AMO – Advanced Modeling and Optimization, Volume 17, Number 2, 2015

Decentralized Automated Engineering Analysis Workflow Development and Execution

Steven Finley

Senior Engineer
Pratt & Whitney – Hot Section Engineering – Advanced Methods
400 Main St.
East Hartford, CT 06118
steven.finley@pw.utc.com

Abstract

Development and execution of automated engineering analysis workflows has become more important in industry as integrated product development teams seek to rapidly explore multi-disciplinary design spaces and execute optimization and design for variation studies. Traditional centralized approaches to automated workflow development require a single person to build and execute the workflow. This approach is infeasible for high fidelity analyses because it is too challenging for one person to retain all the modeling and automation expertise for each model and analysis in the workflow. This paper proposes a decentralized approach when engineers collaboratively build and execute an engineering analysis workflow together. To execute a decentralized approach, a new tool called Collaborative Computing Environment was developed and is discussed in this paper. Furthermore, the Collaborative Computing Environment tool enables engineers to easily follow software best practices such as revision management, modular coding, and testing. These software best practices allow for more efficient automated workflow development. To demonstrate the feasibility of both the decentralized approach and application of software development best practices to automated workflow development, the approach is applied to a real world design.

Keywords: Automated Engineering Analysis Workflow, Computer Collaboration, Integrated Simulation

Acronyms
API – Application Program Interface
CAD – Computer Aided Design
CCE – Collaborative Computing Environment
CFD – Computational Fluid Dynamics
DFV – Design for Variation
DOE – Design of Experiments
DOME – Distributed Object-based Modeling Environment

AMO - Advanced Modeling and Optimization. ISSN: 1841-4311

GUI – Graphical User Interface
HPT – High Pressure Turbine
IPT – Integrated Product Team
LCF – Low Cycle Fatigue
LHS – Latin Hypercube Sample
LPT – Low Pressure Turbine
LSF – Load Sharing Facility
TMF – Thermal Mechanical Fatigue
TMTF – Turning Mid-Turbine Frame

Software Packages
ANSYS® (ANSYS®, 2007)
ANSYS Workbench® (ANSYS Workbench®, 2013)
Apache Subversion® (Apache Subversion®, 2006)
Isight® (Isight®, 2011)
PHX ModelCenter® (PHX ModelCenter®, 2011)
Remote Solve Manager® (ANSYS Workbench®, 2013)
Revision Manager® (Revision Manager®, 2009)
SIMULIA Execution Engine® (SIMULIA Execution Engine®, 2013)
Unigraphics® (Unigraphics®, 2008)
Unigraphics Advanced Simlutation® (Unigraphics®, 2008)

For readability, these software packages will no longer be cited throughout the paper.

1.0 Introduction

Automated engineering analysis workflows are becoming more prevalent. There are many potential benefits to analyzing the integrated behavior of a potential design throughout the design cycle. Integrated simulations allow engineers to rapidly explore the design and quickly analyze different design topologies. This yields higher quality products with better performance. The ability to rapidly explore a design space enables creative and try innovative designs.

Automated engineering analysis workflows are also the foundation to creating emulators and executing probabilistic analyses for high fidelity models and analyses. Companies are applying probabilistic design and analyses techniques such as Design for Variation (DFV) (Reinman et al. 2012). The DFV methodology for physics-based model requires a robust parametric model that can be driven through a Design of Experiments (DOE). The DOE results are used as training data for the development of emulators that enable probabilistic analyses. For example, Bunker (2009) utilized a simple automated workflow to access the effects of manufacturing tolerances on gas turbine cooling. Beyond variation and uncertainty quantification, automated engineering analysis workflows facilitate rapid design space exploration and optimization.

Decentralized Automated Engineering Analysis Workflow Development and Execution

There are several challenges to building and executing an automated workflow. First, tool automation requires a different skill set in addition to knowledge of tool usage and application. Many engineering applications provide automation capability through macro languages and/or Application Program Interfaces (APIs). This requires programming and software development skills. In addition, engineers need to manage aspects of the program such as license availability in automated workflows. Second, compute resources are required to execute the workflow. For simple workflows, a single desktop computer can be sufficient. Workflows with long running physics-based models require high performance computing systems. Thus, knowledge of high performance computing system execution software such as Load Sharing Facility (LSF) and parallel execution is required. Third, it is often the case that automated workflows must be built, executed, and post processed within aggressive schedules. Fourth, knowledge of parametric geometric modelling and robust meshing definition is needed if the workflow involves a Computer Aided Design (CAD) model. Fifth, technical expertise is required for each model of the workflow to verify that the model accurately represents the physics of the problem it is intended to model. Finally, engineers need to be able to quickly test and debug each piece of the workflow, as well as, the entire workflow.

The challenges listed above are more acute for multi-disciplinary problems that required an Integrated Product Team (IPT). An IPT is divided based on different engineering disciplines. For example, an IPT may include aerodynamic, design, thermal, structural, and manufacturing engineers. Each discipline engineer is responsible for their particular aspect of the part but the entire team must work together to develop a part that satisfies all design criteria. These types of problems are inherently more challenging and require multiple engineers. Thus, the technical expertise and the tool knowledge to understand and analyze the problem are split amongst several individuals. This creates difficulty when developing automated workflows.

