persistence service. The simulation repository stores the simulation data: an important asset for the industry over the years. It stores data in an open format (e.g. CGNS) and guarantees version control. We aim to store and to put under version control geometry model, physics model, boundary conditions, charts, simulation results, contour plots and generated documents. The server does CRUD (create, read, update and delete) operations on the repository. However, nowadays each simulation file takes up gigabytes of disk space and over the years the number of simulations grows incredibly: it is not practical to store all data for all simulations. Manufactures (based on their needs) should choose which data must be stored. Often, engineers perform parametric studies on the same geometry model changing only the physic values. In these cases, it is convenient to store the geometry only one time and correlate simulation results, physical model and the configuration case to the same geometry model. Sometimes, it is convenient to not apply version control to the geometry model. In our analysis, CFD engineers do not run and solve an old simulation, they only need data for results compare. Engineers can access to configuration case made by physical model, charts, tabular results and contour plots without run CFD software again. We add metadata and services (like versioning) to existing simulation data.

Industries use CFD simulators that only work with proprietary file format not allowing the import/export in an open format. In this situation it is still feasible to have also a vendor format independent central repository that stores simulation data in a neutral format. Our solution is to convert proprietary format data into an open format and to adopt an isolation layer [12] between CFD simulators and tools (the solution is detailed exposed in the next section III-B).

The central web server interacts with other servers and repositories. For example, companies usually already have internal servers that provide security and authentication services, so our architecture does not directly provide them but relies on other existing Intranet servers. The web server directly

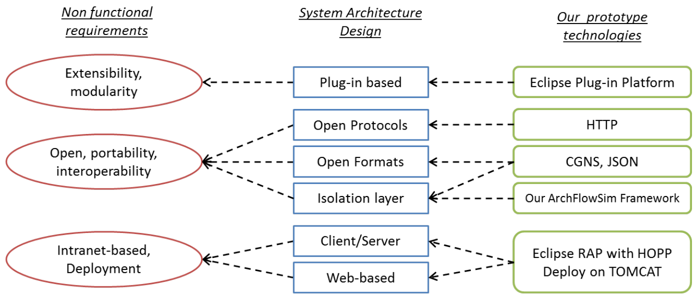

Non functional requirements
System Architecture Design
Our prototype technologies
Extensibility, modularity
Open, portability, interoperability
Intranet-based, Deployment
Plug-in based
Open Protocols
Open Formats
Isolation layer
Client/Server
Web-based
Eclipse Plug-in Platform
HTTP
CGNS, JSON
Our ArchFlowSim Framework
Eclipse RAP with HOPP Deploy on TOMCAT

Fig. 3. System Requirements Mapping.

connects to HPC resources and it makes them available to end-users. Job submission, job kill, simulation monitoring and real-time convergence charts are all services available to non-HPC system experts.

B. Server-side software architecture

The server-side software architecture (Fig. 4) consists of three layers. The tools are in the top layer. A CFD tool is an end-user GUI that provides one or more useful functionalities to engineers. The middle layer is the isolation layer and finally, the bottom layer consists of CFD software wrappers.

Each tool performs a well-defined engineering task. Some of the tools depend on the internal team organization and by the business process. For example, document generator tool accepts document templates and creates documents from them. The server-side architecture is based on a pure plug-in architecture in which everything is a plug-in [13]. Each box of the software architecture (Fig. 4) is a plug-in. To achieve extensibility and modularity requirements, we choose a pure plug-in architecture because it supports the extensibility by plug-ins. Every plug-in provides well-defined hook points called *extension points* that describe the way to extend the plug-in's functionality. Other plug-ins can add new functionalities by implementing an extension point. A plug-in can be modified or replaced by another equivalent implementation.

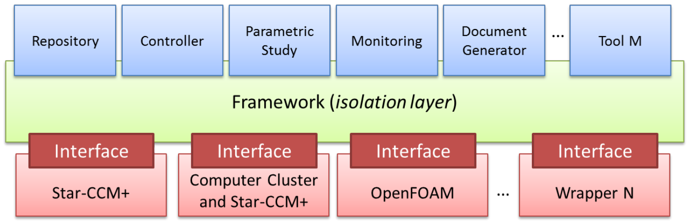


Fig. 4. Server-Side software architecture.

Our architecture supports the concurrent use of multiple CFD software for companies who have decided to employ, for example, both commercial CFD software (e.g. CCM+ developed by CD-adapco) and OpenFOAM as reported in [11].

