A FRAMEWORK FOR APPLIED COMPONENT ZOOMING IN GAS TURBINES

D. Woelki & D. Peitsch, Fachgebiet Luftfahrtantriebe, Institut für Luft- und Raumfahrt
Technische Universität Berlin, Marchstraße 12-14, 10587 Berlin, Deutschland

Abstract

Much of academic research on gas turbines is conducted at component level. Promising concepts such as enhanced blade design or active flow control are often investigated on isolated domains, e.g. single compressor stages. Possible interactions with adjacent components, such as effects on secondary power and air provision or the thermodynamic cycle are rarely or just subsidiary considered. Nevertheless, the early inclusion of those effects is crucial for a reliable assessment of novel concepts' benefits. For example, the reduction of aero engine component weight or the increase of turbo-component stage efficiency is intangible regarding the quantified benefit on system level. This gap can be closed by a technique called component zooming. Here, a gas turbine performance model representing the cycle and components basic interaction is coupled with more detailed models which are capable of specific investigations on component level. This paper presents a suitable architecture of frameworks for component zooming in general.

In order to maintain high applicability especially for academic purposes, focus is put on flexibility in manifold sense: Adaptability to arbitrary concepts and zoomed components, expandability of available workflows, expandability to new disciplines, exchangeability of used software (e.g. commercial or in-house) and portability to common platforms. Various examples will demonstrate the capabilities of component zooming performed with such a framework. The first case discusses zooming on the secondary air system (SAS) of gas turbines. Another case demonstrates the coupling of engine performance with models for an initial assessment of aero engine components weight reduction benefit, e. g. realized by tandem blades or flow actuation in duct components. The resulting change in overall mass of the engine leads to basic, pragmatic options for aircraft operators.

Keywords: component zooming, preliminary design, gas turbine performance

NOMENCLATURE

Δ	[-]	Difference
Π	[-]	Pressure Ratio
η	[-]	Efficiency
Alt	[ft]	Flight altitude
err	[-]	Error vector
i	[-]	Running variable
Ma	[-]	Mach number
n	[-]	Amount
p	[Pa]	Pressure
s	[NM]	Aircraft mission range
T	[K]	Temperature
t	[min]	Time
v_{TAS}	[kt]	True air speed
w	[kg/s]	Mass flow
x	[-]	Guess vector

Indices:

CL	Creep life
CR	Cruise
date	Current value
eng	Engine
f	Fuel
mat	Blade material
mission	Flight mission
h	Hot gas
i	Running variable
PSN	Pre-swirl nozzle
ref	Reference value
rel	Relative value
SAS	Secondary air flows

Abbreviations:

A	Aircraft module
AFC	Active flow control
C	Correction model
CFD	Computational fluid dynamics
CPU	Central processing unit
CVC	Constant volume combustion
DLR	German Aerospace Center
E	Evaluation model
File I/O	File input/output writing and reading
G	Geometry model
GUI	Graphical user interface
HPC	High pressure compressor
HPT	High pressure turbine
IPSM	Interface for performance and secondary air system modeling
IRC	Internet relay chat
LAN	Local area network
LPT	Low pressure turbine
N	Network model
OEM	Original equipment manufacturer
OP	Operating point
OS	Operating system
P	Performance model
SAS	Secondary air system
SND	Swan neck duct
TOW	Aircraft take-off weight
XML	Extensible markup language

10.25967/490174

1. INTRODUCTION

The demand for the development in gas turbine technology is unbroken. Main drivers are today's ecological requirements and economical enhancements with certain legal requirements and certification standards. A frequent objective in gas turbine research is the increase of gas turbine efficiency. This can be achieved by novel cycles, integration of features and enhancements on component level. Examples for these three options might be pressure gain combustion, active flow control (AFC) or blade profile optimization. The range of possible measures is diverse and subject to manifold industrial and academic research.
Increases in component efficiency and the common increase of efficiency on system level may result in fuel flow reduction and hence reduced operational costs. In addition, the development is triggered by manufactural aspects, life time optimization and weight reductions. The latter is especially an objective for mobile applications of gas turbines. Thus, the successful integration of novel concepts must match a bunch of partly conflicting objectives. This can be ensured by multi objective optimization. Nevertheless, optimization on component level only, i.e. isolated from the context of the entire system, might still be insufficient: Interactions with aerodynamically, mechanically or thermally connected components must be considered. This usually results in highly iterative processes with other component departments.
Gas turbine performance is appropriate to represent the entire system and hence serves often as central discipline. Its inputs are best knowledge component characteristics. Typical outputs are interface or overall system quantities, e.g. inter-component mass flows, temperatures, pressures or power offtakes and fuel flows. However, gas turbine performance is not suitable for detailed component design.
The need for final component matching is initially not necessary for research on new concepts. Nevertheless, academic research in particular tends to focus on technological details, often without including effects at the overall system level. This also applies to the assessment of the actual benefit of concepts. For this reason, the justification of research sometimes cannot be perceived by the public. A suitable technique to counter this problem is the so-called component zooming: coupling of a whole system simulation such as gas turbine performance with higher fidelity component models. In this way, component zooming is closely related to conceptual pre-design. However, the latter typically includes the interdisciplinary design of all relevant components. In contrast, component zooming tends to be limited to one component. A complex redesign of other components is initially not intended.

1.1 Existing Solutions

For the purpose of preliminary design, different software exists, e.g. the NASA code NPSS [1], the MTU Aero Engines code MOPEDS [2] or the German Aerospace Center (DLR) code GTlab [3]. A computer-aided solution to facilitate and accelerate the iterative, multidisciplinary component design is developed with the participation of Rolls-Royce Deutschland in the research project VIT [4]. Existing solutions vary in different ways. Some software is company's in-house code and hence usually unavailable for other research institutions. Licensable software may not or only partially be extensible for specific purposes. Some key features may also be overload for academic research. For example, data management issues for the exchange of boundary conditions between different component departments such as described in [3] are vital in the multidisciplinary design process of bigger institutions or in companies. In university research, which is conducted in small groups or even by one person only, the resulting advantages are sometimes seen as disadvantageous or inefficient due to the complexity.
All presented characteristics of existing solutions can lead to a situation where the overall benefit assessment of novel concepts leads to the quick implementation of tools which are only suitable for studies on one specific concept. Adaptation to modified problems can then be difficult so that such tools are rarely recycled. Exemplary reasons are hard coded parameters, deep embedded interfaces to required simulation software or rigid, non-expandable equation systems in the case of iterative processes.
All in all, there is a dilemma between well-engineered, complex commercial or industrial solutions on the one hand and one-way solutions on the other, mostly university side.

1.2 Objectives

This paper has three major objectives. The first is to introduce the reasons to use component zooming in early concept development phases. For this purpose, exemplary concepts will be introduced (section 2.1).
When zooming is performed e.g. from gas turbine performance to one or more components, different models are applied. The resulting coupled model effectively implements a workflow, which must manage the exchange of each models boundary conditions and run simulations in the right order, iteratively if required. The second objective is therefore the formulation of elementary workflow features and a guideline for the design of workflows (section 2.2).
Third, important decisions will be discussed in order to create a suitable framework which allows such workflows to be run (section 3). A major topic will be flexibility in the problem definition itself as well as in the use of different simulation software.
In addition, the implementation of the framework IPSM (interface for performance and secondary air system modeling) is shortly presented (section 4), which both serves as an example implementation of the before discussed aspects and is the vehicle for studies on the previously introduced concepts. The results of these studies will be presented and discussed in section 5.

2. EXAMPLES FOR COMPONENT ZOOMING

The advantage to apply component zooming in gas turbine research and development is various. The subsequent examples are suitable to give first an overview about different feasible approaches and second to define the resulting requirements for workflow implementing applications.

2.1 Introduction of Concepts

The subsequent examples focus on three different components. The first concept aims for an off-design optimized secondary air system (SAS), here applicable in both stationary gas turbines and aero engines. The other concepts primarily deal with new technologies in the swan neck duct (SND) and high pressure compressor (HPC) of common turbo fan engines. Figure 1 sketches a turbo fan

with integrated thrust nozzle. The considered components are highlighted. Some axial sections of radial adjacent components of the SND and HPC are also highlighted in advance of some later introduced assumptions. It is worth to note that the depicted components don't match relative sizes in real engines. For example, the HPC is commonly longer than the SND.

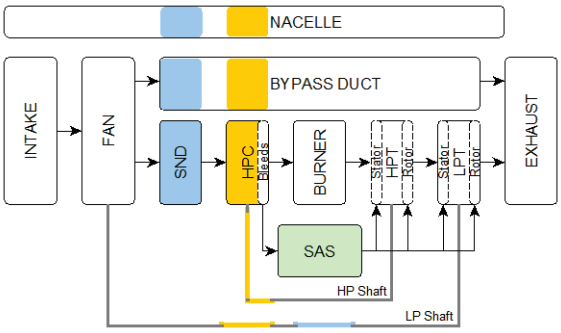

Figure 1 Sketched turbo fan engine with focused components and associated areas

2.1.1 Flexible Secondary Air Systems

The SAS is a subsystem, which provides secondary air for different purposes such as turbine blade and disk cooling or sealing against hot gas ingestion. Since most of secondary air is typically extracted in rear compressor stages, it is effectively bypassing the combustion process. The return to the main flow in different sinks of the turbine results in reduced turbine rotor work. In addition, secondary air flows are the source of several other losses. Especially cooling air flows with the highest share of secondary air are usually designed to critical operating points (OP) such as warm day take-off in aero engines or warm day full/base load in common stationary gas turbines. At OPs with demands for lower power output the amount of cooling air frequently exceeds the requirements. In contrast, other SAS requirements such as sealing of bearing chambers against oil emission might be critical at very low power output demand. Overall, the design of secondary air flows must meet the requirements of different operational conditions. Operation within the off-design space will usually result in the local provision of too much secondary air, resulting in avoidable losses. A flexible SAS which is optimized to meet each SAS circuits requirements in various OPs will reduce overall losses and may reduce the fuel flow w_f.
Nevertheless, the reduction of cooling air will result in higher blade metal temperatures T_{mat}, even if held on an acceptable level. The benefit of flexible SAS will hence result in a trade-off assessment between e.g. fuel burn and blade life reduction. This topic is also discussed in [5]. Finally, the zooming on the SAS as component requires various models. The most important models are the gas turbine performance model, which provides the SAS sources and sinks boundary conditions on the one side, and a SAS model suitable for investigations on modified circuits on the other side. The consideration of changes in T_{mat} and blade creep life t_{CL} requires additional evaluation models. All these models must be coupled in a workflow.