There are several software packages that facilitate the development of automated workflows such as Unigraphics Advanced Simlutation, ANSYS Workbench, Isight, and PHX ModelCenter. Each of these tools provides methods via a Graphical User Interface (GUI) to link tools and analyses together. Isight works with SIMULIA Execution Engine and ANSYS Workbench has Remote Solve Manager to simplify high performance computing system execution. These tools enable the building, testing, and debugging of workflows.

There are still challenges to automated workflow development and execution that these software packages do not address. A couple key challenges are the development of robust parametric model and the required technical knowledge to analyze the automated workflow results. While Unigraphics and ANSYS Workbench certainly facilitate the development of CAD models, the creation of robust parametric models is left to the skill and knowledge of the engineer. Furthermore, these tools are intended to be utilized by a single engineer who is responsible to building, testing, and executing the automated workflow. This is known as a centralized workflow.

Centralized automated workflow development and execution works well for single discipline workflows but does not easily scale to high fidelity multi-disciplinary workflows that require IPTs. These types of problems need several engineers, who are responsible for different models, to execute the entire analysis manually. It is unrealistic for single engineer to be required to build, test, and execute the automated workflow. A single person would need to have the parametric robust modeling skills and the technical expertise for each tool and discipline within the workflow. If a team is required to run the analyses manually, a single engineer cannot be expected to run the analysis automatically.

There have been many efforts to develop a decentralized approach through an integrated modeling environment. Wallace et al. (2001) developed an integrated simulation environment called Distributed Object-based Modeling Environment (DOME) based on the World-Wide Web. This framework is best suited for problems where only meta-data is shared between models and analyses are relatively simple and robust. Wong and Sriram (1993) created an information model for incorporating product information. Toye et al. (1994) created a prototype environment to help design teams gather, organize, re-access, and communicate both informal and formal design information. Wellman (1994) applied a market model using “design economies” to well-defined design problems and demonstrated that a design can be created relatively quickly for simple examples. Molina et al. (1995) provides a summary of research on computer systems in support of simultaneous engineering. Bliznakov et al. (1996) describe an environment for meta-level design information integration of CAD systems with other application programs. Case and Lu (1996), Cutkosky et al. (1996), Dabke and Cox (1998), and Kim and Kim (1998) all researched the development of a distributed system for collaborative design.

Despite the wide range of approaches, there is a key challenge that is not addressed. These efforts do not provide a clear mechanism for building and testing individual pieces of the workflow. If the individual pieces of a workflow are not robust, then the overall workflow will not execute. When workflows are applied to models that involve complex geometry and high fidelity models, testing the individual workflow pieces is critical. This research applies software development and testing practices, such as, revision management, object-oriented programing, and unit testing to decentralized automated workflow development.

This paper proposes a collaborative decentralized approach to multi-disciplinary workflows that enables the application of software development best practices to automated workflow development. This approach follows the philosophy described by (Cao and Wallace, 2012). The decentralized approach and systematic procedure are rooted in software development practices such as revision management (O’Sullivan, 2009), object oriented programming (Savitch, 2007), and code testing (Kumar and Bansal, 2013). The goal of this paper is to demonstrate that principles from one field of study, computer science, can be applied to computer simulation in the field of

mechanical engineering. In fact, these principles can be applied to any scientific field that seeks to couple complex computer simulations into an automated workflow. A new tool called Collaborative Computing Environment (CCE) that facilitates the application of these principles similar to integrated development environments and other tools that facilitate software development. The decentralized approach via CCE address the workflow development challenges that the centralized approach fails to handle while still retaining the benefits of the centralized tools and approach.

Centralized and decentralized workflows and appropriate applications are discussed in section 2. Next, section 3 talks about the architecture of CCE and how it can be applied to both centralized and decentralized workflows. Section 4 discusses how CCE enables the application of software development practices. The feasibility of the decentralized approach and the workflow development procedure is demonstrated in section 5 by discussing a real world application. Finally, conclusions and potential future work are addressed in section 6.

2.0 Centralized and Decentralized Workflows

Engineers need a methodology to determine if a centralized or decentralized approach should be employed for automated workflow building and execution. There are two aspects of an automated workflow that can be centralized or decentralized: workflow building and workflow execution. When a workflow is fully centralized, it is built and executed by a single individual from a central location. A fully decentralized workflow is built and executed by multiple people. The decision between building a centralized or decentralized workflow is situation dependent as shown in Figure 1. As a general guideline, answer this question: "If this work was going to be executed manually, how many practitioners would be needed?" If the answer is one, then a centralized approach is appropriate, otherwise, a decentralized approach is typically the appropriate choice.