Plug-in based architecture meets the extensibility and modularity non-functional requirements. Open data formats and isolation layer meet the open and portability non-functional requirements. Open protocols and open formats guarantee the future interoperability with other software. The client/server and the web-based architecture avoid the installation of the software on each workstation. Worries about responsiveness and bad tolerance of networks outages, typical disadvantages of these architecture, are mitigated by the Intranet setting.

IV. Floasys Prototype Architecture

Our aim is to design and implement an architecture that provides software reusable-building boxes to create new CFD tools upon any CFD simulator. This section presents the prototype based on the architecture described in Section III: it is currently under development and on site testing. It has a pure plug-in architecture [13] based on the Eclipse Platform [14]. Floasys's plug-ins can be arranged in three layers as shown in Fig 5: (1) The top layer consists of CFD tool plug-ins. Each tool can add a new perspective or can add a new view to an existing perspective. Usually, tool design is based on the MVC pattern. (2) The middle layer is the isolation layer. It provides the core API and the common simulation model. Plug-ins in the middle layer provide services to up layer tools by abstracting CFD simulators on bottom layer. (3) In the bottom layer, simulator wrappers communicate and exchange data with CFD simulators.

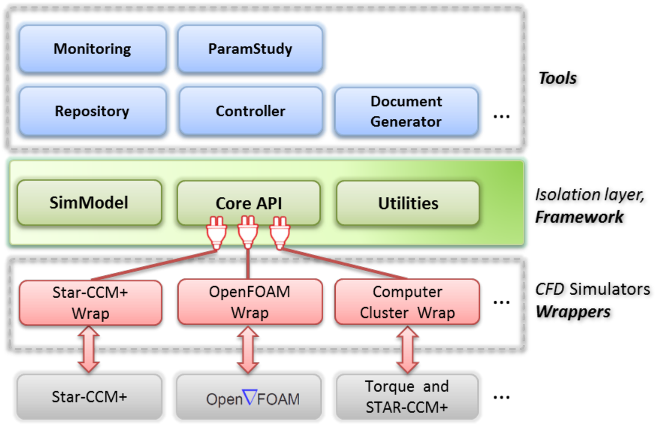


Fig. 5. Floasys Architecture.

Floasys is a web-based client/server architecture (Fig. 6). We use the Eclipse Remote Application Platform (Eclipse RAP) to develop the web application. RAP core implements [15] the Half Object Plus Protocol pattern [16]. Our assumption is that the system runs on the industry Intranet infrastructure, so the use of HOPP pattern is not in contrast with the end-to-end principle [17] because Intranet provides high bandwidth, high availability and low latency compared to Internet connection. The server-side software needs a JEE servlet container (e.g. Apache Tomcat) and a framework for the plug-in life cycle management (e.g. Eclipse Equinox OSGi).

A. Simulation Model

The Simulation Model (SimModel) stores the simulation data. Its aim is to store everything about the simulation not only the geometry, so it stores the configuration case with all settings and results. For example, it stores the boundaries, the physical properties, the stopping criteria, the log files, the outcomes, the charts, the tabular data, the contour plots and the generated documents. SimModel is a tree-like data structure

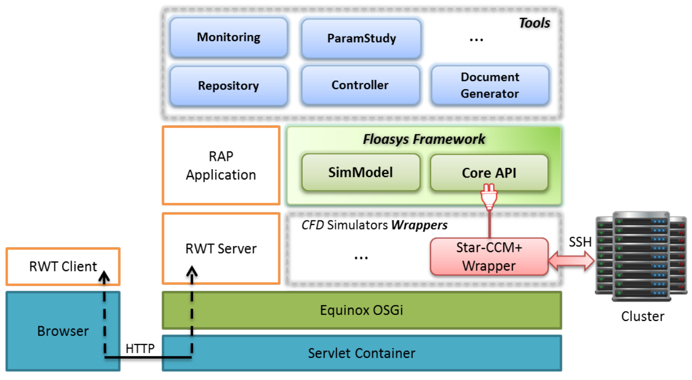


Fig. 6. Client/Server architecture.

as in CGNS standard [9], so it can store each information about the simulation with a tree node. The system can also link other metadata (e.g. the tag names) to the simulation by adding a new tree node. CFD tools should be independent from any particular CFD simulator and should be used across many CFD simulators. To obtain such simulator independence, we have developed the SimModel plug-in showed in the middle layer of figure 5. SimModel is the only joint point between CFD simulators on bottom layer and CFD tools on top layer. Tools implementation is based only on the simulation model and they interact only with simulation model.

