Proceedings of the 1st Global Power and Propulsion Forum
GPPF 2017
Jan 16-18, 2014, Zurich, Switzerland
www.pps.global

GPPF-2017- 159

DEVELOPMENT OF A SysML FRAMEWORK FOR GAS TURBINE DESIGN UNDER UNCERTAINTY

Arun Ramamurthy
Siemens Energy, Inc.
arun.ramamurthy@siemens.com
Orlando, FL, USA

José Valenzuela del Rio
Siemens AG
jose.valenzuela_del_rio@siemens.com
Mulheim, NRW, Germany

Frédéric Villeneuve
Siemens Energy, Inc.
frederic.villeneuve@siemens.com
Orlando, FL, USA

Jelena Veer
Siemens Energy, Inc.
jelena.veer@siemens.com
Orlando, FL, USA

ABSTRACT

Designing complex systems such as gas turbines is an endeavour generally undertaken by multiple teams within an enterprise. Traditional design methods often fail to capture the complexity of sub-systems interaction, leading to sub-optimal designs. One aspect behind the complexity of the design process is the challenge behind design tools and models management. There is thus a necessity for efficient models management. Another aspect is the ability to elicit and manage uncertainty. Uncertainty plays an important role in the design of any product, even more so in the design of a complex system. Determination of the sources of uncertainties is difficult and often limited to subject matter experts, which become bottlenecks and increase design cycle time.

While a vast majority of the publications on current standards in design approach the problem from the perspective of a single engineer, the process of design requires contributions from several teams of engineers and is one of the key aspects addressed by the paper. In order to capture the collaboration between several engineers, the paper develops an extensible and modular SysML framework that permits collaborative and concurrent engineering. The framework brings forth the use of design history knowledge in the creation of new. The framework also permits the use of the same model by multiple engineers to perform their individual tasks.

The paper demonstrates the application of the framework in the simulation of the workflow involved in the conceptual design of a gas turbine subject to technological uncertainties. An engineer within this framework is able to interactively create, access, revise and publish models, which are in turn shared across the various stages of the design. The role-based delineation across different engineering tasks, permits effective data management and hence faster design loops. The application of the proposed framework is intended reduce the design cycle time of complex systems such as gas turbines.

INTRODUCTION

A complex system is one that involves numerous interacting components displaying non-linear behaviour [1]. Examples of such systems range from one in which several physical components interact with each other such as automobiles, to entities where several different disciplines interact with each other, such as a gas turbine. In such a case where multi-disciplinary design is involved, traditional design methods involve development of mathematical models specific to the discipline [2]. In the example of a gas turbine, the complexity in the model necessitates the decomposition of the entire system into subsystems, and analysis is carried out at these decomposed levels. These analyses typically fail to capture the emergent behaviour seen at the system level leading to sub-optimal designs. An alternative to the subsystem modelling approach is to model the entire gas turbine, with all its complexity, but such a model requires a large amount of computational resources and modelling knowledge that are typically not available to any one branch of the organization. Thus, there is a necessity for a streamlined model and data sharing framework such that models can be accessed with ease across the organization to aid the modelling of a mathematically accurate complex system.

A further hindrance to the development of such complex models is the difficulty in transfer of knowledge across the different stages of design. The design process consists of analyses across several fidelities with designs ranging from the conceptual design identifying and selecting an architecture, to the detailed design, optimizing the down-selected concept, for the system [2]. Often, the traditional design process, assuming that the relations are well defined, falls short in communicating the information between these design stages due to the lack of an efficient medium facilitating the inter-fidelity data transfer. The design process applied to a gas turbine is carried out over several months, if not years, and the data associated with the design has to be maintained throughout this duration in order to facilitate the afore mentioned communication. Further, there is a necessity to maintain the generated data in order leverage the generated design data in a redesign process or in service. The current standards rely on the data transfers occurring in a unidirectional manner with design knowledge [2] being fed forward, but the entire design process has to be restarted if any changes are made to the concept or the architecture due to a non-parametric interface between the multi-fidelity models. Further, upon such changes, several inconsistencies are created in the design that require manual interference by the engineer leading to a significant increase in the design cycle time.

