Article 1

FSSteering: A Distributed Framework for Computational Steering in a Script-based CFD Simulation Environment

Christian Wagner[1,2], Markus Flatken[1], Michael Meinel[1],
Andreas Gerndt[1], Hans Hagen[2]
[1]German Aerospace Center, Braunschweig, Germany
[2]University of Kaiserslautern, Germany
christian.wagner@dlr.de, markus.flatken@dlr.de,
michael.meinel@dlr.de, andreas.gerndt@dlr.de,
hagen@informatik.uni-kl.de

In order to get insight into interesting flow phenomena, the traditional work-flow of computational fluid dynamics (CFD) consists of setting up and computing the flow field followed by a consecutive post-processing analysis. Only after this analysis one can identify parameters that may have been set wrongly in a configuration stage. Once these parameters are corrected, another time-consuming loop has to be started. To identify inadequate parameter settings already during the simulation run, online monitoring concepts were introduced. Combined with computational steering methods, parameter values can additionally be adjusted which eventually reduces the number of required iterations to yield satisfactory results.

At the German Aerospace Center, a comprehensive framework called FlowSimulator has been developed to offer a generic Python-based interface for the management of CFD simulations. It can easily be en-

hanced by add-ons. One of these extensions is *FSSteering* which is described in this paper in more detail. As a computational steering environment, *FSSteering* provides functionalities essential for interactive visualization and explorative analysis. Besides existing computational steering environments and frameworks, a user-centred and domain-specific view is proposed. Existing functionality can be reused without rewriting simulation code to enable for effective steering in CFD.

To be more efficient, components of the architecture are distributed across different resources. Whereas the CFD simulation typically runs on a parallel supercomputer, the visualization is carried out on a high-performance virtual reality system which allows interactive data exploration. The post-processing in between can be performed on the supercomputer or on a separate parallelization cluster. But it is also possible to switch between different existing post-processing toolkits. This is just possible because of the very flexible configuration management of the distributed steering framework. We will demonstrate the steering capabilities and the system flexibility by two current research examples. An outlook for future steps concludes this paper.

1.1 Introduction

In order to gain insights into complex flow situations, computational simulation is a common tool for modern engineers. Therefore, a computational fluid dynamics simulation is set up involving necessary parameters. After solving by a cluster or supercomputer post-processing algorithms are applied to generate visual feedback. Many parameters chosen wrong can only be identified at that point and potentially the simulation has to be done again with tweaked parameters. With iteration times of days or even weeks methods with higher productivity are desirable for providing quick research insights. Therefore, being able to check if a simulation was setup properly and is still on track is important. If possible, changes to guide the simulation should be applied during runtime.

To tackle this situation, computational steering systems were developed to interact with ongoing simulation runs. Most of the computational steering environments available are enhanced visualization tools and concentrate on data management providing meaningful visualizations. Complicated instrumentation of simulation codes is needed to make data available to those systems. Contrary, computational steering frameworks concentrate on easy simulation coupling with minimalistic interfaces. Visualization and analysis have to be implemented by the user. However, main drawback of both approaches is that all steerable parameters as well as callable methods have to be known at compile time.

In this paper, we introduce a domain-specific approach heavily depending on an existing scripting environment. We developed the computational steering environment *FSSteering* as an extension to the German Aerospace Center's computational fluid dynamics system FlowSimulator. For this reason, simulation scripts can be made steerable without internal knowledge by users. Nearly no instrumentation and data conversion is needed. Furthermore, steering commands to be executed are mostly interpreted by the *FSSteering* extension. Therefore, domain-specific tasks provided by the FlowSimulator environment can be executed or parameters can be changed during runtime without either being known to the simulation script or having to be implemented by CFD engineers . For example, the underlying mesh of any CFD simulation can be changed and adapted during runtime resulting in better simulation convergence without changes to the simulation setup or script.

Since CFD data are multi-modal with complex features, we coupled our systems with VRFlowVis, an explorative visualization systems including a parallel post-processing system.

The remaining paper is structured as follows: In the next section, related work in the field of computational steering environments and frameworks is reflected including an overview of FlowSimulator and VRFlowVis. Then, the developed *FSSteering* architecture is presented. In section 1.4, two steering examples are shown, followed by final conclusion and future work.