B. Core API

From our framework point of view, CFD simulators are black boxes which we can extract model information from and interact with, in order to change some properties. Each simulator has its own internal data model; it consists of geometry model, regions, boundaries, physical properties and so on. Framework API allows to extract simulation data stored inside a proprietary CFD simulator to obtain the framework simulation model for a given simulation. Framework relies on CFD wrapper simulator implementation to extract or change simulation data. At the end of the day we are interested in collecting simulation data and building the simulation model. We store the built simulation model in an open source format (i.e. xml) to avoid the vendor lock-in issue and to provide other services such as the version control and the simulation finding by its data. Potentially, each CFD tool works independently by the CFD engine technology (e.g. operating system, programming language, proprietary and open source software) because each tool interacts directly with the simulation model.

C. CFD Wrappers

CFD wrappers embed CFD software features and provide them through a common interface defined by the Core API. We have categorized CFD simulators by the source code availability and open protocols implementation. CFD software wrappers currently available in our prototype are: Star-CCM+ on Torque cluster and OpenFOAM CFD codes. Star-CCM+ is a proprietary software developed by CD-adapco while OpenFOAM is open source; both CFD software do not directly implement an open format (e.g. CGNS). For

OpenFOAM it exists a converter called *foamToCGNS* available into OpenFOAM Extend Project. From the technical point of view, the interaction between CFD simulators can be based on simulator's API invoking, command line simulator execution or network based interaction. The implementation of a wrapper is a non trivial task and is widely studied; literature describes many techniques and automate wrapper generators [18] (e.g. CORBA wrappers generators). Star-CCM+ Wrapper on bottom layer (Fig. 6) interacts with a computer cluster through SSH to submit jobs, to handle job queue, to monitor the running simulations, to modify simulation file and to extract information from simulation.

V. CFD Tool examples

This section describes briefly some CFD tools actually developed and based on the above architecture. A CFD tool is an end-user GUI that provides one or more useful functionalities to engineers. Currently available tools are: the repository tool, the simulation controller tool, the parametric study tool, the simulation monitoring tool and the document generator tool.

Floasys application GUI is based on the perspective and view concepts, a traditional GUI organization in Eclipse [19]. Each perspective is a visual container for a set of views. Perspectives support the task oriented interaction [20], the CFD engineer will use a different perspective depending on the task to perform. Each tool can contribute with a new set of views arranged in one or more perspectives or it can contribute to an existing perspective with a new view.

The Simulation controller perspective shows data about the selected simulation; it shows simulation data in a tree-like structure (Fig. 7). This perspective is the main perspective because it allows to access to other tools such as document generator tool and parametric study tool. For example, from the simulation tree the user can drag-and-drop simulation items in the parametric study tool. In an industrial context, CFD engineers perform several simulations on the same model by changing the set of parameters values. Engineers run many simulations about the same vehicle model; each simulation runs with a different inlet velocity value and produces a different result. Finally, all results are compared together. Parameter study aim is to assess how a variation of a parameter value affects simulation solution and which impact has on design. Parameter study can be laborious, tedious, repetitive and error-prone task without an automated tool [21].

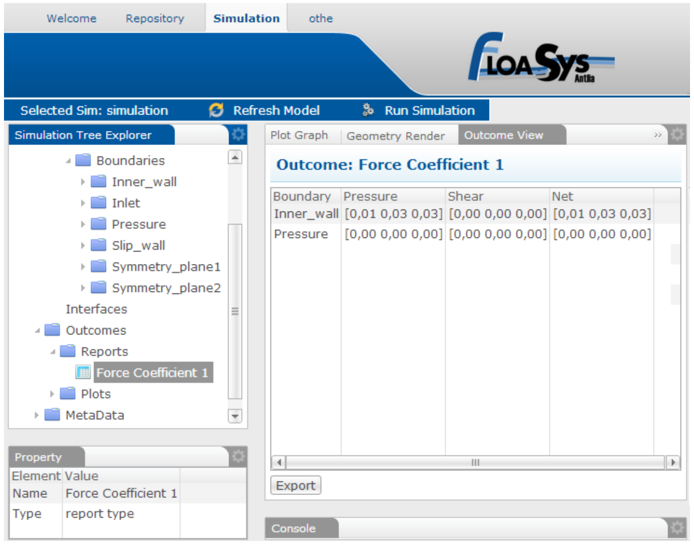
VI. Acknowledgments

The authors gratefully acknowledge the help and the support of Comprensorio CRF Elasis Pomigliano (Fiat Group Automobilies) and, the interactions with Aerothermal CFD team. In particular, the work has been significantly improved by the interesting and stimulating discussions with Antonio Cucca and Ugo Riccio. The authors also thank the anonymous reviewers for useful comments.