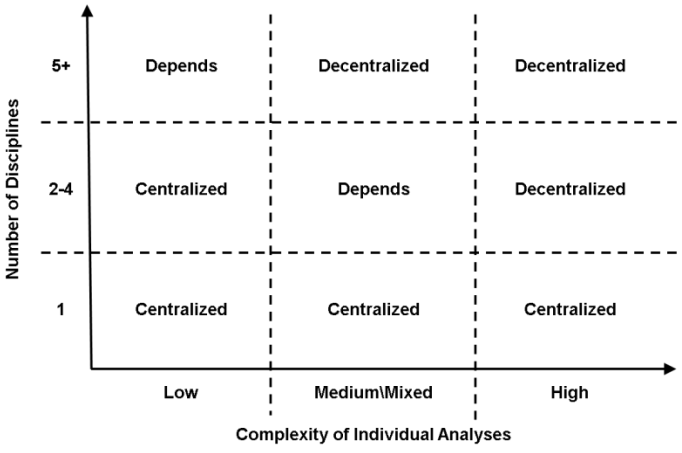

Figure 1: Decentralized vs. Centralized Decision Matrix

A centralized approach is appropriate when a single practitioner typically executes each of the individual steps. This occurs most often when a single discipline is involved or when lower fidelity tools are used by a single practitioner to model several components or engineering analyses. One practitioner is able to understand all of the analyses involved and can be expected to build, test, and execute the entire workflow independently. All the analysis files can be organized and managed in a central location

A decentralized workflow approach is appropriate for workflows involving multiple disciplines working as an IPT. An individual is unlikely to possess all the required technical and modeling expertise. An exceptional engineer may be able to do it but it is unlikely that the strategy would be successful throughout the entire work force. Furthermore, each discipline engineer has expertise in their particular discipline. The decentralized approach keeps the models with the appropriate engineers.

In some cases it is not clear which approach should be used. For example, a single engineer may be able to handle several low fidelity tools in a centralized manner. At some point, the quantity of tools becomes so large a single engineer cannot manage all the tools and a decentralized approach should be used. It depends on the situation when one switches from the centralized to the decentralized approach.

3.0 CCE Architecture

CCE was developed to aid engineers in building decentralized workflows. However, CCE is still useful for centralized workflows. CCE is comprised of several self-functioning modules. Each module can be run individually or concurrently with the other modules. This organization creates a flexible approach for developing automated workflows. The user can focus on a single piece of the workflow or the interactions of several components. The ten modules are described here.

Listener: The listener sits and waits for the upstream tasks that are being run by another discipline engineer to be completed before allowing the rest of the CCE process to execute. When files have been checked into a revision manager system call Revision Manager via subversion commands, the listener is triggered and it copies files into the user working copy. Once the copying is finished, the listener is complete and the subsequent CCE modules can begin. The listener is only used in a decentralized workflow and is not used with the make DOE module, which is described in the next paragraph.

Make DOE: The make DOE module generates a DOE file called doe.txt, which lists all of the variable values for each design point that is to be executed. The make DOE can generate a DOE file with a single nominal value, a set of max-min cases, or a Latin Hypercube Sample (LHS) (Fang et al. 2006). A set of max-min cases is where one variable is at its max with the rest of the variables at their nominal values. For the next

design point, the same variable is at its min value with the rest of the variables at nominal. This sequence is repeated for every variable. A set of max-min cases is useful for testing. If desired, a DOE file can be created by an external source for a custom set of cases. The LHS is typically used as training data for a future emulator (Santner et al. 2003). The make DOE module is not used with the listener module.

Custom Pre-Process: The custom pre-process module allows users to run any custom program prior to the design point execution.

Submitter: The submitter module runs all of the jobs that are defined by the DOE file. The submitter can either run the jobs serially and locally or can run the jobs in parallel on a compute cluster. The submitter module runs the jobs in parallel by communicating with LSF, which is installed on the compute cluster. The submitter module is responsible for copying the necessary files to the compute cluster, executing the job, and copying back the desired files. The submitter also communicates the run status of each batch job.

Job Status Checker: The job status checker module is primarily responsible for waiting until the batch jobs run by the submitter are complete. It retrieves the status for each job from the submitter. Once all the jobs are complete, the job status checker allows post batch job execution scripts to run.

Check for Files*:* One way to determine if a batch job was successful is to determine if it produced the required output files. The check for files module will determine if the required output files were produced for each batch run. The results are then stored in a summary file that lists a 1 or 0 for each case based on the existence of a file.

Summarize: The summarize module gathers results from each of the batch runs and creates summary files that contains the results for all of the runs. The summarize module can gather results from any text file that has a “key=value” format.

Check Success*:* If a batch job produced all of the required output files and numbers, then that job is considered to be a good case because it ran to completion. A more thorough analysis is required to determine if the results are valid. The check success module reads the summary files produced by the check for files and summarize modules to determine if each batch job was successful.

Post Process & Custom Post-Process: The post processing module formats the data for statistical analysis software and other post processing programs. There is also an option to run a custom post processing executable.

Pusher: The last module is the pusher. This module creates a “path to results” file. It then checks the path to results file, DOE file, and variation definition file into Revision

Manager via subversion commands. Any subsequent tasks are then triggered and will begin execution.