1.2 Related Work

Since steering simulations is of interest for many years now, a lot of work has been done. An overview of earlier systems can be found in [Mulder et al., 1998]. Online monitoring is essential to identify in what kind a simulation has to be steered. Therefore, most steering systems concentrate on visualization or are even enhanced visualization tools, like [Parker et al., 1997] and [Eickermann et al., 2005]. Native frameworks like [Jenz & Bernreuther, 2010] are available to enable for computational steering, but having high adaptation overhead to specific problems. [Coulaud et al., 2003] uses XML descriptions of simulation scripts to handle data and concurrency at instrumentation points. Only few existing systems try to tackle domain-specific requirements, one CFD-specific is [Kreylos et al., 2002].

The FlowSimulator is an open and efficient framework to unify massively parallel and multidisciplinary CFD simulations independantly from the tools incorporated [Meinel & Einarsson, 2010]. This is achieved by a layered approach. The FlowSimulator DataManager (FSDM) forms the common backbone and provides a common interface to store and exchange data in memory. Written in C++ it provides a number of classes that hold structured data typical for CFD-related numerical simulations. Using the automatic interface generator SWIG [Beazley,

1996] all of FSDM's interfaces are also provided in Python.

Explorative and interactive visualization is supported using the VRFlowVis application, a visualization frontend for steady and unsteady CFD data sets based on ViSTA and ViSTA FlowLib [Schirski et al., 2003]. ViSTA allows the frontend to scale from simple desktop systems to high-end immersive VR environments. ViSTA FlowLib is a specialized library that provides particular interaction methods [Wolter et al., 2007a][Wolter et al., 2006] and efficient rendering techniques for working with time-dependent CFD data.

The rendered features are extracted from the raw data and mapped to visualization components by a post-processing application based on Viracocha [Gerndt et al., 2004][Wolter et al., 2007b]. It is decoupled from the visualization frontend and distributed to High Performance Computing (HPC) resources, preferably the same resource used by the simulation to be steered. Visualization features are extracted in parallel, and as soon as first results are available the extracted geometry data is sent back to VRFlowVis to be rendered.

1.3 Computational Steering Architecture

The developed computational steering architecture aims on enabling computational fluid dynamics simulations to be steered with little impact on already existing simulation scripts. Therefore, *FSSteering* was developed as an extension to the FlowSimulator system providing easy access to existing functionality and coupling simulation scripts with parallel post-processing back-end as well as front-end systems.

In the target work-flow different connected computing systems are involved, c.f. figure 1.1. A supercomputer or cluster system is supporting a set of simulation tasks in batch-processing. To steer one of the running simulations on-demand different front-end and back-end systems need to be attached in a flexible connection topology dealing with heterogeneous networks.

1.3.1 System Architecture

The overall *FSSteering*-architecture can be seen in figure 1.2.

Although *FSSteering* makes use of the scripting interface offered by FlowSimulator, performance-critical tasks need to be implemented efficiently. For this reason, a core module provides connection handling and data transfer methods. Access to these functions is provided by lightweight APIs. The Python-API is also bound to the FlowSimulator-API allowing inheriting its functionality and providing it to the connected applications via command requesting.

Both, Python- and C++-API, are used in the runtime examples in section 1.4.

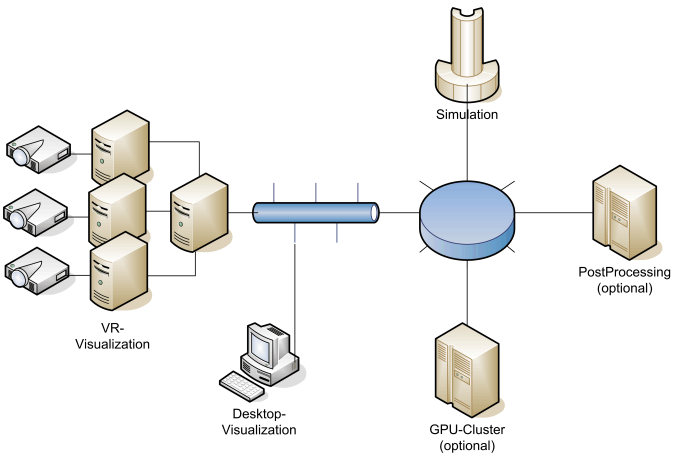


Figure 1.1: CFD simulations can be connected by multiple post-processing and front-end visualization systems on-demand. TCP/IP- as well as MPI-connections can be used for data communication.