References

[1] M. Boucher, "The ROI of Cuncuttent Design with CFD," 2011.

[2] C. K.-R. Michelle Boucher, "Getting Product Design Right the First Time with CFD," 2011.



Outcome: Force Coefficient 1

Boundary	Pressure	Shear	Net
Inner_wall	[0,01 0,03 0,03]	[0,00 0,00 0,00]	[0,01 0,03 0,03]
Pressure	[0,00 0,00 0,00]	[0,00 0,00 0,00]	[0,00 0,00 0,00]

Element	Value
Name	Force Coefficient 1
Type	report type

Fig. 7. Simulation controller perspective.

[3] D. H. Michelle Boucher, "Engineering Envolved: Getting Mechatronics Performance Right The First Time," 2008.

[4] J. Weber, *Automotive Development Process*. Springer, 2009.

[5] E. Sindhu, A. Lee, and S. M. Salim, "Coves: an e-business case study in the engineering domain," *Business Process Management Journal*, vol. 10, no. 1, pp. 115–125, 2004.

[6] A. Geraci, F. Katki, L. McMonegal, B. Meyer, J. Lane, P. Wilson, J. Radatz, M. Yee, H. Porteous, and F. Springsteel, "IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries," 1991.

[7] B. Bruegge and A. H. Dutoit, *Object-Oriented Software Engineering Using UML, Patterns and Java-(Required)*. Prentice Hall, 2004.

[8] R. Shah, J. Kesan, and A. Kennis, "Lessons for open standard policies: a case study of the Massachusetts experience," in *Proc. of the 1st inter. conf. on Theory and practice of electronic governance*, 2007, pp. 141–150.

[9] T. H. Christopher L. Rumsey, Bruce Wedan and M. Poinot, "Recent Updates to the CFD General Notation System (CGNS)," *50th AIAA Aerospace Sciences Meeting*, 2012.

[10] M. Poinot, M. Costes, and B. Cantaloube, "Application of cgns software components for helicopter blade fluid-structure strong coupling. 31st european rotorcraft forum," *Florence, Sept*, 2005.

[11] V. Bertram and P. Couser, "Aspects of Selecting the Appropriate CAD and CFD Software," *9th Conf. Computer and IT Applications int the Maritime Industries (COMPIT). Gubbio.*, 2010.

[12] W. H. Brown, R. C. Malveau, and T. J. Mowbray, "Antipatterns: refactoring software, architectures, and projects in crisis," 1998.

[13] D. Birsan, "On plug-ins and extensible architectures," *Queue*, vol. 3, no. 2, pp. 40–46, Mar. 2005.

[14] J. Des Rivières and J. Wiegand, "Eclipse: a platform for integrating development tools," *IBM Syst. J.*, vol. 43, pp. 371–383, 2004.

[15] Last checked on May 24, 2013. [Online]. Available: http://eclipsesource.com/blogs/2013/02/01/rap-2-0-countdown-15/

[16] J. O. Coplien and D. C. Schmidt, Eds., *Pattern languages of program design*. NY, USA: ACM Press/Addison-Wesley Publishing Co., 1995.

[17] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Trans. Comput. Syst.*, pp. 277–288, 1984.

[18] S. J. Eric Wohlstadter and P. Devanbu, "Generating Wrappers for Command Line Programs: The Cal-Aggie Wrap-O-Matic Project," *Proc. of the 23rd Inter. Conf. on Software Engineering*, 2001.

[19] D. Rubel, "The Heart of Eclipse," *Queue*, 2006.

[20] D. Springgay, "Using perspectives in the eclipse ui," *Eclipse Corner Article, Object Technology International, Inc*, 2001.

[21] M. Yarrow, K. M. McCann, R. Biswas, and R. F. V. d. Wijngaart, "An Advanced User Interface Approach for Complex Parameter Study Process Specification on the Information Power Grid," in *Proc. of the 1st IEEE/ACM Inter. Workshop on Grid Computing*, 2000, pp. 146–157.