2.1.2 Aircraft Mission at HPC Modifications

Compressors in stationary gas turbines and HPCs in aero engines are key components regarding overall engine efficiency. The trend to higher overall compressor ratios is unchanged. Nevertheless, axial compressors are also subject to flow stability requirements. An increase of compressor stage number or specific work per stage can be addressed with multi-shaft design, enhanced contouring, optimized aerofoils or AFC. Besides that, shortening of the compressor is desired, since this component contributes to approximately 7-14 % share of total engine weight in common turbo fans [6]. Extra weight of new integrated concepts must at least compensate the penalty on aircraft operation by their benefits. The importance of HPC weight gets even more important, when surrounding components are considered as well: A shortening of the component would additionally result in reduced shaft, bypass duct and nacelle lengths.
As an example for novel concepts in axial compressors, this paper focuses on possible advantages of so-called tandem blades [7]. One intent of introducing tandem blades is the increase of compressor stage work by highly loaded blade rows still meeting the requirements of flow stability. An example for a tandem blade configuration is shown in Figure 2, here designed to achieve the same flow deflection as in the associated conventional blade row. The enhanced flow control by application of tandem blades could allow for the reduction of blades per stage, since the solidity can be increased.

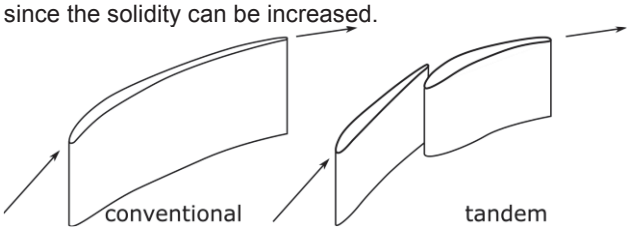

Figure 2 Conventional and tandem blade configuration

Analysing the overall impact of tandem blades requires an involvement of the aircraft and its flight mission. The effectively changed overall aircraft weight will result in either changed thrust requirements or a changed flight mission trajectory caused by altered accelerations. Both will result in a change of mission fuel burn, hence requiring engine performance as integral discipline of this topic. Compendious: Component zooming is applicable by coupling gas turbine performance with models for mass estimation, aircraft thrust demand and the flight mission itself.

2.1.3 Aircraft Mission at Shortened Duct Components

The SND is a critical duct component, which guides the low pressure compressor's outlet flow to the inlet of the HPC, typically at a much smaller radius (Figure 3).

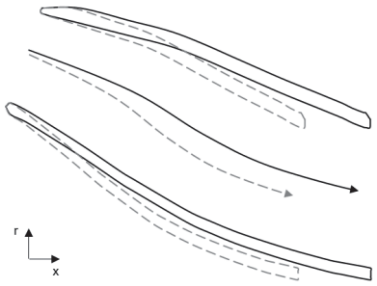

Figure 3 Principle of SND shortening, challenging flow control

Besides of this S-shaped deflection the front struts of the engine frame are crossing this duct, too. Even if the benchmark of individual SND weight is given to only 3-4 % of overall engine weight [6], surrounding components could also be scaled down in their axial dimension as described in sec. 2.1.2. Independent of the mass reduction effect are further possible benefits regarding a redesign of adjacent components. For example, a more aggressive deflection towards the HPC could allow for a design of the booster rear stages at higher radii, which could increase specific work or efficiency.
The challenge of SND shortening – basically a disturbed inflow of the HPC – as well as concepts to overcome this problem by end wall contouring is described in [8]. Another field of interest is the introduction of active flow control (AFC), e.g. realized with plasma actuators [9]. These plasma actuators are assumed to be lightweight and less power consuming. If information on objected component design and AFC is available, component zooming is again suitable to estimate the expected overall benefit.
This topic is basically familiar to the previously presented HPC shortening. Furthermore, AFC is topic of research in HPCs as well, but not emphasized in this paper.

2.2 Generation of Coupled Model Workflows

2.2.1 Basic Approach

The following describes the approach creating a workflow of a coupled model for component zooming. It is not this paper's objective to introduce single models. For more information about the applied disciplines and their typical models refer to [10], [5] (flexible SAS) and [11] (aircraft and flight mission model). It is also assumed that all required models are available and can be run as standalone models. The workflow definition just fixes their order and interactions.
Dealing with models of different order, or in other words dimensionality, is probably the major challenge in component zooming. The interfaces between models may be resolved in different ways. For example, gas turbine performance commonly considers only a small, fixed number of SAS sinks in the turbine. In the real engine, there are manifold sinks, which are typically represented in SAS network models by various axially and radially distributed sinks. This requires conversion methods of both sink boundary conditions (performance to SAS network) and sink flow results (SAS network to performance). Different approaches are available to overcome this issue, e.g. correlations, inter-stage characteristics or meanline models (e.g. [12]). The range of possible conversion methods indicates that workflows for the same problem can be implemented with different models. In this sense, flexibility for later implementations is desirable, too.
All models have to be arranged in a clear order, i.e. primarily sequential structured. Where iterative processes are required, those must be placed properly in the sequence, opening a child sequence on a lower embedded level. This means that the top level sequence will pause at this point, call an iterator which repeats the lower level sequence until convergence is reached and finally continue on top level.
Regarding iterative processes, two different types can be introduced:

1. Fixed-point iteration: Some matching procedures, where involved models are feeding themselves with new inputs are likely self-converging. This allows no hysteresis and demands high individual robustness of the simulations. In this case, the iterator just requires the definition of break criteria and will restart the iterated sequence until one break criterion is reached.
2. Complex problems and also those integrating simulations known to run not robust are not guaranteed to reach convergence with simple iterators. Replacing the iterator with an optimization algorithm may be the choice, since the algorithm is providing guess values for defined input parameters, following an individual approach in order to find convergence by evolving these guesses. The nature of optimization algorithms is of a different kind and should be fit to the problem to be solved iteratively. If an algorithm supports handling of side condition violations, which might be in general exceeding maximum or minimum values of any parameter, there is also the chance to treat robustness problems within the problem.

2.2.2 Coupling SAS with Gas Turbine Performance

Examples for workflows coupling SAS network models with gas turbine performance models have been subject of different publications, e.g. [13] referring the approach of [14], further adapted in [5] and [10]. The latter two will now be discussed in a scope, which focuses on the before mentioned challenges, hence using some simplified form of the workflows.
The very generic form of such a workflow is illustrated in Figure 4. In the focus is the matching between a gas turbine performance model (P) and a network model of the SAS (N). As mentioned before, different approaches exist for the conversion from the performance to the network model ($C_{P\rightarrow N}$) and reverse ($C_{N\rightarrow P}$).

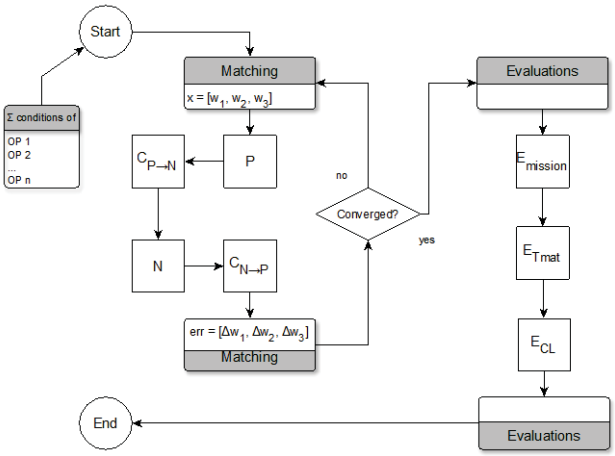

Figure 4 Rudimentary workflow for component zooming on flexible SAS

The here presented workflow is very generic and its adaptions to even different types of gas turbines will most probably look a little different. This can be also seen when comparing the workflows presented in [13] and [10]. Usually there will also be more than two correction models C overall. For example:

- One $C_{P\rightarrow N}$ model converts compressor bleed pressures and temperatures from P to boundary conditions of the sources in N
- Another $C_{P\rightarrow N}$ model converts sink pressures to sink boundaries in N
- The third $C_{P\rightarrow N}$ provides a hot gas temperature distribution within the turbine, which would be required in case of local hot gas ingestions to the disk wheel space
- The model $C_{N\rightarrow P}$ summarizes secondary air flows w_{SAS} simulated in N in order to feed them to P

In the depicted case, the gas turbine performance model would model three sinks w_{SAS} which have to be matched to the SAS network model flows summarized in $C_{N\rightarrow P}$. Changed w_{SAS} will result in an overall change of the gas turbine performance at remaining OP boundary conditions, hence changing source/sink pressures and temperatures. This part of the workflow is iterated until the errors between set mass flows in P and associated flows in N fall below a threshold value. The lower this threshold, the more accurate is the matching of the performance and network model.