Certain commercial software packages integrate and automate simulation tools generally effectively. Most of them also offer optimization algorithms and visualization capabilities for post-processing. Some of the most popular are ModelCenter, iSIGHT, Heeds, and OptiSlang. These tools are user friendly and cater to the generation of general workflows maintained by one engineer at a time. These solutions are however difficult to scale for large simulations involving several tools and variables, and difficult to keep track of the revision history for large organizations designing complex systems. Also the integration of these commercial software into the complex workflows of large organizations and their existing infrastructure such as collaboration servers represent a challenge.

Simulation data and process management systems also exist on the market. Teamcenter for Simulation, MSC SimManager, ANSYS EKM are some popular ones. The proposed framework within this paper extends these software solutions with different collaboration workflows and simulation process definitions.

In order to address the issues of data management and design interface definitions, the following paper develops a Model Based Systems Engineering (MBSE) backend supported by a System Modelling Language (SysML) [3] framework that permits the use of system engineering fundamentals such as collaborative and concurrent engineering. The framework allows any generated model to be shared between engineers using a centralized database and interfaces are built into the models using standardized semantics rather than being defined after-the-fact.

The paper is organized to first introduce the current standards in the design of gas turbine engines identifying the gaps, followed with an introduction to MBSE. A brief description of the framework architecture and the module involved is then provided followed with an example application to the conceptual design of a gas turbine in which various roles of the different engineers are simulated in the a technology uncertainty study.

SOFTWARE ARCHITECTURE

The framework leverages concepts from several Open-Source Software (OSS) entities developing an effective interface between these tools. Based on a survey of some of the popular programming languages [4], and the requirement of an object-oriented (OO) software, the framework was developed in Python [5]. The developed framework is comprised of four main components, as illustrated in Figure 1:

1. A systems engineering module (SEM) that allows the creation and definition of the analysis models. Further, the interface definition between different models is handled by this module. An enhancement to the traditional engineering models developed via SysML, is the capability to execute the any model by defining its appropriate behaviour. Further the module serves to define the requirements for the created entities within the models. This module is developed in-house.
2. A Design Space Exploration (DSE) environment, which is developed based off of the capabilities available in Dakota [6], SciPy [7], and pyOpt [8] that enables the optimization, uncertainty quantification, reliability and robust design, as well as design of experiments.
3. A data-mining and statistical processing module that allows the processing of results generated from the exploration process and permits the reuse of generated statistical models in future design space explorations. The module is based off-of the open source python package scikit-learn [9].
4. An advanced visualization module (AVM) that allows an engineer to post-process the results of the analysis and perform requirements analysis and verification on the generated results. The open source python packages, matplotlib [10] and PyQtGraph [11] are used to perform the 2D and 3D visualizations respectively while the decision making framework is built in-house.