Figure 2 depicts the modules that are utilized for centralized and decentralized workflows as well as the input and output files. This illustrates the process flow. Note that the custom pre-process and post-process modules are not shown in Figure 2.

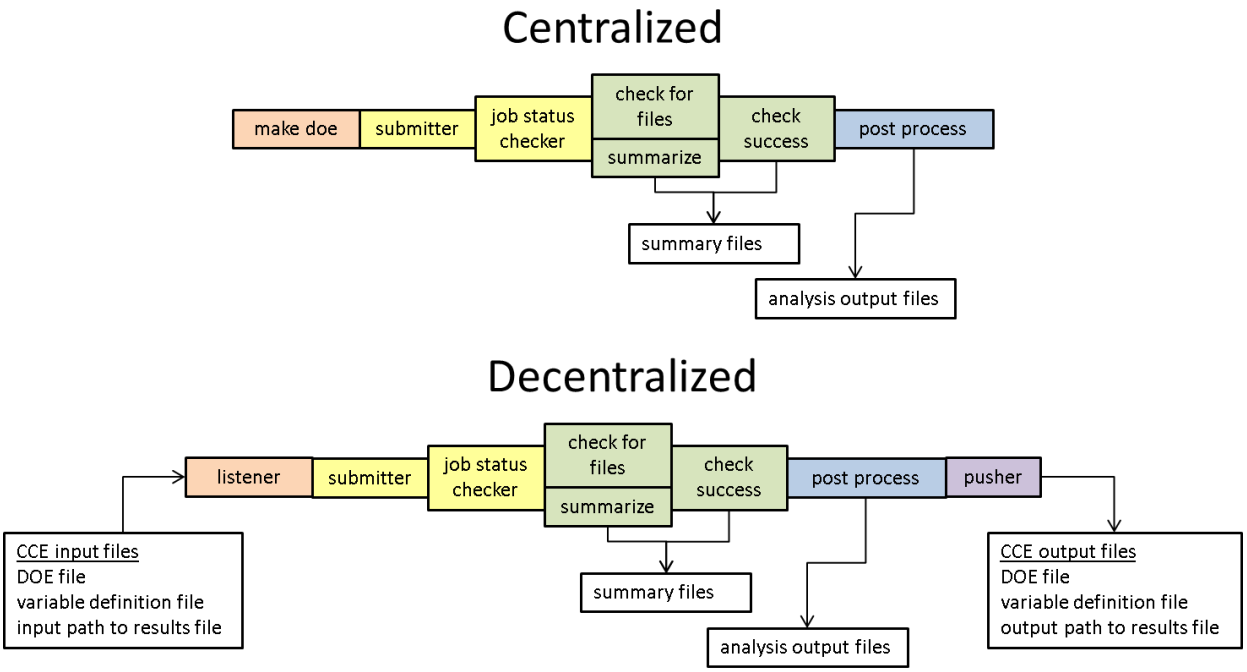

Figure 2: Centralized and Decentralized Process Flow

For a decentralized workflow, Figure 2 shows the modules that are used by a single discipline. This becomes a piece of a larger workflow decentralized workflow as shown in Figure 3.