In the end, the second, sequential part of the workflow evaluates certain requirements. In the here presented example for studies in aero engines, OP holding times in a reference flight mission are calculated ($E_{mission}$), blade material temperatures approximated (E_{Tmat}) and with the results of both the current OP's blade creep life is calculated (E_{CL}), see also [5]. The entire workflow may be repeated for several OPs considered in the reference flight mission.

What can be clearly seen in this example is the need for the introduced features, namely sequential arrangements combined with iterators and optionally the easy support of multiple workflow simulations at various boundary condition (OPs). The sketched workflow can only describe the structural arrangement of involved models. The workflow's structure is mostly mandatory. Here, only $E_{mission}$ and E_{Tmat} could be switched in place, because they are independent from each other and only providing E_{CL} with inputs. Even if this generic workflow looks small and straight forward, the number of inter-model parameter exchanges can easily reach some hundred. In this matter, it is important that the parameter exchange is not bound to certain sub-workflows. For example, the creep life estimation E_{CL} requires the shaft speed from P for the determination of current centrifugal stresses, blade coolant temperatures from N and hot gas temperatures from $C_{P\rightarrow N}$.

2.2.3 Aircraft Mission at HPC Modifications

New technologies in HPCs and turbo components can be initially investigated at component level. In many cases there will be focus on either loss reduction at maintained specific work or vice versa. Changed outlet conditions will affect components downstream. This will finally require a re-matching of the overall cycle. This is a typical task for preliminary design with gas turbine performance as the central discipline.

The here discussed introduction of tandem blades will not consider aerodynamics to hold down the complexity of this example. It is rather assumed that the application of tandem blades matches the performance of conventional blades. The benefit is assumed only to result in either stage or blade reduction. Recent studies such as [7] deal with stator tandem vanes. For this reason, the subsequent use of the term tandem blade refers to stator vanes only.

Engine mass changes have no practical relevance for the engine cycle when operating in steady state OPs. The effect of mass compensation addresses the aircraft mission and, if changes are significant, the aircraft design. Thus, the question of engine weight reduction benefit requires component zooming, which includes aircraft and flight mission models. The final question can be: How can an engine weight reduction decrease the mission fuel burn $m_{f,mission}$?

Important to the workflow design are the constraints which are formulated for the engine operation. There are two options:

1. The engines are considered to deliver defined thrust in order to match the flight mission's current OP target quantities, e.g. flight altitude or velocity, with the relative low pressure shaft speed or aircraft lift coefficient as control parameters, see [11]. This respects flight mission trajectory changes during phases of aircraft acceleration and deceleration.
2. The flight mission trajectory is held constant. This requires matching of each individual OP's thrust to the aircraft's current operational requirements.

The second option comes with a relatively simple assumption, but is chosen for the here presented scenario for the purpose of clarity. Since the flight trajectory and hence the aircraft OPs are fixed to the reference case – conventional instead of tandem blades –, the mission model $E_{mission}$ is not required to be iterated. It is arranged to be the initial model within the workflow. The definition of the here used long range mission as well as applied aircraft masses can be found in the appendix, Table 1.

The workflow for the here discussed concept studies is depicted in Figure 5.

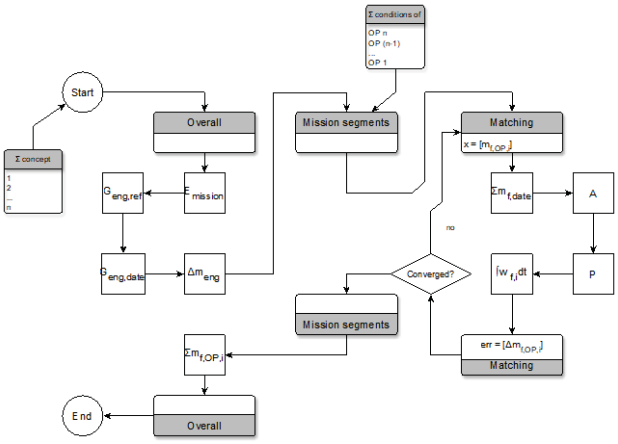

Figure 5 Workflow for engine mass effect on aircraft mission

Every investigated concept requires the determination of the change in aircraft operational empty weight, which is assumed to be the change in the weight of aero engines Δm_{eng} only. This is the difference of calculated masses in the geometry model $G_{eng,ref}$ of the reference and the current concept $G_{eng,date}$. In both models G an engine part weight method is applied which is inspired by feature based preliminary component design as presented in [15]. This model includes a detailed geometrical representation of all relevant engine components, which are subdivided to part and even sub-part level. This allows the scaling of individual parts in order to derive mass changes at first order. In predesign, such models would be extended with e.g. mechanics models in order to ensure durability to stresses. This is neglected in the here presented studies. The model $G_{eng,date}$ is rather stretching the vane geometry

as well as associated, radial adjacent parts in axial direction. This is exemplary sketched in Figure 6: Compressor stage shortening would include blades/vanes, associated casing structures, disk including drive arms and annulus hub wall. In addition, the low pressure shaft as well as bypass duct and nacelle would be affected – these components are not depicted here.

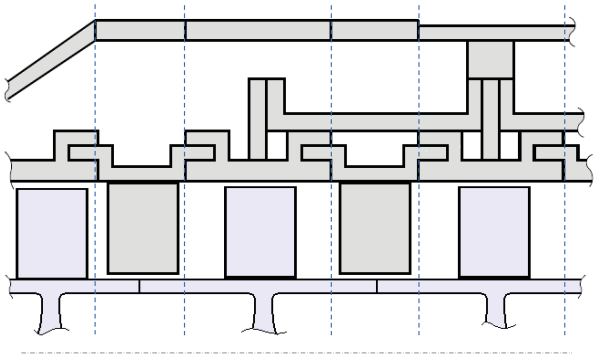

Figure 6 Principle of feature based HPC geometry model including axial segmentation to key parts (dashed lines)

Downstream in the workflow, two different types of iterative processes are embedded, see Figure 5. The middle part organizes the iteration over different OPs. Their ambient and flight conditions as well as individual range and holding times have been calculated in $E_{mission}$. Each OP's individual fuel burn $m_{f,OP,i}$ is unknown. For this purpose, the aircraft's thrust demand is derived from aircraft force balance according to [11]. The thrust demand is hence a function of lift, drag and weight forces. The applied polar curve relating lift and drag is unchanged and there is hence no detailed modeling of any changes in the aircraft's aerodynamics. The weight forces are calculated with the overall aircraft mass as unified point mass of all considered part weights.
The thrust result per engine from model A is set as target parameter to the engine performance model P with ambient conditions provided by the flight mission model. The current OP's fuel flow $w_{f,OP,i}$ is integrated over holding time $t_{OP,i}$. The result of the model sequence at the right hand side is $m_{f,OP,i}$. However, this quantity as well as the entire amount of fuel required in later OPs is an input to A, since it is one term of the aircraft's weight force. For this reason, an optimization algorithm or simple iterator is applied, providing a guess of $m_{f,OP,i}$, which is added in $\Sigma m_{f,date}$ to the fuel burn of all later flight segments. For this reason, the OP placed in the intermediate level of the workflow starts with the last OP in order to perform the matching for all OPs in reverse order. Finally, each OP's fuel burn is added to the entire mission's fuel burn. This workflow is repeated for different concepts.
The aircraft modeled in A is based on a common narrow body aircraft presented in [11]. The same applies for the engine in model P. Compared to the coupled aircraft-mission-engine model presented in [11], the here applied workflow connects $E_{mission}$, A and P as independent models, whereas the reference model just integrates the engine performance with lookup tables. In other words, the reference case, implemented in Matlab-Simulink, is a tailor made solution, optimized to computational performance.

2.2.4 Aircraft Mission at Shortened SND

This scenario is almost identical to the one presented in sec. 2.2.3. It uses the same base work flow. The only difference is the simulation within $G_{eng,date}$, which accounts for other geometrical changes. These are namely the SND component instead of the HPC and associated parts of the LP shaft, bypass duct and nacelle. It has to be noted that the areas highlighted in Figure 1 will not mandatorily reflect the parts to be modified in the objected studies. For example, the LP shaft segment in the axial region of the SND has some structural features, which cannot be directly scaled with the SND component. A shortening of the SND would most probably result in the adequate reduction of LP shaft length located more towards the middle of the shaft where no structural reinforcements can be found.

2.2.5 Considerations Regarding Models of Higher Order

For both HPC and SND modifications, the presented workflows do not consider aerodynamics as key discipline of the presented concepts. Fundamental assumptions are that the conceptual measures maintain the performance of their reference components. However, component zooming is intended to be free for extension with additional models. The additional integration of HPC meanline is advisable and even stationary CFD duct models are feasible. The extension of workflows as well as the type of integrated models always require the consideration of following questions:

- What is the benefit of probable accuracy enhancement regarding the overall quality of the concept studies to be carried out?
- How does a model from higher order increase the workflow's computational time? Will the model be a bottleneck?
- Is the originally zoomed component the overall workflow's bottleneck? If not, is the computational expense of large bottlenecks compatible with the requirements of the conducted concept studies?
- Is it possible to resolve the interfaces between models of different orders in a reasonable way?