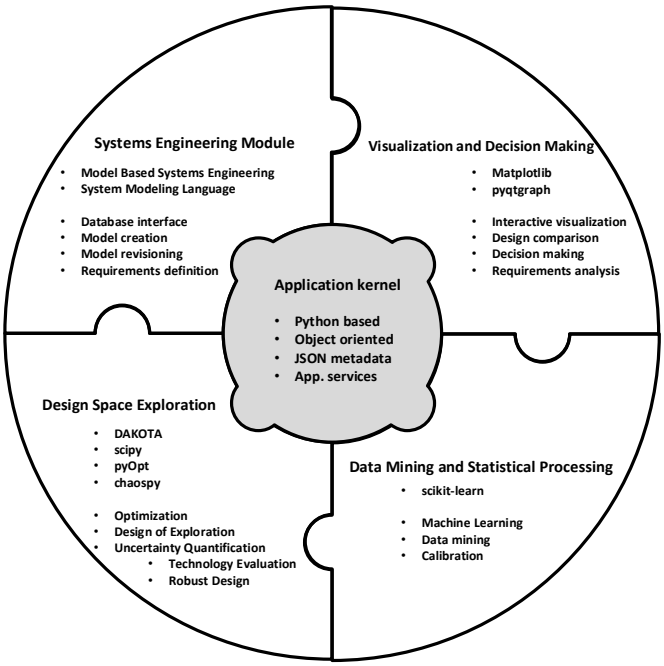

Figure 1: Four component modules of the SysML framework

In order to streamline data management and interaction between different tasks and engineers, the framework was integrated with a central database that serves to store the models developed over the life cycle of the design. An open source version control platform is used to function as both the data management system and database for the framework. The final piece of the framework is the execution manager, built in-house, that controls the execution of the various models developed within the systems engineering module. The execution controller is based on a master-slave architecture [12] in which an "activity" drives the execution of the model. The framework is developed based on the python binding to the C++ Qt library, PyQt [13] in which a thread pool is used to execute a set of runners. In order to ease the access of the execution manager, it was built as a service of the application with the application itself being a singleton. Figure 2 illustrates the architectural structure of the developed software and the communications between the different modules. It is important to note that due to the modular and independent nature of the framework, any one of these entities can easily be swapped for alternative source without reduction in the capabilities of other modules.

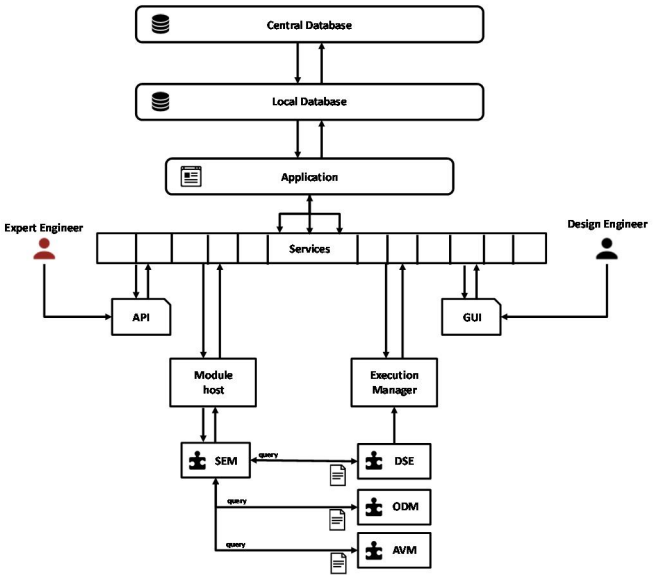

Figure 2: Software architecture

An important aspect that was considered during the development of the framework was the inherent limitation of SysML where in the engineers performing the day-to-day design activities are neither computer engineers nor are they software architects. Thus the authors develop a mask (a graphical user interface) that is to function as the interface between the engineer and framework through which the engineer can interact with an interface (s)he is familiar with while the internal translation to the required models is carried out automatically in the background. In addition, to the interface provided, an engineer more familiar with programming and development process can access the framework via an Application Programming Interface (API) and using the object oriented nature of the framework, can develop new or access previously developed models with very little difficulty. Using the developed interface, a streamlined workflow is generated that minimizes the engineers necessity to get familiar with new concepts and interfaces.

A key aspect of model-base system engineering is the development of reusable models along the different branches of the organization that are involved in the design process. In order to support such a reusability of developed models and to maintain traceability of each engineer's work, each model defined in the framework is represented as a version-controlled repository in the engineer's local database. A local database is created for every user in order to house work that is under development. The engineer can access various historic revisions of the model and, at any point of the design process, move across them to compare or merge the different designs. When an engineer has completed her work with the design they are then published to be used by the rest of the team. This action of publishing a design makes the developed the model visible on the central database and hence accessible to other members of the team. Conditional permissions for visibility are permitted in order to restrict and protect the flow of data outside a particular branch or team of the organization. When a model is published on to

the database, other users can now base their designs off of this state of the model in one of the three following ways:

1. The engineer creates a clone (fork) of the model to make their local changes on the same revision of the design. In such an instance the model is cloned to replicate not only the necessary structure, behaviour, parameters, and requirements, but also the revision history. Upon completion of the design alterations, a request is placed to the original designer to review the suggested changes, based on which, the suggested changes can be accepted, rejected or selectively accepted. Such a situation is useful in cases where the designer is interested in locally experimenting with the design or does not have access to modify the original model, but only read the data. Further, the local experiments conducted by the engineer are visible only on her local database and the requested merge is seen on the central database.
2. The engineer creates a new revision off of the original version but the new revision is a child of the original model. Upon completion of the modifications to the design the child revision can either be published to the database or a request can be submitted to the original design in order to review the proposed changes. If the child design is published to the original tree, not only can the changes can, at any time they be reviewed and accepted, but also, new revisions can be created from these changes. This scenario is useful in the situation where the engineer does not have the access to write to the published design, but has access to write to the database, i.e., create a new design.
3. Finally, the engineer can make modifications to the original design and update the already published design revision. In such a situation, the design revision described in case 1, will not see any updates as the revision is a cloned deviation of the original, while the design revision described in case 2 sees the modifications made to the original as it is a direct child of the original design. In a scenario in which the a conceptual design is modified during the detailed phase of design, for example, to include a new technology, the modifications are seen in the detailed model without minimal interference from the engineer due to the detailed design being derived from the original conceptual model.

The reusability of models promotes a framework for the standardization of processes, which are essential for the development of any complex system where design teams are also often spread across different continents. In the current framework a standard process is created by locking a particular model from future edits, i.e., a protected design.

Further, an important capability brought forth by this approach is the ability to perform collaborative concurrent engineering in which various engineers can work on the same models in the respective design branches and upon completion "merge" their work with the parent revision such that both can be used in future design iterations. A **caveat** to this is the ability to introduce conflicts in the design. Such a scenario arises when multiple design engineers work on the same aspect of the model. In such a scenario, in order to resolve the conflict, the model owner is required to perform a three-may merge [14] based on the local, incoming and outgoing copies of the design.

MODULE DESCRIPTION

The framework is comprised of four main modules. Expanding the work carried out by [15], the different modules interact with each other via metadata description of their associated submodels represented using the JavaScript Object Notation (JSON) format. The submodels associated with each module are also subject to revisions but with the key difference that the revision histories of these submodels are dictated by the parent model. The following passages briefly describe the key model based aspects of each module.

1. Systems Engineering Module (SEM)

The module functions as the interface between the engineer and the database enabling the capability to create, revise, publish and withdraw models. The SEM acts as an interface for the other modules to interface with the database, i.e., upon creation of a submodel in any other discipline, the system model is update with changes to the submodel. Each model is comprised of the four main components of system engineering, a structure, the behaviour, requirements and parametrics. The structure sub-block defines and establishes semantics for the reusability of models and submodels, while building acting as the building block for nested models. This in addition, permits the submodels to interface with other models. The abstract entity representing a structure is termed a <<*block* >> and is imbued with the following properties:

- name:: string
- description:: string
- attribute:: array{Property}
- children:: array{block}
- parent:: block
- uid:: string

The object <<*property*>> further inherits from a block is imbued with the following properties:

- attachments:: array[string]

where, an attachment represents a file/folder that contains data pertaining to the property.

The key building block for each model is a <<*submodel*>> which inherits form the properties block and it is represented by the following attributes:

- variables:: array{Variable}
- behaviour:: array[compiled-code]
- interfaces:: array{Mapping}

Finally, the top level entity is the <<*model*>> which inherits from the block with the following attributes:

- database:: string
- root:: {submodel}

The behaviour aspect of the model defines the capability to represent the physical behaviour of a model. In the current framework the behavioural aspect of the model is represented by a set of compiled code that can be executed upon query. The root submodel for the model is exposed to the

behavioural code such that data stored within the model can be accessed via an object oriented parent-child relationship.

The parametrics represent the relations and equations that relate the various block and parameters within the structure and also functioning as the interfaces between the different blocks which is represented by the "Mapping" block comprised of the attributes:

- equation:: [compiled-code]
- from_uid:: string
- to_uid:: string
- parent:: {submodel}

Finally, the variable block is used to define the parameters that can be exposed to the engineers for each submodel. The properties of the <<*variables*>> which inherits from a <<*block*>> include:

- default:: string | float | int | bool | array
- value:: string | float | int | bool | array
- units:: string
- traits::{RoleTrait | FunctionTrait}
- attributes::{key: value}

The traits associated with the variables distinguish the function and role played by the variable in an analysis. The variables function indicates if the variable is an input, an output, an input-output or a local variable while the role trait indicate the role played by the variable in an analysis such as design parameter, constraints parameter (equality or inequality), uncertain parameter, response parameter etc. The role of the parameter is dependent on the activity it is a part of and also the function it is assigned.