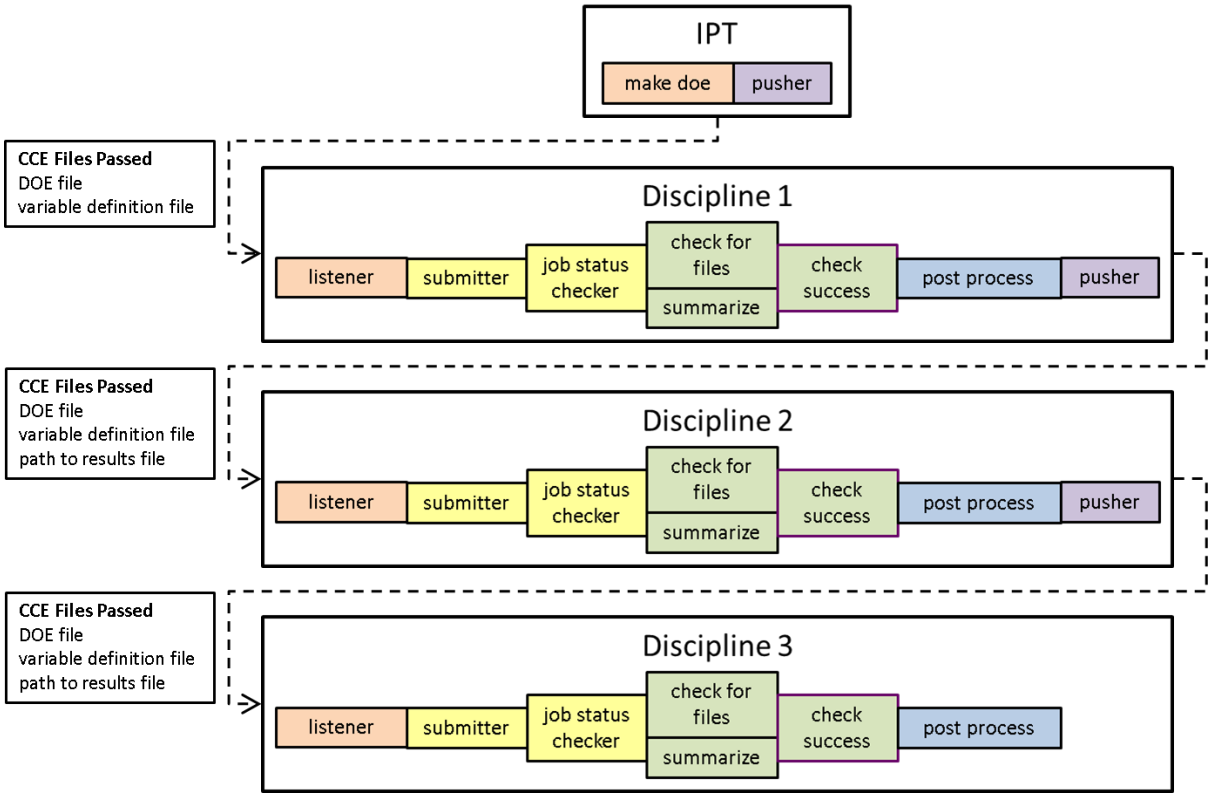

Figure 3: Example 3 Discipline Decentralized Workflow

The decentralized workflow shown in Figure 3 can be thought of as a collection of centralized workflows. This is intended because it minimizes the work required when switching back and forth from testing a piece of the workflow to running the entire workflow. Each centralized workflow can be thought of as an object from object oriented programming. It is a bundle of information that defines how the model behaves and it has a clear interface to the other centralized workflows. The files that are passed between disciplines are CCE input and output files shown in Figure 2. Notice that a small workflow that simply creates a DOE starts the larger workflow. This small workflow is owned by the entire team since everyone is responsible for defining the design space. Thus, the DOE generation is broken out separately rather than being part of the first discipline's workflow.

There are three requirements that CCE places on the task automation. First, the task automation must be able to be run from a command line in the background on a single operating system. This is known as running in batch. Second, the task must completely run a single analysis in the current working directory. All the files needed to execute the task must be in the current working directory. This does not mean that the task cannot reference programs installed in a network location as long as that location is known to the compute cluster. The task automation simply needs to handle a single case since CCE handles running multiple cases. Finally, the task must take a variation text file as an input and produce "key=value" formatted files as an output. This is the

defined interface between CCE and task. See Figure 4 for an illustration that depicts the CCE-task interface. It should be noted that a task can be executed manually if warranted by the situation. However, an automated task is preferred.

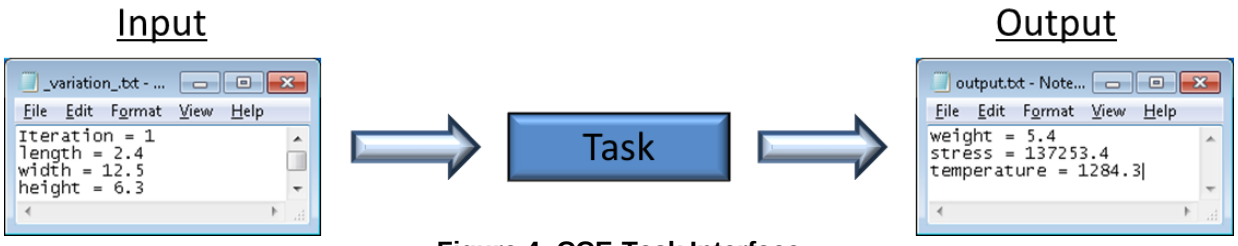

Figure 4: CCE-Task Interface

CCE requires that the variation file be named "_variation_.txt". The file lists input variables and values in a "key=value" format. This is a file produced by CCE. The output files list output variables and values and the file itself can have any name. This simple text file interface allows CCE to work with any task automation method that satisfies the three requirements listed above. Thus, CCE has no tool dependencies.

Tool automation refers to the tools that are included in an individual task. All tool automation must be completed as a prerequisite to running CCE. Tool automation can involve a single tool or multiple tools. There are multiple ways to automate a task. Many engineering tools such as Unigraphics and ANSYS provide APIs or macro languages. Code written via an API or macro language provides a method to automate the tool. Engineering tools have started to add additional capability through products such as Unigraphics Advanced Simulation and ANSYS Workbench for running multiple tools as part of a task. There are also tools such as Isight from Simulia® and PHX ModelCenter from Phoenix Integration® that provide generic capability to link multiple tools. In some instances, tool automation can be a custom program. The best approach for tool automation is task dependent.

4.0 Software Development Best Practices

CCE was developed such that software development best practices can be easily applied when building an automated workflow. The application of these best practices reduces development time and improves workflow robustness.