1.3.2 Runtime Execution

At runtime a steerable simulations act as servers waiting for connections by clients. Clients again can act as servers allowing arbitrary connection topologies.

A connected client sends commands to the simulation server and waits for response. A set of predefined system commands exists for registration and updating variables and sending geometry or field data over connections. Calling domain-dependent FlowSimulator functionalities like mesh adaptation offer the possibility to change simulation behavior without being implemented in the application scripts in the first place. All commands unknown to *FSSteering* are assumed to be user-commands and are returned to the caller, e.g. the simulation script. For simple handling commands are represented as Python dictionaries including necessary parameters and are mapped to dictionaries of strings in the C++-API. Commands are sent through the system in a serialized representation. The interpretation occurs when triggered by the simulation.

The execution of commands is based on message queues. For command execution with centralized request management [Esnard et al., 2004] a simple, yet efficient synchronization scheme is used. All commands are gathered at a clients master node and are send to the servers master node. When a simulation triggers the processing of upcoming commands, the server broadcasts all new commands to the servers slaves. This choice perfectly fits to the single program multiple data programming model used in FlowSimulator scripts.

Special care was taken managing different connections in order to provide a flexible connection topology. Although command communication is always gathered

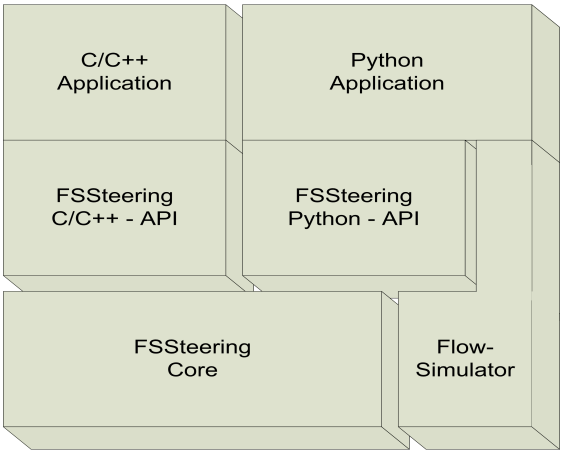
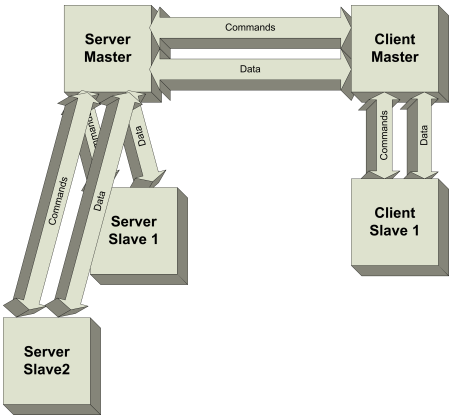
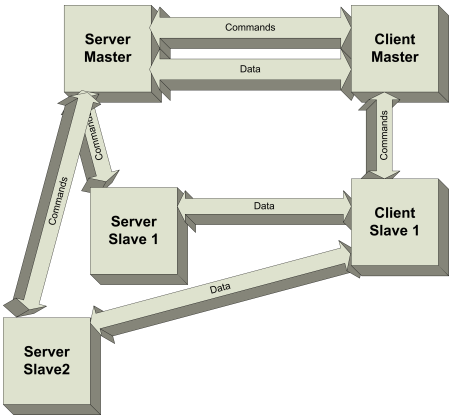


Figure 1.2: *FSSteering*'s main functionalities are implemented in the core module and made accessible by lightweight APIs. The Python-API can also use functions of FlowSimulator.

and scattered through the master nodes this does not hold for data communication. As depicted in figure 1.3, additional to 1-to-1 connection via master to master connection it is possible to establish n:m connections, where each server node is connected to an arbitrary client node. This setting is used in the steering examples of section 1.4. For general purposes, geometry and field data can be sent as raw binary data, the VTK file-format is also supported.



(a) 1:1-connection: Data and commands are gathered at master nodes and redistributed to slave nodes.