2. Design Space Exploration (DSE)

The design space exploration module functions as the evaluator of the analysis in which various designs are analysed in an automated fashion. The module permits capabilities such as optimization, design of experiments, uncertainty quantification, robust design and technology evaluation. The module interfaces with the submodels developed in the SEM such that evaluation points are generated based on the algorithm defined. The definition of the analysis carried using the defined models the output of which is a revised reusable activity model. With the roles of engineer roles, the standardization of these activity models is supported within the DSE module in which the models are protected upon being published to the central repository based on the engineer's role.

The DSE module interfaces with the execution manager in order to streamline the parallelization and execution of the analyses. The primary driver on which the algorithms in the DSE module are based off of is Sandia National Laboratory's Dakota which is an open source toolkit that providing the necessary capability for general purpose design exploration. Additional algorithms are implemented using open source python packages such as pyOpt, SciPy and chaospy [16]. Each of these packages have the capability to treat the analysis system as a black box enabling an easy interface with the developed models. Due to the singleton nature of the application, the database (both local and central) models are easily accessible enabling control of the execution process.

3. Object oriented data-mining (ODM)

A key extension to the SysML frameworks that is found in literature is the ability to build reusable statistical models based off either analysed or observed data even for design engineers with no statistical background. The data-mining module permits the use of advanced data-mining methods in order to allow the use of machine learning techniques to incorporate models of higher fidelity. The module provides the capability to couple dynamic models that can be coupled with the DSE processes in order to identify and resize the parameter vectors based on past observations. Due to the model history being stored on a database, the learning algorithm is continuously updated with data such that future design attempts are better informed propagating design knowledge through the various designs in an automated fashion.

4. Advanced Visualization Module (AVM)

The visualization module is the key decision making entity of the framework. The module permits the engineer to efficiently arrive at conclusions based off of the generated designs. The ability to compare several designs and store the decision steps in models a key aspect of the module. Capabilities to perform requirements analysis and virtual verification coupled with effective decision making methods can help identify regions of interest in the design space, and decide the next steps based on the information gathered from the visualization steps. The coupling of the module with data mining provides a powerful tool for designers to quickly analyse various combinations of designs in the presence of uncertainty, such as performing sensitivity studies on uncertain parameters or technology studies in which the technology can be turned on-or-off to estimate their impact and interactions, i.e., technology sensitivities.

It is essential to note that the latter two modules are still in their early stages of development, although the capability framework to create models that can be accessed and reused has already been implemented and tested.

APPLICATION

Motivation

In an industry as dynamic and competitive as the power generation investing in the right technologies is of crucial importance. A very important aspect for technology evaluation is the capability to analyse technologies under uncertainty since technologies must be evaluated at different maturity levels. It is of paramount importance to identify the right set of technologies on the correct component and evaluate their integration within a system is essential. Lastly, in order to make the most out of new technologies, especially for gas turbine redesign efforts, designers need to allow for geometrical changes within the system constraints. Given the above reasons, the application of turbine heat transfer and bond coating technologies are chosen to demonstrate the

capabilities of the framework developed over the course of the work.

Modelling and Simulation

The baseline model selected is a large gas turbine similar to a Siemens SGT6-8000H gas turbine working in simple cycle mode. This type of engine will further evolve in the future and benefit from the framework proposed in this paper to correctly simulate further design and technology changes.

The analysis models are composed of technology impact assessment model, a legacy turbine meanline program, and a legacy gas turbine thermodynamic model. The turbine meanline programs contains heat transfer, aerodynamic loss correlations and thermodynamic state equations use to compute the turbine performance. The thermodynamic model computes the overall gas turbine performance using information from the turbine meanline program as well as modelling assumptions from the design engineers and technology forecasters.

Uncertainty models are developed on the pre-selected important uncertain parameters. For instance coating heat conductivity, cooling technology effective thermal efficiency, and aerodynamic loss improvement parameters. Finally, proper variable mapping is performed using the SysML framework.