4.1 Revision Management

Revision management tracks the changes to documents during the building and execution of automated workflows. Revision management removes ambiguities from file versions because the latest is always the last version checked into the revision management system. It also keeps track of changes so the developer does not have to do housekeeping of files and folders. While revision management will not force good naming conventions, it will keep track of who did what and when. The files are kept in a central, backed-up, repository where engineers can check out working copies. This

enables multiple developers to work on the same project simultaneously. Furthermore, a single engineer can manage multiple working copies for separate ideas. This helps the developer follow a “one idea, one commit” philosophy. This is important because it makes it more efficient to identify and fix bugs and improves traceability. CCE directly integrates with a revision management system (Revision Manger) through the listener and pusher modules. Using a revision management system allows the engineer to focus more time on the analysis and less time on file management and organization. This facilitates the building and maintenance of automated workflows. The setup of a revision management system is dependent on the tools used for revision management.

4.2 Object-Oriented Programming and Modular Coding

Each task of the workflow can be thought of its own independent object with an interface. The task contains all the data, attributes, and methods. With an object-oriented approach, individual tasks of a larger workflow can be built separately, see Figure 3. The only requirement is that all the inputs and outputs of the various tasks be in sync with one another. Breaking the workflow down into independent tasks allows workflow development to be parallelized amongst multiple developers. This speeds up workflow development, which is important to ensure that schedule restraints are satisfied.

4.3 Testing

Since the workflow has been modularly developed, each task of the workflow can be tested individually. This is similar to unit testing in software development. A workflow builder can follow a develop-debug loop like a software developer. Each task of the workflow can be fully tested before being integrated with the larger workflow. This makes debugging easier because it allows the builder to focus on a single piece of the workflow while testing. Once the entire workflow has been coupled, it is possible to employ system testing. This testing ensures that all of the tasks of the workflow fit together.

5.0 Example Application – Turning Mid-Turbine Frame

To demonstrate the feasibility of the decentralized workflow approach and workflow development procedure, the approach and procedure are applied to a real world problem.

5.1 Problem Introduction

A Turning Mid-Turbine Frame (TMTF) is in the gas path between the High Pressure Turbine (HPT) and Low Pressure Turbine (LPT) of a jet engine. The purpose of a TMTF is to allow access to the shaft for a bearing while minimizing aerodynamic loses. The TMTF also protects the bearing structure from high gas path temperatures. It is

challenging to design a TMTF due to harsh boundary conditions, model uncertainty, and multiple objectives. The figure below shows a representative engine cross section with a TMTF.

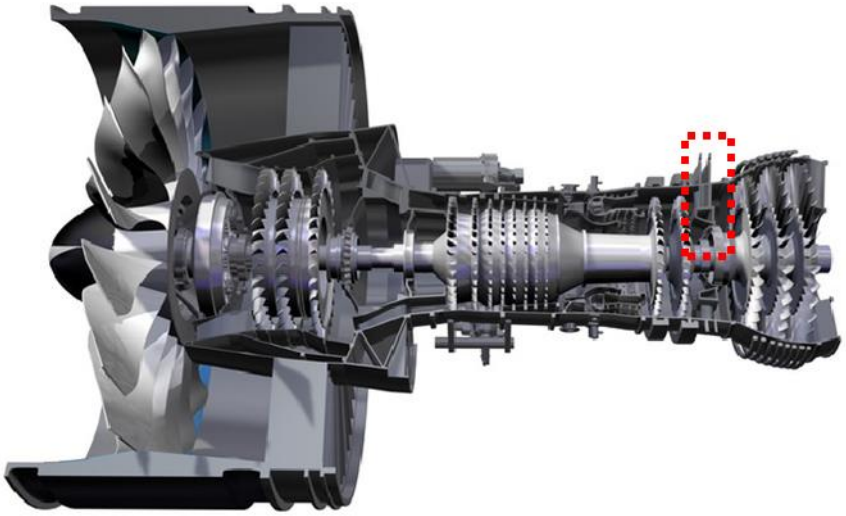

Figure 5: PW1000G Cross Section with Highlighted TMTF Region (http://www.a320neo.com/pratt-whitney-pw1000g.php)

When designing a TMTF, the goal is to ensure that production hardware meets the requirements in the production engines. This is a difficult problem because there is a great deal of uncertainty. For example, the gas path temperature profile is an important driver of Thermal Mechanical Fatigue (TMF) and Low Cycle Fatigue (LCF) life. The actual gas path temperature profile that is seen by the hardware in the engine is unknown. In many instances, available gas path temperature profile data is from engines with different configurations. Models that are used to predict the gas path temperature profile have inherent uncertainty. Furthermore, the models that are used to predict the stress, strain, metal temperature, and life have inherent uncertainty. The models may contain inaccurate parameters, biases, or stochastic residual error. Finally, manufacturing variation effects final part performance. While the nominal design may meet the objectives, a manufactured part within tolerance may not. Clearly, it is difficult to meet all of the requirements without needlessly increasing cost given the high degree of uncertainty.

The goals for the TMTF IPT were to allow for more aerodynamic analysis iterations within the schedule, quantify the effect of temperature profile uncertainty, recover previous design life requirement misses, and recover the LPT module efficiency miss. Typically, TMTF IPTs were able to manually execute roughly twelve analyses in six months. Within these twelve analyses, only one to two aerodynamic iterations would be executed.