3. FRAMEWORK ARCHITECTURE

The design of a framework architecture applicable for arbitrary use cases of component zooming, requires some decisions to be made at an early design phase. Programming a reusable and extendable framework might come along with relatively high expenditure in the beginning of this process, but will allow for very quick extensions with e.g. new disciplines or model approaches at a later point of time. Subsequently, such a framework will often be ready for simulations later than very tailor-made solutions. However, the adaptation of those tailor-made solutions to other cases is mostly relatively complex or even impossible and therefore often requires the development of a new solution. This section introduces those aspects, which are recommended to be considered from the beginning of the design process on.

3.1 Level of Model Integration

First to be discussed is the basic nature of concept studies to be conducted with the framework. In numerical gas turbine research, there are two major domains:

1. Novel concepts on component level: These demand for benefit assessments on the overall system, usually the gas turbine. A detailed component model is available and formally requires the extension with gas turbine performance.
2. Concepts on system level, e.g. novel cycles: The gas turbine performance model is available and subject to optimization studies. In order to increase the significance of the study results, key requirements are included via additional low-dimensional models. An example could be the estimation of turbine vane and blade temperatures as limiters, i.e. side conditions in the optimization.

In both cases, gas turbine performance is a key discipline. However, there is an additional use case of component zooming that doesn't aim for evaluations on whole system level:

3. Novel concepts on sub-component or part level: An example could be studies on overall compressor performance reflected by a meanline model, but coupled with a detailed high order model such as 3D CFD of a single stage.

All these three cases emphasize that the framework must allow for extremely flexible, arbitrary workflows. In consequence, the probably most important decision is whether the framework is supposed to be an independent application or not.
Regarding flexibility, the extension of e.g. a gas turbine performance code by integrating interfaces to potential other models should be seen as disadvantage regarding both arbitrary workflows and software integration. Another conclusion is that the framework should essentially contain a core process that connects the individual models with each other and organizes their data exchange.

3.2 Different Disciplines

Component zooming requires the capability to couple models from different disciplines with each other. The most obvious disciplines are gas turbine performance, aerodynamics (e.g. meanline, 3D CFD), network models (e.g. SAS or arbitrary pipe systems) or optimization itself. Depending on the kind of researched problem or modeling depth, mechanics and thermo-mechanics (e.g. FEM), aero elasticity and weight assessment might be necessary as well.
The core process of the framework must support all these disciplines by provision of interfaces, which are designed to enable the coupling of state of the art software. For this reason, all subsequent uses of the term discipline will refer to less physical disciplines than the nature of models, e.g. performance, network, meanline or CFD.
In addition, a totally generic interface is recommended to support any other disciplines, which are not yet considered.

3.3 Different Disciplines Implementations

There is usually a variety of software available, which features the same discipline. In the following, such software will be described as implementation of a specific discipline. Not exclusive examples for different implementations of some disciplines are e.g.

- performance: GTlab-Performance (in-house code DLR, conditionally licensable), Mars (in-house code Rolls-Royce Deutschland),
- network: Flowmaster V7, Flownex SE (both commercial),
- CFD: Ansys CFX (commercial), Numeca (commercial), OpenFOAM (open source).

These examples show some fundamental challenges. One is the availability of the implementation. Free and commercial software is basically available everywhere. In contrast, in-house codes of gas turbine OEMs (original equipment manufacturer) are barely available at other institutions or their use is restricted to certain projects. In other words: The specification of an entire discipline to one implementation only tends to be insufficient, since future work could require the use of another implementation.
Another challenge is the coupling of the implementation to the core process. Open source software allows the direct integration of an interface on side of the implementation, realized via libraries. Commercial software optionally provides an API (application programming interface). If not, it is at least required to provide possibilities for batch processing, which is most popular realized by pre-processing, running and post-processing the software via command-line interpreters.
Finally, implementations may differ in the nature and scope of their modeling and customization for specific applications.
In consequence, the interfaces of the core process to on discipline must also be generic to support different, sometimes unknown implementations.

3.4 Integration of Implementations and their Computational Performance

All implementations need an interface in order to be coupled to the core process. This may also be a specific interface, which communicates through one of the previously introduced discipline interfaces. Talking about specific interfaces: The introduced ways for integration – either via library or bash process – have different benefits and disadvantages also worth to be considered.
A bash process is commonly the easiest and fastest way to address a software as long as it offers file input reading and file output generation. The framework would then require the implementation of a specific interface, which exports new input to the appropriate format and read generated output to be converted for the core process. Common disadvantages are the relatively computational time consuming input/output writing and reading (file I/O) as well as the continuous reinitialization of the software in iterative processes. Furthermore, not every software supports bash processing in an effective way. An example is software, which is optimized to operate on a database service instead of fie I/O.
The usually better choice is the direct coupling by loading the implementation as library to the framework at runtime. This way also requires the implementation of a specific interface, which initializes the model once and manages the data exchange with the core process via objects. In this scenario, file I/O is usually not required.
A special case is embedding the implementation as module directly into the framework, which is only possible, if the source code of the framework is available.
The modular plugging of any implementation's library may be challenging in following terms:

- The provided API or open source implementation is written in another programming language: This can be solved by intermediate wrapper libraries with the purpose to transform the data types between the languages.

- The implementation runs on another operating system (OS) or in another process architecture (e.g. 32- vs. 64-bit model): This requires inter-process communication, which can also be realized by wrapper libraries. These are acting as client application on the framework side and host application on the implementation side. In fact, host-client services can solve almost every problem regarding the communication between processes, but are relatively complex solutions.

3.5 Performance of Core Process

Codes generated in academic research projects tend not to be effectively written from the point of view of computer science. It is important to note that this is an acceptable matter, since research projects are time limited and must focus on the scientific contents. Nevertheless, both implementations and framework itself should be designed to run studies in a reasonable frame of time. When dealing with component zooming, the question arises, if the framework must be considered as critical factor regarding computational time.
In fact, this matter highly depends on the kind of workflow and even more important the implementations used within the workflow. Some disciplines in general tend to be consuming much computational time, e.g. 2D/3D CFD. In contrast, gas turbine performance is originally designed to run fast on a specific set of boundary conditions. Thus the order of the model is the first indicator to define bottlenecks in the workflow. Besides there are other characteristics indicating bottlenecks, e.g. an implementation's internal solver performance and the already introduced data exchange with the core process.
Overall, typical implementations should be rather considered as issues regarding computational performance than the framework itself. Beyond that it is more beneficial to identify the actual bottlenecks and put effort into their performance enhancements, if possible.

3.6 Portability

Another important issue is the likely portability of the framework to other operating systems. This might be required because of following reasons:

- Different institutions or companies willing to work with the framework might have different OS defined as standard.
- The same might apply to different departments within one institution or the freedom of employees to choose the OS.
- Some machines in one institution might work on different OS. A prominent example is Windows as OS of employees' desktop computers and a computer cluster operating on a Unix derivate.
- Available implementations might be compiled for certain OS or process architectures. A recompilation is not possible at the institutions site. This also frequently applies for commercial software.

This can be solved by implementing the framework itself in a way that allows the portability to other OS. The choice of programming language is important in this matter. Unfortunately, it is not possible to resolve this problem completely. Possible restrictions can be resolved by the use of virtual machines or once again host-client services.
It is obligatory that the topic of portability doesn't solely apply to the development of the framework, but also to the development of any associated implementation.

3.7 Parallelization, Cluster Operation and Remote Job Distribution

Broad concept studies can require a lot of computational resources. Where already standalone gas turbine performance studies may be a candidate for operation on computer clusters, workflows including models of higher order with the need of inner matching procedures are consuming even more computational time.
A popular used approach enhancing the performance of used computational resources is parallelization, realized by multi-threading: Parts of the workflow or the simulation in general are split into independent problems in order to assign them to individual central processing unit (CPU) cores. Branching the workflow and reasonable merging of threads at their end is a complex affair that can, in the worst case, increase the cumulated CPU time.
Looking at the exemplary workflows presented in sec. 2.2, there is only little potential for branching them. Best options are provided in iterative processes. If such loops are controlled by optimization algorithms which include independent function evaluations, multi-threading is an option. This is e.g. the case by application of evolutionary algorithms, in which broad populations are evaluated and results are merged in the creation of the next generation. Another example are gradient algorithms when evaluating the Jacobi matrix. Another option would be the parallelization of different OPs. However, this is not applicable in the workflows presented in sec. 2.2.3-2.2.4, since the results of OPs at the beginning of the flight mission depend on results in later OPs: The amount of fuel mass required in later mission segments affects the thrust demand of the current OP, but not vice versa.
Overall, typical workflows for component zooming are more or less sequential with only embedded loops, which are typically sequential themselves. For this reason, a parallelization is only a minor business case when designing the framework. In contrast, some implementations may benefit from parallelization. Best examples are commercial FEM and CFD codes, which are commonly computed for effective multi-threading.
If a workflow supports multi-threading, it tends to be running effectively on a computer cluster, too. But even if parallelization is not considered, cluster operation might be an issue regarding the parallel, independent simulation of workflows on either different OPs such as suggested in sec. 2.2.2 or on any concept study dealing with different, independent geometries such as applying for both the flexible SAS and aircraft mission vs. engine mass assessments. However, this type of parallelization doesn't need multi-threading, but just running multiple instances of the framework. Then, the requirements of the framework would be limited to the issues discussed in sec. 3.4 and 3.6.
Even if there might be justification to run workflows on a computer cluster, it might not always be a reasonable solution. There might be conflicts with departments running software, which is highly optimized for cluster operation, e.g. again CFD. This dispute can be avoided by a relatively simple alternative: remote job distribution, which became especially popular with the SETI@home project with the publication of the software BOINC [16].
The implementation of a remote job capability is relatively simple, if it is aimed to be realized in a trustworthy environment, namely the institution's local area network (LAN). Everything required is a thin client-host service, which allows for the submission of rudimental prompts such as load, run or stop a simulation, and another

service, which allows the update of new workflow input files and the collection of results. Assuming the framework and all required implementations to be installed on a remote client desktop computer and ramping up after system start, it can be an effective solution regarding computational resources. E.g. the evaluation of computational resources at desktop computers at the Chair for Aero Engines, TU Berlin showed high shares of operation at CPU loads below ($1/n_{CPU}$), with n_{CPU} defining the number of cores located on the main processor. A significant deviation has arisen only for computers frequently used for simulations with 3D CFD models. However, remote job distribution should only be seen as feature, but never as requirement for a new framework.