Uncertainty Quantification

Technology development is normally a time consuming process that involves the identification, evaluation and selection of the right technologies under the simulated technology impact parameters give an investment allocation. In such a scenario, both the technology performance and the allotted investment are subject to uncertainties and hence, risk mitigation and prioritization are essential, in part, due to the current situation of the gas turbine market. There are two main sources of uncertainty:

1. Technologies, in early stage of maturity, can be hard to analyse and evaluate due to a lack of concrete description in their concepts and simulation tools, for example, the emerging technology of additive manufacturing. It is difficult to quantify the value of the technology without accurate physics based models, but the current standards in design do not account for these emerging technologies. Further, tests on such technologies are typically infeasible due to monetary constraints and hence comparisons to established methods are difficult to perform.
2. Another source of uncertainty is the interactions between new technologies and the components of a complex system such gas turbines. While individual component performance can typically be quantified by expensive experiments or simulation, the interactions between technologies and hence their impact on the system are difficult, if not impossible, to quantify without a dedicated technology space exploration.

Technology impact uncertainty was assessed on the turbine meanline input parameters such as coating system material properties, cooling effectiveness and aerodynamic efficiency.

Technology Impact Assessment

Turbine component technologies belong and operate within a very complex system such as the gas turbine. Therefore, they need to be analysed not only within the right physics, but also within the whole gas turbine context. Traditionally, gas turbine companies are organized in component groups that own their own design tools, which make model and tools collaboration more challenging. With the framework developed, communication is established between models to represent the turbine performance tools, which are interfaced with the representative models for the gas turbine performance tool to assess the impact of individual technology on the system performance via standardized interfaces.

When assessing new turbine technologies, gas turbine manufacturers pursue the full technology potential, which requires the ability to upgrade existing gas turbine designs for the given technologies instead of plugging the new technologies on a fresh design. Thus, a framework that permits the capability to trace the changes in the design over the various phases is very important in order to exploit the full potential of new technologies. For gas turbine redesigns, this design framework needs also to account for the constraints of an existing gas turbine frame. In the current example the assessment of new bond coat materials and cooling technologies is more effective if the turbine blades and flow path are allow to changed, however due to blade aero-mechanical constraints the turbine exit cannot be exceed a certain value.

Workflow Simulation

As previously discussed, several different working roles typically exist within the design of a complex system running the gamut from integrated project team leads to design engineers. Employees with different educational and experience backgrounds collaborate together to build the product. This is very evident in the design of a gas turbine in which engineers with skills that range across the multiple disciplines (aerodynamics, heat transfer, structural and system design) are crucial to provide competitive gas turbines to the fierce market.

The paper delineates the tasks of the various groups into six primary categories in order to analyse gas turbines technologies under uncertainty efficiently:

1. The first role is that of a “tool developer” who is responsible for the creation and maintenance of analysis tools which are used to evaluate the performance of different components of the turbine. It is essential to note that several of the tool developers are necessary to capture the multiple disciplines across which the component is to be designed. These engineers can be treated as subject matter experts (SMEs) who are familiar with process of numerical modelling of the physical behaviour of the gas turbine components.

2. The second role is that of an "expert engineer" typically represented by a person with years of experience in designing gas turbine components and can be, for the purposes of the paper, treated as a person who is familiar with the physics driving the behaviour of the component.
3. The third simulated role is that of the integrated project team lead who is entrusted with communicating the management requirements down to the engineers and ensuring the project stays on track.
4. The fourth simulated role is that of a "system designer" who is an expert at the development of processes and interfaces between different tools in order to retain the system perspective while setting up the process of design space explorations.
5. The fifth simulated role is that of the "regular engineer", one who deals with the day-to-day design activities associated with the gas turbine component.
6. The final role is attributed to that of a "statistical engineer" who excels in performing data analysis and decision making in order to narrow down the myriad of designs that are generated during the design space exploration.

It is essential to note that an engineer can embody multiple roles but in order to maintain a clear distinction between the tasks performed, the above roles standard are adopted.