(b) n:m-connection: While commands are gathered and redistributed, data is distributed in parallel.

Figure 1.3: Commands are always gathered and redistributed in master nodes. For data, each data node can be connected to arbitrary client nodes.

1.4 Computational Steering Results

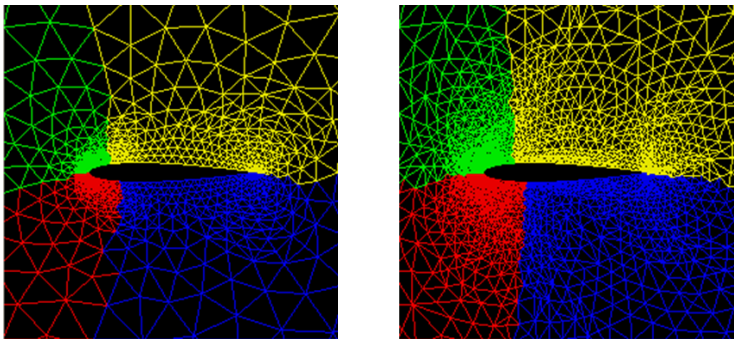
This section demonstrates the effective usage of *FSSteering* in two examples. A FlowSimulator simulation running on four computational nodes is made steerable using the *FSSteering* Python-API. The parallel post-processor Viracocha connects to the computational nodes via parallel data channels, one to each simulation node. Simulation and post-processor are controlled using a ViSTA front-end. In this setup we will show two frequent steering applications.

1.4.1 Numerical Steering

Since the underlying simulation mesh is essential for numerical convergence to physical meaningful results, the additional possibility to influence the mesh during runtime can prevent restarting simulation runs. Figure 1.4 shows the effect of additional adaptation runs initiated in the FlowSimulator environment. Note, that no additional code adjustment are needed since mesh adaptation is one of the algorithms provided in FlowSimulator and *FSSteering*.

1.4.2 Simulation Steering

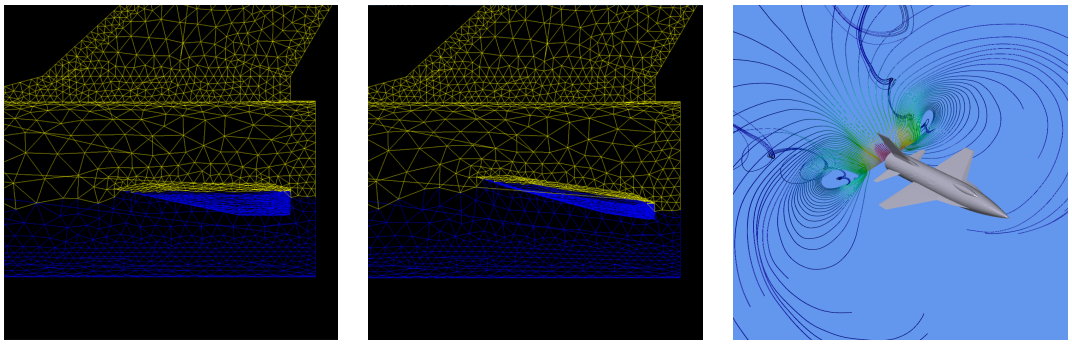
Contrary to the first example, this example shows how a simulation script is enriched with user-defined code, see figure 1.5. The used simulation script has the ability to change aileron, rudder and elevator angles in a synthetic aircraft model.



(a) Initial mesh. (b) Adapted mesh.

Figure 1.4: For an initial computational mesh (a) background adaptation is triggered improving numerical correctness.

FSSteering's abilities to schedule user-defined steering commands during runtime is used to successfully deform the mesh. Mesh deformation is controlled and viewed by the front-end application. Two wire-frame and a virtual reality view of the front-end is shown in figure 1.5.



(a) Initial configuration. (b) Changed angle. (c) Explorative view with changed angle in VR front-end.

Figure 1.5: In simulation steering a command to change elevator angle was triggered, (a) and (b). The influence on the flow field is analyzed in the explorative visualization environment, (c).