3.8 Framework Integration to Parental Processes

The very first question discussed the integration level with some good arguments to design the framework as independent application with all required simulations plugged in. If choosing this way, every end-user should understand the benefits of this approach and hence respect that decision. In other words: It should never be the intent to plug the framework itself as external service to another simulation.
However, there might still be one reasonable situation providing an interface to plug the framework to an external process: Making two different frameworks interact with each other. For this purpose, it is advisable to provide at least one way to run the framework as batch process.

3.9 A Word About GUI Programming

Graphical user interfaces (GUI) can support the engineer e.g. by setting up workflows, checking simulation outputs at runtime and review errors. Especially for users who are not familiar with a software, they show advantages. On the other hand, daily users sometimes prefer the manual configuration of projects. A good example of this are common performance codes, which tend to offer only very rudimentary GUIs.
Since GUI programming can also be time-consuming, because usability requires high robustness, it should be prioritized rather low for such a framework. However, the integration of GUI features should be discussed from time to time as the complexity of a program grows. This might especially apply for the academic area, when the framework is considered to be used in teaching or scientific staff is frequently changing.

4. IMPLEMENTATION OF IPSM

This section will shortly describe, how the before formulated questions have been answered by implementing the framework IPSM, which has been mostly developed in the scope of the framework AG Turbo and which was introductory presented in [13]. It is also intended to give suggestions for the application of techniques, which are more or less available platform and programming language independent.

4.1 Choice of Programming Language

The first decision has been the choice of an object oriented programming language because of their universally known benefits, most of all class inheritance facilitating the recycling of code and support later extensions. Further, a relatively modern and common language should be chosen such as C++, C# or Java.
Regarding platform portability, C++ and Java are originally suited with support of different, common OS. In contrast C# as .NET programming language was primarily designed for Microsoft Windows OS, even if there are different projects aiming for a more general support on other platforms.
Finally, Java was chosen as major programming language for IPSM. However, C++ is also evident for a proper framework implementation and may generally provide a higher flexibility to enhance the computational performance of the framework. But as stated in sec. 3.5, the framework is not considered to be a bottleneck. In the end, this paper won't discuss advantages and disadvantages of these two programming languages – both are suitable. At least it should be clarified that such languages for the here presented purpose are preferable to implementations in e.g. VBA, see also [17].

4.2 Core Process and Parameter Exchange

The fundamentals of the core process design have been published in [13], but will be covered due to extensions in the meantime. The framework's entire architecture is organized in a modular way, where the core process itself, the individual generic interfaces of each discipline and different auxiliary tools are stored in various Java libraries. These libraries are connected with each other as plugins, forming the entire framework application. This is depicted in Figure 7 with gas turbine performance, network and an arbitrary additional discipline, specified by each one implementation. The direction of edge arrows defines the dependency of plugins. For example, all modules depend on the auxiliary modules and the specific implementation interfaces always inherit certain classes from its associated generic discipline interfaces. The latter are registered at the core process, so that it can communicate with these interfaces. Indeed, the core process doesn't have to know about the specific implementations behind the generic discipline interfaces: These are resolved at runtime by using a technique in Java called instantiation via so-called META-INF services. In this way, it is also possible to provide the framework to a third party 'as is', still allowing the third party to attach own implementations to the framework without the need of recompiling it. Comparing Figure 7 with the original from [13], the changed hierarchy of dependencies becomes clear – the former structure didn't allow the independent integration of third party implementations.

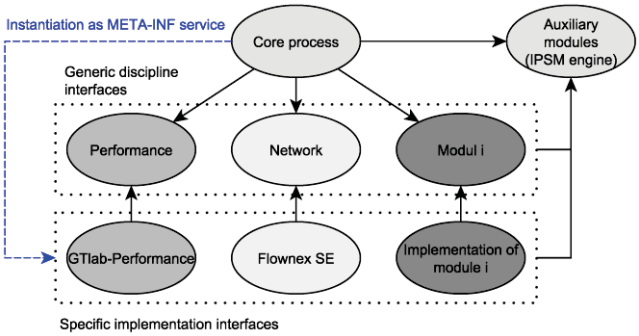

Figure 7 Modular architecture of IPSM, extended from [13]

Figure 8 demonstrates the basic implementation of workflows, which are combined of following key features:

- Modules: Each module is the computational representation of usually one model. The kind of model is arbitrary, both regarding its discipline as well as its complexity. A module can basically represent the 3D CFD model of an entire component or just a scaling equation of a certain, single parameter.
- Process chains: Common modules are arranged in their sequential order in process chains. Process chains itself can be handled as common modules, too.
- Optimization modules: Suited with the same base properties as common modules, they are primarily representing optimization as discipline. A special feature is that the optimization is performed on a problem, which must be represented by a process chain. The simulation run of an optimization module hence opens an embedded loop, which is iterated until user-defined break criteria are reached.
- Operating points: In the scope of the workflow, OP means the run of the entire workflow or a part of it under a certain set of input parameters. These might be associated to e.g. different boundary conditions, but may in principle also represent changed geometries or even solver settings. Operating points may be placed on the very top level – hence repeating the entire project several times – or on the level of any process chain. The latter allows for multi OP studies as e.g. required by the workflows presented in sec. 2.2.3-2.2.4.
- Constraints: In the context of the core process, the term constraint refers to a parameter exchanged between modules. The evaluation of each module is managed by the core process. This includes one initialization while loading the workflow and three typical steps at runtime. The first is preprocessing by means of transferring new input values registered at a central constraint map to the module. This is followed by the simulation run at current inputs. Result parameters registered in a workflow's definition file are extracted and stored to the central constraint map, where they are available for the preprocessing of other modules. The definition of constraints is not bound to any workflow level so that modules from embedded modules may interchange parameters with modules in other process chains and even with OP definitions.

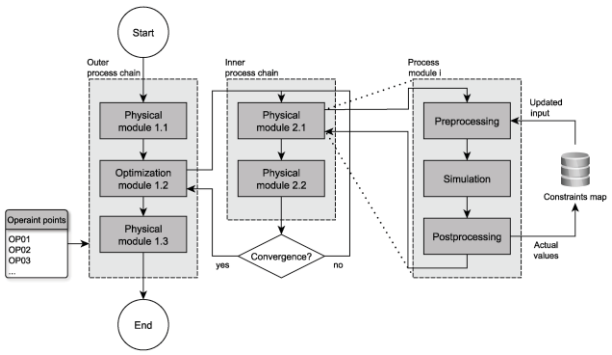

Figure 8 Principle of an IPSM workflow with all available key features, extended from [13]

The implementation of these five features is sufficient to support nearly arbitrary workflows. Even the here implemented feature of running IPSM as slave in external workflows (see sec. 3.8), not to be emphasized here in detail, integrates the external process as common workflow module.

The workflow itself is defined in a small set of XML-files. XML (extensible markup language) is a file standard, organizing data in tree structures. All IPSM input and most output files are using this format, which facilitates usability and is relatively robust at parsing. Workflow input files are

- one project file defining the entry point and top level OPs,
- each one process chain definition file including the specifically included modules in their correct order and
- one overall file with the definition of all constraints.

Extension of available workflows is straight forward, since additional modules may be inserted to process chain definitions by adding XML-elements declaring the new module's discipline, implementation and fundamental information such as the location of required input or generated output files. In this way, it is also easy to switch between different implementations by just updating these entries.

Considering the constraint definition, the most important entries are, which unique module is acting as provider of the constraint's value and which are the modules receiving updated values as new inputs. Since both inputs and outputs as well as their transfer through the specific implementation interfaces may be arranged in total different ways, the parameters to be addressed as inputs/outputs must be declared with a so-called link. This link is effectively a string, which must be parsed in the specific implementation interfaces, usually including a trace e.g. to the targeted component, part and finally the parameter description on side of the implementation. When switching the implementation, these links must be adjusted as well. This might be overcome in future versions by naming conventions for all implementations of a discipline. This is in fact an optional requirement, which solutions like GTlab already support due to their strict necessity in collaboration of different departments.

4.3 Interface Example: The Generic Discipline

The direct communication between the core process and all generic discipline interfaces is hard integrated in the core process plugin and is effectively not of interest for third party developers, who want to extend the framework with a new implementation. Coding the specific implementation interface requires a dependency setting of the most appropriate discipline interface or in worst case the generic discipline's interface. The latter is described in this subsection referring to the most important, not exclusive methods to be implemented by the developer. This gives a good inside of what to be considered when creating a similar framework on its own. The prefix of each method declares its return type. Boolean return values always indicate the success of the method's intent.