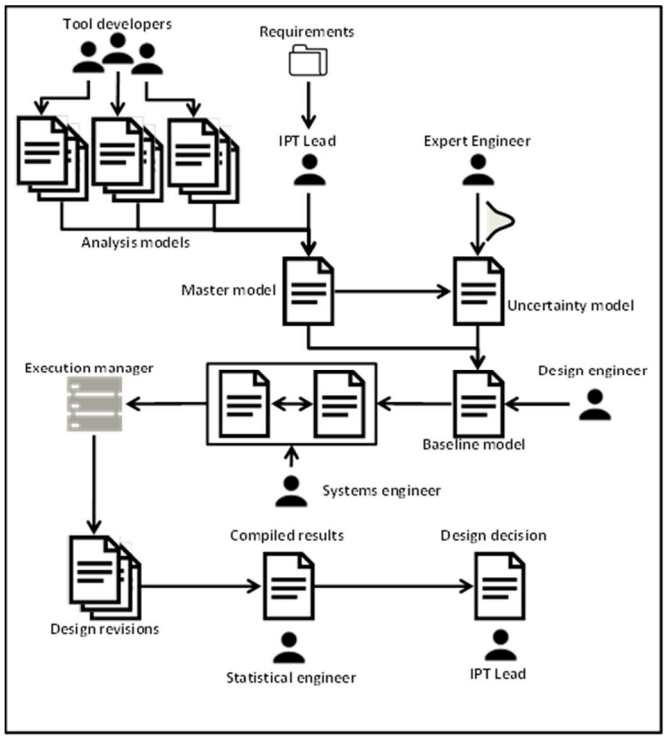

Figure 3: Workflow associated with simulated design of a gas turbine under uncertainty

Figure 3 illustrates the various roles and the tasks associated with these roles in the design of a gas turbine in the presence of technology uncertainty. The tasks begins with the integrated project team lead (role 3) identifying a set of requirements that are to be met. These requirements are translated to a set of utility curves to represent the preferences of various potential customers. In parallel, a set of tool developers (role 1) initiate the development of the analysis models. A set of parameters, much larger that the final set used in the design space exploration process, are exposed such that the parameters encompass the various combinations of technologies that can be studied. Each developer publishes a set of models associated with their discipline upon which the integrated project team lead (role 3) reviews and merges the various models into one master model.

Upon creation of such a master model, in collaboration with the expert engineer (role 2), a set of behavioural representations are attached to the model in order to simulate the physics associated with the component modifying the master model such as to serve as the starting point to be used by other engineers. The expert engineer (role 2), based on experience defines the sources of uncertainty and the associated properties defining the uncertain distributions. This definition of uncertain parameters and their distributions is carried out in a child branch (or a fork), so that the master model can serve as the standardized process to be reused across multiple analyses. Upon completion of application of uncertainties, the models are published ready for the next step, their use in an uncertainty quantification activity. The tasks mentioned previously are carried out for each one of the model that is to be used in the uncertainty quantification process. In the current example, a non-intrusive uncertainty quantification algorithm, a sampling method, is used. The primary advantage of the algorithm is its ability to treat the simulation blocks as black boxes sampling from the defined uncertainty distribution for the variables modifying these values locally to the design. Hence, the systems engineer (role 4) defines a new model containing a definition of the process, the algorithm and the interfaces between the various models.

The systems engineer (role 4) and expert user (role 3) parameterize the model in a way that is manageable by the uncertainty quantification algorithms. Typically, this parameterization does not account for all the variable in the model at which point the regular engineer (role 5) introduces a baseline upon which the uncertainty quantification process is to be carried out. Upon introduction of the baseline, the process is launched by the execution manager, where the various design revisions are created exploiting the distributed computability inherent to the uncertainty quantification algorithm. Each design is stored locally in the engineer's database with the metadata for each design being published to the central database from which an aggregated design history is generated. At any point, a collection of generated designs can be published to replace the metadata on the central database.

Finally, during the execution (and upon completion) the statistical engineer (role 6) reviews the results based on the aggregated design history from which decisions are made based. Typically, decisions involve the down-selection of a select few designs from the vast number generated during the design space exploration process. These decisions are

transmitted to the design engineer (role 5) and to the Integrated Project Team lead (role 3). The design engineer (role 5) then publishes the determined designs, which are transmitted to the management by the Integrated Project Team lead (role 3).