1.5 Conclusion and Future Work

In this paper we presented *FSSteering*, a flexible computational steering environment. As an extension to the FlowSimulator framework domain-specific needs of CFD engineers are addressed. A flexible connection and data management between the simulation on the one hand and front-end as well as post-processing back-end modules on the other hand was demonstrated. Existing FlowSimulator functionalities are inherited and can be used with very little programming effort. Simulation-specific functions can be added by users with a convenient Python-API. The combination with an existing parallel post-processor and a virtual-reality environment provides a rich set of analysis and explorative tools.

To serve typical batch processing systems running many simulations at the same time, future work will include management of running simulations to allow for a convenient selection of which simulation to steer. For simulations running with high counts of computational nodes additional data redistribution and streaming methods are needed to provide quick insights. Further research includes investigation of adequate interaction techniques in the front-end visualization to allow best usage of the functionalities provided for CFD simulations steered with *FSSteering*.

Bibliography

[Beazley, 1996] Beazley, D. M. (1996). Swig: an easy to use tool for integrating scripting languages with c and c++. In *Proceedings of the 4th conference on USENIX Tcl/Tk Workshop, 1996 - Volume 4* (pp. 15–15). Berkeley, CA, USA: USENIX Association. URL: `http://portal.acm.org/citation.cfm?id=1267498.1267513`.

[Coulaud et al., 2003] Coulaud, O., Dussere, M., & Esnard, A. (2003). Toward a distributed computational steering environment based on corba.

[Eickermann et al., 2005] Eickermann T., Frings, W., Gibbon, P., Kirtchakova, L., Mallmann, D., & Visser, A. (2005). Steering unicore applications with visit. *Philosophical Transactions of The Royal Society. Journal*.

[Esnard et al., 2004] Esnard, A., Dussere, M., & Coulaud, O. (2004). A time-coherent model for the steering of parallel simulations. In *Europar 2004* (pp. 90–97).: Springer Verlag.

[Gerndt et al., 2004] Gerndt, A., Hentschel, B., Wolter, M., Kuhlen, T., & Bischof, C. (2004). Viracocha: An efficient parallelization framework for large-scale cfd post-processing in virtual environments. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing* (pp.5̃0). Washington, DC, USA: IEEE Computer Society.

[Jenz & Bernreuther, 2010] Jenz, D. & Bernreuther, M. (2010). The computational steering framework steereo. In *Para 2010*.

[Kreylos et al., 2002] Kreylos, O., Tesdall, A. M., Hamann, B., Hunter, J. K., & Joy, K. I. (2002). Interactive visualization and steering of cfd simulations. In S. Müller & W. Stärzlinger (Eds.), *VISSYM '02: Proceedings of the symposium on Data Visualisation 2002* (pp. 25–34).

[Meinel & Einarsson, 2010] Meinel, M. & Einarsson, G. O. (2010). The flowsimulator framework to unify massively parallel cfd applications. In *Para 2010*.

[Mulder et al., 1998] Mulder, J. D., van Wijk, J., & Liere, R. V. (1998). A survey of computational steering environments. *Future Generation Computer Systems*, 13.

[Parker et al., 1997] Parker, S. G., Weinstein, D. M., & Johnson, C. R. (1997). The scirun computational steering software system.

[Schirski et al., 2003] Schirski, M., Gerndt, A., van Reimersdahl, T., Kuhlen, T., Adomeit, P., Lang, O., Pischinger, S., & Bischof, C. (2003). Vista flowlib - framework for interactive visualization and exploration of unsteady flows in virtual environments. In *EGVE '03: Proceedings of the workshop on Virtual environments 2003* (pp. 77–85). New York, NY, USA: ACM.

[Wolter et al., 2007a] Wolter, M., Bischof, C., & Kuhlen, T. (2007a). Dynamic regions of interest for interactive flow exploration. In *Proceedings of Parallel Graphics and Visualization 2007* (pp. pp. 61–68).

[Wolter et al., 2006] Wolter, M., Hentschel, B., Schirski, M., Gerndt, A., & Kuhlen, T. (2006). Time step prioritising in parallel feature extraction on unsteady simulation data. In *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization 2006*.

[Wolter et al., 2007b] Wolter, M., Schirski, M., Kuhlen, T., Bischof, C., Bücker, M., Gibbon, P., Joubert, G. R., Mohr, B., (eds, F. P., Wolter, M., Schirski, M., & Kuhlen, T. (2007b). Hybrid parallelization for interactive exploration in virtual environments.