- *boolean setSetup(IDiscSettings)*
  This method hands over a prepared, standardized object from an IPSM specific type called "IDiscSettings" including information about the represented model's configuration, e.g. location of input and output files. All of this is information previously read from the associated process chain definition file.

- *boolean initInstance()*
  This performs the rudimental initialization of the module, which might contain loading a dynamic link library or starting the application.
- *boolean setProject()*
  Initializes the "real" model on side of the implementation, e.g. loading a network model defined by contents of the configuration object from the data base.
- *boolean editProject(ITreeStructure)*
  Represents the preprocessing method of the interface according to the depictions in Figure 8. The object from type "ITreeStructure" is a standardized object, which represents an XML-tree – here containing the links to the input parameters and the new value of each. Documented classes implementing the Java interface "ITreeStructure" are accessible for the developer and contained in the auxiliary plugins. It is also the type of data structure evaluated and generated by the core process.
- *boolean runSimulation()*
  Performs the simulation on the preset model under the current input conditions.
- *int getNSI()*
  Returns the so-called numerical status indicator (NSI) of the last simulation run, which is a four-digit number. It represents a return code, which can be used to rate the overall validity of results and partly trace errors. The specific implementation interface or the implementation itself should define those return values in accordance to the standards defined in [18].
- *ITreeStructure getResultParse()*
  Returns – depending on the implementation – all or partial results. Hence that method is responsible for the module's postprocessing. The specific implementation interface must transfer the results to the standardized XML-tree structure from type "ITreeStructure". Contained are the available output parameters and its values, where the XML-tree reflects the links to be used when applying those parameters as constraint value providers.

These are the methods regarded to be required for any implementation working on any framework. Additional operations are usually very specific and hence can be ignited within one of these methods. In IPSM, technical spoken, the developer implements a Java interface, effectively the pendant to the implementation of virtual methods in C++.

4.4 Remote Job Distribution and Control

The remote job distribution and control is realized with the provision of two network clients. One of them is a simple IRC client (internet relay chat), which receives messages from a channel logged to. These messages are parsed by an interpreter for certain keywords and forwarding identified commands by calling certain methods of the core process. This includes e.g. loading, running or stopping of workflows or sending feedbacks on specific workflow or module states. The major advantages of using the IRC protocol are that the protocol is compact, the host doesn't need to be implemented itself, since various hosts are yet existing, and even the remote control from any arbitrary platform is possible, which provides an IRC client software, e.g. smartphones, too. In order to avoid unwanted operation of the remote control, different authorization requests are integrated in this IRC client.
The data exchange for the configuration of remote jobs as well as the collection of results is realized via a purposely independent, separate client. This client is bound to LAN operation and equipped with a relatively strict security concept, which e.g. only allows communication between fixed defined network addresses.

5. CONCEPT STUDIES

5.1 Coupling Secondary Air System Modeling with Engine Performance

The workflows and some results for flexible SAS approaches were presented in [5] (aero engine) and [10] (stationary gas turbine). New results from ongoing concept studies are not in the focus of this subsection. Although, both cases are suitable for demonstrating the advantages of the flexibly designed framework.

5.1.1 Switching of Implementations

In case of the aero engine application, the originally applied implementation of the discipline performance has been GTlab-Performance. In a later phase, this tool has been exchanged by a customized library of the industrial partner's in-house performance code, including the reference engine's original model as hard coded version. Performance code input and output parameters are relatively well standardized. For this purpose, the generic discipline interface of performance has been extended with constraint link interpreters, which allow the use of standardized links to be translated in each implementation interface. In this way, switching between performance implementations almost only requires to change few entries of the module definition within the associated process chain definition file.
Another example is the network discipline, originally implemented with Flowmaster V7, version 7.9.3 (software A). The SAS network model is subject to problems regarding solver robustness, both in the coupled model of the aero engine and the stationary gas turbine. In the latter case, there is an appropriate network model existing for Flownex SE, version 8.9 (software B), which showed comparatively better robustness. In consequence, the implementation was switched for the studies presented in [10]. This required in addition to the module definition some manual changes of constraint links. A small set of constraints had to be redefined because of minor general differences in the network model setups. The latter can't be finally avoided, but are low, the more similar the model setup of two implementations is. Most important: All changes had to be done in workflow definition files presented in sec. 4.2, hence without the need of any code changes. However, it is trivial that implementation switching always requires the specific implementation interface to be coded or provided once before.
Another advantage of adapting the workflow of the stationary gas turbine model to a more robust overall version allows for additional enhancements. The two different types of iterative processes – simple iterator and optimization algorithm – have been introduced in sec. 2.2.1. Previously, the robustness problems required the application of an optimization algorithm. Expecting the alternative SAS network model in software B to run robust, the simple iterator has been selected for subsequent studies. In IPSM, the simple iterator is technically an optimization algorithm, too. The difference

is that it doesn't provide guess values on its own, but can simply forward the new derived secondary mass flows as new input to the performance model. This approach is valid for that type of workflow. This basically self-converging process is beneficial in terms of computational time – or being more precise: less iterative steps.
This exercise also provides the following reverse conclusion: The applicability of a theoretically self-converging iterative process should always demand for the possibility, to be controlled by an optimization algorithm as well. This must be considered both when setting up the workflow and deciding for or programming a framework. Models may run robust in a certain, well known scope, e.g. at design point conditions, but are sometimes not known to run problem free in other OPs are with certain geometry changes.
Figure 9 highlights the changes made in the workflow for flexible SAS studies in the stationary gas turbine presented in [10]. Also included is a plot demonstrating, how the overall simulation time can be accelerated. It includes the optimization algorithm's and simple iterator's convergence progress in the objective function value fcn over function evaluations, hence iterations.

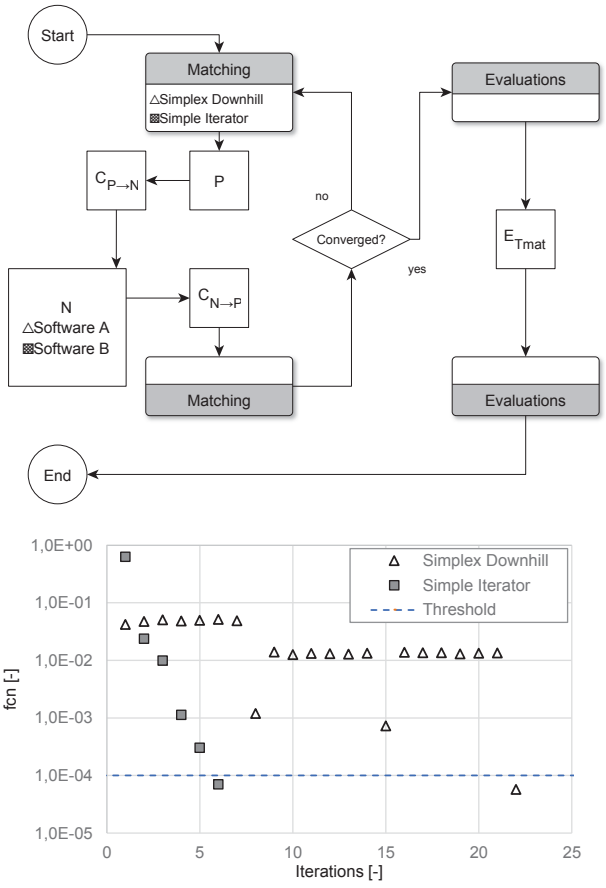

Figure 9 Workflow modifications for flexible SAS in stationary gas turbine and benefit regarding computational time

The recorded progress of fcn represents the matching process of an exemplary OP. The threshold marks the tolerance level fcn has to fall below to rate a solution as matched. In case of the optimization algorithm, the simplex downhill algorithm presented by Nelder and Mead [19] is applied, which has been implemented in IPSM including a rudimentary handling of side condition violations. Here, the latter includes overall simulation robustness control by evaluation of the iterated process chain's overall NSI. It has to be denoted that the presented plot cannot provide a final statement on the benefits of the simple iterator. Each optimization algorithm has another performance for a given optimization problem. Besides, even for the switched network model applies: No model can be guaranteed to run robust under all reasonable conditions.

5.1.2 The Advantage of Zooming to Components

In [5], trades between blade creep life t_{CL} and fuel flow w_f are presented for a hypothetical SAS modulation in an aero engine's high pressure turbine (HPT). Most of the benchmarks are based on a shortened workflow, which doesn't consider a detailed SAS network model. Figure 10 shows the response of a full coupled model including the network model. Throttling the major supply flow of the inner SAS, which is injected through a pre-swirl nozzle system (PSN), results in both a reduction of the fuel flow (benefit) and creep life (penalty). For example, a fuel flow reduction of $\Delta w_{f,rel}$ = -0.13 % is predicted by the full coupled model to be reached at approximately 95 % supply flow throttling, which corresponds to a blade cooling flow of 93.5 % referred to the base setting, see Figure 10-11. However, the drop in creep life to approximately 66 % of the reference is remarkable.