DISCUSSION

Design of complex systems require the development and management of tools, processes and scripts that in the past have often been stored and revised in ad-hoc locations in a engineering organization. This was probably sufficient for smaller-scale products and teams as in the past. Today's challenges of designing complex systems require frameworks where engineers can build on each other models and scripts, revise these models over time, connect them for automation, and insure modelling and simulation traceability. The approach proposed in this paper attempts to address these issues.

The benefits of the proposed framework are: 1) overall gas turbine performance increase due to system analysis as opposed to isolated components analysis, 2) reduction of process development time due to the framework support to the subject matter experts for including new processes, 3) process standardization due to the model reusability and a 4) fast and easy-to-setup technology impact evaluation.

A key future development planned for the expansion of the object oriented data mining and advanced visualization modules in order to make them available to the engineers via the user interface. A key aspect of this expansion is the coupling of data-mining techniques in order to drive the process of design, i.e., guided design space exploration, where engineers due to experience are often aware of the region of interest but due to modelling limitations have to wait for the exploration process to determine the region based on its natural progression. With the concept of guided design space exploration, an automated algorithm based on past engineering steps quickly guides the design space exploration algorithm to the region of interest resulting in a reduction in the design cycle time.

Further, the concept of requirements analysis is currently static in that the requirements defined initially are not updated until the design space exploration process is terminated. While the design space exploration design steps though its natural life cycle, there are often changes to design requirements, either due to the volatile market environment or due to changing subsystem designs. Thus there is not only a necessity to handle requirements dynamically during the design process, but also one to correspondingly alter the design space to correspond to the altering requirements. It is the intention of the author to follow up the current work with a paper on dynamically handling requirements over the life cycle of the design. The current work serves to prefigure this future work.

REFERENCES

[1] Systems Engineering Guide, MITRE Corporation, 2014

[2] Dieter, GE, "Engineering Design: A Materials and Processing Approach", 3rd Edition, McGraw Hill Publications

[3] Friedenthal, S, Moore, A, and Steiner, R, "A practical guide to SysML: The System Modeling Language", 2nd Ed., MK/OMG Press

[4] Ray, B, Posentt, D, Filkov, V, Devanbu PT, "A Large Scale Study of Programming Languages and Code Quality in Github", Proc. 22nd ACM SIGSOFT International Symposium on Foundation of Software Engineering, 2014

[5] van Rossum, G, "Python tutorial, Technical Report" CS-R9526, Centrum voor Wiskunde en Informatica (CWI), 1995

[6] Adams, BM, Bauman, LE, Bohnhoff, WJ, *et al.*, "Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.0 User's Manual," Sandia Technical Report SAND2014-4633, July 2014.

[7] Jones E, Oliphant E, Peterson P, *et al.,* "SciPy: Open Source Scientific Tools for Python", 2001-, http://www.scipy.org/

[8] Perez, RE., Jansen, PW, and Martins JRRA "pyOpt: A Python-Based Object-Oriented Framework for Nonlinear Constrained Optimization", Structures and Multidisciplinary Optimization, Vol. 45(1) 2012, pp. 101-118

[9] Pedregosa *et al.*, Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825-2830, 2011.

[10] Hunter, JD, "Matplotlib: A 2D graphics environment, J. Computing in Science & Engineering, Vol. 9:3, 2007, pp. 90-95.

[11] "PyQtGraph: Scientific Graphics and GUI Library for Python", www.pyqtgraph.org

[12] Zilles, C, Sohi, G, "Master/slave Speculative Parallelization", Proc. 35th International Symposium on Microarchitecture, 2002

[13] PyQt Whitepaper, www.riverbankcomputing.com

[14] Lindhold, T., "A three-way merge for XML documents", Symposium on Document Engineering, 2004

[15] Balestrini-Robinson, S, Freeman, DF, and Browne, DC, "An object-oriented and executable SysML framework for rapid model development", Procedia Computer Science, Vol. 44, 2015, pp. 423-432

[16] Feinberg, J, Langtangen, HP, "Chaospy: An open source tool for designing methods of uncertainty quantification", J. Computational Science., Vol. 11, 2015, pp. 46–57