The team decided to apply the DFV approach (Reinman et al. 2012) to achieve the assigned goals. This required the development of a multi-disciplinary automated workflow. The CCE tool and the workflow development procedure were applied to develop a decentralized automated workflow.

5.2 The workflow

The TMTF was a full wheel cast part consisting of fourteen airfoils. Only two airfoils were analyzed with cyclic symmetry to capture the necessary physics while reducing computational solution time. Twenty-nine input variables were parametric and thirty-three outputs were tracked in the automated workflow. For the twenty-nine input variables, twenty-four were geometric and five defined the temperature profile. The geometry is shown in the figure below.

Figure 6: TMTF Geometry

These parameters were selected because they were expected to be important drivers of stress, temperature, and life in the sixteen fillet locations. This was based on previous sensitivity analyses and engineering experience. Design space limits were assigned to geometric parameters based on expert opinion that far exceeded typical manufacturing tolerances because the goal was to find a nominal geometry that was robust to variation on uncertainty.

The thermal temperature profile was a 1D radial profile that is defined by a b-spline. Engine data was utilized to determine the number of the b-spline coefficients and corresponding limits. TMF life and LCF was tracked at every fillet location for each airfoil in the model for a total of thirty-two life outputs. The last output was the weight of the part.

The engineering analysis workflow consists of the following analyses: 3d geometry generation, gas path temperature profile generation, external thermal boundary condition generation and application, internal thermal boundary condition generation and application, thermal meshing, thermal Finite Element Model (FEM) solution,

structural meshing, structural FEM solution, TMF life calculation, and LCF life calculation. The engineering workflow is shown in the figure below.

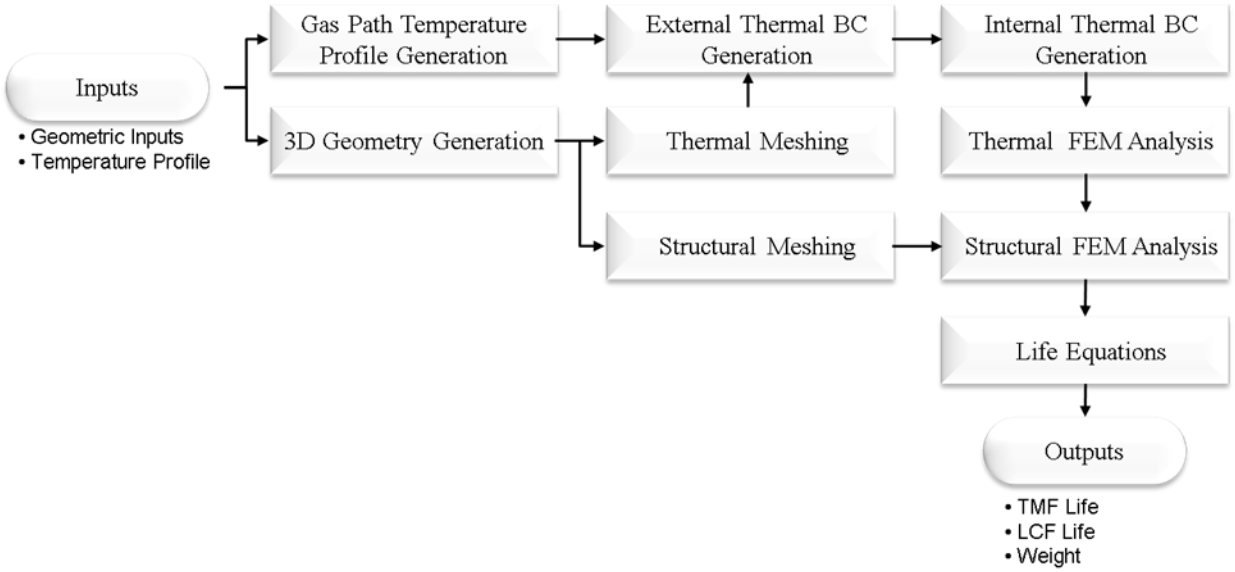

Figure 7: Automated Engineering Analysis Workflow

The 3-dimensional TMTF geometry was modeled and meshed using Unigraphics and Unigraphics Advanced Simulation. The weight of the part was also calculated in Unigraphics. Two separate meshes were created for the analysis, one thermal and one structural. Transient internal and external boundary conditions were generated via proprietary Pratt & Whitney software. The thermal and structural analyses were solved in ANSYS. During the transient structural solution, the stress, strain, and metal temperature were stored at the life limiting time point for the life limiting regions. These values were used to calculate TMF and LCF life via propriety Pratt and Whitney equations.

5.3 Workflow Development and Execution

All of the analyses were coupled together in a decentralized automated engineering analysis workflow by applying the systematic approach and the CCE tool. Every particular analysis was assigned to a discipline engineer. Each individual piece of the workflow was made to be parametric and compatible with a compute cluster. Once testing was complete, the IPT collaboratively coupled the entire workflow together and executed the workflow. There were many building-testing iterations loops at both the discipline analysis and full workflow levels. As the design progressed, topology changes required rebuilding and testing of the automated workflow.