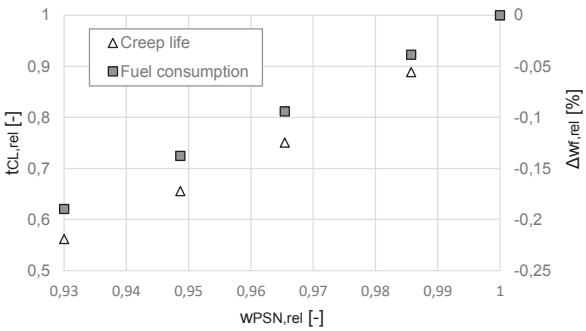

Figure 10 Cruise blade creep life and fuel flow as function of major supply flow

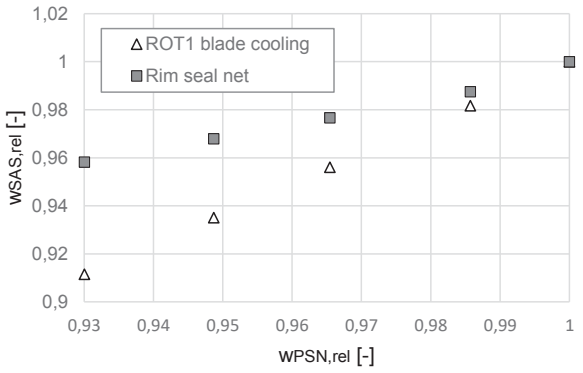

Figure 11 Cruise cooling and rim seal flow distribution as function of major supply flow

Also confirmed is the fact that sink flows facing lower annulus pressures are less affected by throttling of the main supply. This is illustrated in Figure 11 with the comparison of the rim seal net flow upstream of the first rotor blade at a lower sink pressure than the rotor blade's film cooling. Since hot gas ingestion is an additional SAS requirement, the here presented concept must consider

changes in the rim seal flows, too. For this reason, component zooming is a proper approach for holistic investigations on flexible SAS.

5.2 Aircraft Mission at Changed Engine Mass

In the studies of this subsection, the previously introduced geometry models are applied. Their inputs are based on published documents and illustrations and are therefore subject to inaccuracies. Similar applies for the aircraft model. Since this paper deals less with validation aspects of single models than with the possible applications of the framework, the presented results are preliminary benchmarks.

5.2.1 Application of Tandem Blades in HPC

The investigations on application of tandem blades cover three cases, which are compared to a setting with conventional vane aerofoils:

a) Replacement of conventional vanes with tandem configuration in all stages of the HPC
b) Like a), but reducing the number of vanes in each stage to 75 %, motivated by the prediction of enhanced flow control
c) Like a), but removing the second to last HPC stage, motivated by the prediction of enhanced specific work in all other stages. The last stage is not considered for removal, because it is structurally integrated to the interface with the combustor.

The resulting mass reduction per engine as well as the overall benefit in terms of aircraft mission fuel burn reduction at two engine operation is depicted in Figure 12. The diagram contains samples for predictions of general engine mass reductions simulated with the workflow from Figure 5. These samples can be transformed to an exchange rate between mission fuel burn and aircraft take-off weight (TOW), which results in $(\Delta m_{f,mission}/\Delta m_{TOW}) = 0.785$. This result generally corresponds to the order of magnitude of exchange rates presented in [20], which apply to an aircraft in the same competition segment. However, those exchange rates depend on the mission range. A correction to the mission applied in [20] indicates that the here presented fuel burn benefit is likely underestimated, see Table 2.

A fundamental result is that tandem blades tend not to increase the engine mass. Indeed, there is a direct mass reduction, which is caused by slightly shorter axial chord length of the here applied tandem aerofoils compared to the conventional reference design. Applying this technology to the purpose of enhanced flow stability only would come along with a minor positive effect regarding fuel flow consumption, too. However, this effect is negligible, since fuel burn savings are in the order of 1 kg per referred flight mission.

Interestingly, similar applies for case b) which reduces the number of aerofoils. The overall aerofoil mass accounts only minor to the overall HPC mass, which is mainly driven by disks, casing and blade/vane platforms.

More promising is the concept of increased specific work, possibly allowing for the reduction of at least one stage. In contrast to the previous case, this one affects the shortening of radial adjacent components. It has to be noted that these investigations, besides the shortening of the HPC itself, only consider additional shortening of the LP shaft. The associated bypass duct as well as the entire nacelle are not modeled, but should be considered for a more complete assessment.

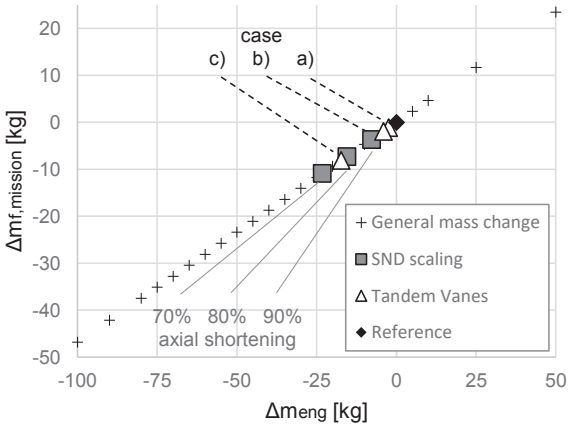

Figure 12 Concepts mass reductions per engine to mission fuel burn

5.2.2 Shortened SND

The results for the axial downscaling of the SND are attached to Figure 12, too. In this scenario, following parts and segments of adjacent components are considered: The SND inner and outer wall, SND strut hollow aerofoils and each one segment of the LP shaft and bypass duct chosen to have the same axial length at base setting. Once again, the nacelle is not modeled. Frame structures and pipes integrated to the SND struts are unchanged, because they are designed to requirements unrelated to the aerodynamics of the SND.

Similar to the introduction of tandem blades, the effect of mass reduction regarding fuel burn savings is limited for SND shortening. It has to be noted that the results don't yet include the masses of possibly required AFC actuation.

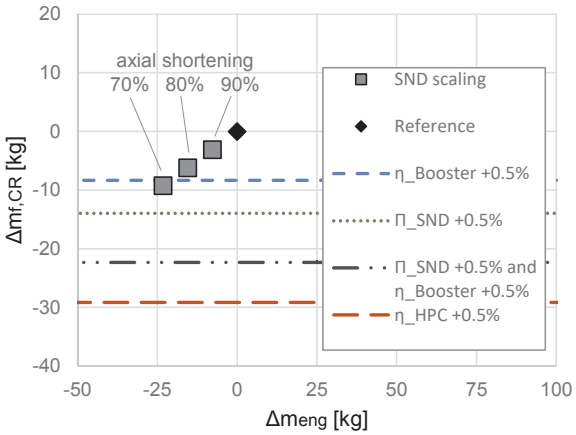

Figure 13 Exemplary SND mass reduction vs. performance enhancements to cruise fuel burn

As mentioned in sec. 2.1.3, SND shortening might be an option regarding mass reduction, but there are other interesting aspects of this technology as well. For this purpose, Figure 13 contains additional benchmarks, which are aiming for enhancements in adjacent component performance. These examples are unrelated to mass changes. The first line marks the level of fuel burn reduction at cruise that can be expected, if the booster efficiency can be increased by 0.5 %. The second aims for a reduction in the pressure loss of the SND, here expressed to an increase of 0.5 % of the total pressure

ratio between SND inlet and outlet. The third line combines both measures. It has to be denoted that these benchmarks are just exemplary on the one side and bound to the applied gas turbine performance model on the other side.

5.2.3 Resulting Options

One conclusion from the presented investigations is that fuel burn benefits are limited when dealing with concepts for engine mass reduction on component level. Other effects of novel concepts should be rather put to the foreground, e.g. the enhancements in flow control, turbo component efficiency or duct pressure losses. Furthermore, today's gas turbine development requires various novel concepts for significant enhancements of the overall performance [21]. For this reason, relatively small benefits in certain fields of the research should not be underestimated. The concept of SND shortening is also interesting in the scope of future engines, e.g. ultra high bypass ratio engines.
One aspect which is likely coming too short in the academic research is the view of the gas turbine or airline operator. Novel concepts providing fuel burn reductions as presented in Figure 12 and Figure 13 are often introduced to real engines as features without an overall gas turbine or aircraft redesign. Assuming an aero engine allowing for a mass reduction per engine in the order of the tandem blade case c) (removal of one HPC stage): In place of reducing the mission fuel burn, the operator may reoffer the mass as additional payload. This allows e.g. for two items of extra luggage per flight cycle, which can be easily translated to an economical benefit.
Remembering the constraint of matching the flight mission trajectory, not all feasibilities resulting from mass reductions can't be discussed here. When the thrust is not rematched to those OPs, which are subject to aircraft acceleration – primarily take-off, climb and descent – the trajectory is adjusted to the changed overall aircraft weight. Cruise altitude can then be reached earlier as well as the initialization of descent. The aspect of acceleration may also be from higher significance in military applications. Last but not least, mass reduction may be converted to mission range extension, too.

6. CONCLUSIONS

Component zooming is a technique that allows for investigations of novel concepts thereby exceeding the boundaries of associated components and technical disciplines. Coupling detailed numerical models with gas turbine performance can assess the benefits of a concept in the context of the surrounding system. One associated challenge is the appropriate definition of model interfaces. Even more challenging might be the establishment of such assessments in a research environment. This paper makes proposals to address this problem.
The early application of component zooming can help to identify the most promising objectives in research on concepts based on the expected benefits. Regarding the results of tandem blade and SND studies presented here, the future focus of research should be put on improvements in aerodynamics rather than on the weight aspect. Nevertheless, it must be emphasized once again that the presented results do not claim to make any final statements about this. The here presented results should be interpreted as conservative benchmarks, strictly considering the assumptions made within the geometry and flight mission model. Follow up studies require the replacement of the current geometry and aircraft models with more accurate models.
In fact, the introduced generic workflows are starting points of more comprehensive investigations. Especially the mass reduction examples allow for the extension with models regarding aerodynamics within the focused components. One of the key disciplines identified by the authors are turbomachinery meanline models, which may serve both for the better modeling of aerodynamics and as interface models, e.g. between gas turbine performance and network simulation. The extension of the here discussed workflows with such models shall be conducted in subsequent works.
All studies presented in this paper focus on the gas turbine. Nevertheless, the workflow for mass benefit assessment may also serve as starting point for deeper integrative studies including both aircraft and engine models. This may also include additional system interfaces such as cabin air supply from the SAS.
All this proves that the sustainable development of a framework for component zooming is worthwhile. It can be applied to nearly arbitrary problems, allows for future extension of available workflows and also facilitates the before mentioned replacement of single models. Besides of that component zooming may promote the inter-disciplinary exchange on institutional level, which is beneficial regarding the overall competence.

7. ACKNOWLEDGEMENTS

The investigations were essentially conducted as part of the joint research programs COORETEC-turbo (AG Turbo 2020) and ECOFlex-turbo in the frame of AG Turbo. The work was supported by the Bundesministerium für Wirtschaft und Energie (BMWi) as per resolution of the German Federal Parliament under grant numbers 03ET2013P and 03ET7091I. The authors gratefully acknowledge AG Turbo and Rolls-Royce Deutschland Ltd & Co KG for their support and permission to publish this paper. The responsibility for the content lies solely with its authors.
Additional thanks go to the student researchers at Chair for Aero Engines who were involved in the implementation of IPSM and test case creation, namely in the order of their employment: Silvio Chemnitz, Jonas König, André Resag, Phong Tran, Jiri Dehmel and Bojan Pijanovic.
The authors thank Jan Mihalyovics and Liesbeth Konrath for provision of conceptual input to the presented studies on SND and tandem blades. Martin Bolemant, Tim Sauer and Nicolai Neumann are valued for manifold general discussions about test cases and study results.
Additionally, we thank the German Aerospace Centre as chair's partner providing the performance code GTlab-Performance.

8. LITERATURE

[1] R. W. Claus et al.: ‚Numerical propulsion system simulation', Computing Systems in Engineering, Cleveland, 2 (4), 1991.

[2] P. Jeschke et al.: ‚Preliminary Gas Turbine Design Using the Multidisciplinary Design System MOPEDS', Proceedings of ASME Turbo Expo 2002, GT-2002-30496, Amsterdam, 2004.

[3] S. Reitenbach et al.: ‚Design and Application of a Multidisciplinary Predesign Process for Novel Engine Concepts', Journal of Engineering for Gas Turbines and Power, 141 (1), 011017, 2019.

[4] M. Swoboda: ‚Abschlussbericht LuFo 3: VIT (virtuelles Triebwerk)', final report to project Nr. 8.1 NKBF 98, FKZ 20T0307B, 2007. doi: 10.2314/GBV:557848172.

[5] D. Woelki, D. Peitsch: ‚Modeling and Potentials of Flexible Secondary Air Systems Regarding Mission Fuel Burn Reduction and Blade Creep Life', Proceedings of ISABE 2019, ISABE-2019-24435, Canberra, 2019.

[6] F. Donus et al.: ‚Accuracy of Analytical Engine Weight Estimation During the Conceptual Design Phase', Proceedings of ASME Turbo Expo 2010, GT2010-23774, Glasgow, 2010.

[7] A. Heinrich, D. Peitsch: ‚Recent Insights Into the Flow Topology Around Highly Loaded Tandem Vanes', Proceedings of GPPS 2019, GPPS-BJ-2019-225, Beijing, 2019.

[8] T. Stürzebecher et al.: ‚Automated Aerodynamic Optimization of an Aggressive S-Shaped Intermediate Compressor Duct', Proceedings of ASME Turbo Expo 2018, GT2018-75184, Oslo, 2018.

[9] V. Motta et al.: ‚Numerical Assessment of Virtual Control Surfaces for Load Alleviation on Compressor Blades', MDPI Applied Sciences, 8(1), 125, 2018.

[10] D. Woelki et al.: ‚Simulation on part load controlled cooling air supplied in stationary gas turbines', Proceedings of GPPS 2019, GPPS-BJ-2019-0221, Beijing, 2019.

[11] T. Sauer et al.: ‚Development of a Flight Mission Model and Its Application in Order to Illustrate the Influence of Variable Thrust Nozzles', Proceedings of GPPS 2019, GPPS-BJ-2019-0217, Beijing, 2019.

[12] N. Neumann, D. Peitsch: ‚Introduction and Validation of a Mean Line Solver for Present and Future Turbomachines', Proceedings of ISABE 2019, ISABE-2019-24441, Canberra, 2019.

[13] D. Woelki, D. Peitsch: ‚Modellierung variabler Sekundärluftsysteme zur Bewertung ihrer Auswirkungen auf das Gesamtsystem Gasturbine', Proceedings of DLRK 2014, DLRK 2014-340112, Augsburg, 2014.

[14] P. Zeller: ‚Effizienzsteigerung von Turboluftstrahl-triebwerken durch Optimierung des sekundären Luftsystems', Dissertation, University of Stuttgart, Stuttgart, 1995.

[15] S. Bretschneider et al.: ‚Compressor Casing Preliminary Design Based on Features', Proceedings of ASME Turbo Expo 2008, GT2008-50102, Berlin, 2008.

[16] D. P. Anderson et al.: ‚Designing a Runtime System for Volunteer Computing', SC '06 Proceedings of the 2006 ACM/IEEE conference on Supercomputing, 126, Tampa, 2006.

[17] P. Kupijai et al.: ‚System Integration as Key for Improving and Speeding up the Preliminary Design Phase of Aero Engines', 2013 SIMULIA Community Conference, Vienna, 2013.

[18] SAE Aerospace: ‚Aerospace Standard AS4191: Gas Turbine Engine Performance Presentation for Computer Programs Using FORTRAN', SAE International, 2008.

[19] J. A. Nelder, R. Mead: ‚A Simplex Method for Function Minimization', The Computer Journal 7, Cambridge, 1965.

[20] F. Deidewig et al.: ‚Methods to Assess Aircraft Engine Emissions in Flight', 20th Congress of the Int. Council of the Aeronautical Sciences 1996, ICAS-96-4.1.2, Sorrent, 1996.

[21] National Academies of Sciences, Engineering, and Medicine: ‚Commercial Aircraft Propulsion and Energy Systems Research: Reducing Global Carbon Emissions', The National Academies Press, Washington D.C., 2016.

[22] EUROCONTROL Institute of Air Navigation Services: ‚Aircraft Performance Database A320', web link, URL: https://contentzone.eurocontrol.int/aircraftperformance/details.aspx?ICAO=A320&ICAOFilter=a320 , called: 30th August 2019.

APPENDIX

	OP	Alt * [ft]	v_{TAS} [kt]	Ma [-]	ΔAlt/Δt [ft/min]	s [NM]	t [min]
01	Take-Off	0	145		0	**1.183**	► 0.49
02	Initial Climb	5000	600		**2500**	► 20.0	► 2.0
03	Initial Climb to FL150	15000	290		**2000**	► 24.2	► 5.0
04	Initial Climb to FL240	24000	290		**1400**	► 31.1	► 6.4
05	Climb at Ma constant	39000		0.78	**1000**	► 112	► 15.0
11	Cruise	39000	450		0	**300**	► 40.0
…	*descretized to nine segments*	…	…	…	…	**…**	…
19	*of each same range*	39000	450		0	**300**	► 40.0
21	Initial Descent to FL240	20000		0.78	**-1000**	► 152	► 19.0
22	Descent to FL100	10000	290		**-3500**	► 13.8	► 2.86
23	Approach	0	230		**-1500**	► 25.6	► 6.67
31	Go-Around Take-Off	2800	230		**2500**	► 4.29	► 1.12
32	Go-Around Circle	2800	230		0	► 38.3	**10.0**
33	Go-Around Approach	0	230		**-1500**	► 7.16	► 1.87
34	Landing	0	137		0	**0.778**	► 0.34

bold Calculation mode based on parameter
► Calculated parameter
* Defines the final OP altitude, since initial altitude set to the final altitude of the previous OP

Operating empty weight	42600 kg
Payload	12000 kg
Fuel regular mission	*calculated*
Fuel reserve	500 kg

Table 1 Flight mission definition [22] extended with go-around procedure and aircraft mass assumptions

$(\Delta m_{f,mission}/\Delta m_{TOW})$ [-]	reference aircraft	reference mission	description
0.73	Boeing 737	according to [20], • $s_{mission}$ = 1080 NM	literature based exchange rate
0.785	Airbus A320 alike	according to Table 1, • $s_{mission}$ = 3130 NM	exchange rate for presented model
0.59	Airbus A320 alike	according to Table 1, but • Alt_{OP05} according to Alt_{CR} from [20] • $Alt_{OP11...19}$ according to Alt_{CR} from [20] • $Ma_{OP11...19}$ according to Ma_{CR} from [20] • $s_{mission}$ according to [20] • without Go-Around (OP31-33) • without fuel reserve	exchange rate for presented model, corrected to mission of literature based exchange rate

Table 2 Exchange rates for fuel burn change to TOW change