The team was able to execute approximately 1200 design iterations. The largest DOE that was executed contained 870 discrete design points. A Latin Hypercube (Fang et al. 2006) was generated to fill the design space. There were twenty-nine input variables in

the Latin Hypercube. The team selected thirty runs per input variable because this was expected to sufficiently fill the design space to produce an accurate emulator. The automated workflow achieved a 60.9% success rate. If a design point was executed through the entire workflow and it produced valid results, it is considered successful. Due to schedule constraints, the IPT was not able to improve the robustness of the models any further. A part design was required by a specified date to meet the overall engine development schedule. The biggest causes of failure were structural meshing and parametric geometry. The structural meshing was difficult due to refined sub models and maintaining boundary conditions at the cut plane boundaries. The parametric geometry would fail due to complex endwall contouring not updating in the Unigraphics model.

While parametric modeling and robust meshing remained challenging, the overall automation was not a challenge. Files were successfully passed between discipline analyses and the team was able to work together to build the automated workflow. The team always used the latest and correct files in the automated workflow. This showed the benefit of applying revision management to workflow development.

To overcome the parametric modeling and robust meshing challenges, the team implemented a 'person in the loop' optimization strategy. In a 'person in the loop' optimization strategy, the team replaced the role of a formal optimization algorithm and designed the part. The IPT selected the design points to execute based on the previous automated workflow results and their expertise. The team would select approximately five to fifteen points to run. Each team member used the automation to run the points. For cases that failed, manual intervention was employed to fix the issue. Once all the cases worked for a given discipline analysis, the next discipline in the workflow would execute. This process would take anywhere from one to three days, which is much faster than the original manual process. The 'person in the loop' optimization was successful because each discipline had ownership of their piece of the workflow and had the expertise to manually fix failed cases. The person in the loop optimization combined with the DFV analysis allowed the team to satisfy all of the design requirements. Table 1 shows the benefits to the TMTF as a result of the person in the loop optimization and the DFV results.

Metric	**Improvement**
Weight	2.2% Below Requirement
LCF Life	1.8x Over Requirement
TMF Life	2.5x Over Requirement

Table 1: Final TMTF Results

6.0 Conclusions and Future Work

The TMTF example illustrates that the decentralized workflow approach and the systematic workflow development procedure are viable approaches for workflow development in real workflow problem. The decentralized approach kept the various models with the discipline experts. Each team member built and tested their piece of the workflow in parallel. The entire team collaboratively assembled and executed the entire workflow within schedule constraints. The decentralized approach allowed the team to make significant progress in overcoming the challenges of robust parametric modeling and meshing to the point where the team could make design decisions based on the workflow.

Furthermore, computer science concepts such as revision manager, code testing, and object oriented programming are applicable to automated workflows in the field of Mechanical Engineering. CCE and Revision Manager were the tools that allowed the team to effectively follow these principles. This is similar to how an integrated development environment facilitates software development. These principles enabled the team to efficiently develop an automated workflow within schedule constraints.

For future work, the decentralized and centralized approaches should be quantifiably compared. In addition, the benefits of a systematic workflow develop procedure should be quantified relative to a non-structured approach. While the decentralized approach has been shown to work for DOEs, it has not been applied to optimization or Monte-Carlo analyses. Finally, hybrid approaches, where the building and execution of the workflow follow different decentralized and centralized approaches can be investigated.

References

ANSYS® (2007). ANSYS Mechanical Version 11.0, ANSYS, Canonsburg, PA.

ANSYS Workbench® (2013). Version 15.0, ANSYS, Canonsburg, PA.

Apache Subversion® (2006). Version 1.4.6, The Apache Software Foundation.

Bliznakov, P. I. et al. (1996). “Integration Infrastructure to Support Concurrence and Collaboration in Engineering Design”. *1996 ASME Design Engineering Technical Conferences*, Irvine, CA.

Bunker, R. S. (2009). “The Effect of Manufacturing Tolerances on Gas Turbine Cooling”. *ASME: Journal of Turbomachinery*, 131, 1 – 11.

Cao, Q. and Wallace, D. (2012). “Crowd-driver Ecosystem for Evolutionary Design”. Defense Advanced Research Projects Agency: Tactical Technology Office (TTO). DARPA/CMO Contract No. HR0011-11-C-0092.

Case, M. P. and Lu, S. C.-Y. (1996). “Discourse model for collaborative design”. *Computer-Aided Design*, 28, 333-345.

Cutkosky, M. R. et al. (1996). “PACT: An Experiment in Integrating Concurrent Engineering Systems”. *IEEE Computer*, 28-37.

Dabke, P. and Cox, A. (1998). “NetBuilder: an environment for integrating tools and people”. *Computer-Aided Design*, 30, 465-472.

Fang, K., Li, R., Sudjianto, A. (2006). Design and Modeling for Computer Experiments. Chapman & Hall/CRC, Boca Raton, Florida.

Isight® (2011). Version 5.6-2, Dassault Systemes, Velizy Villacoublay, France.

Kim, C., and Kim, Y. (1998). “Internet-based Concurrent Engineering: An Interactive 3D System with Markup”. *ASME 18th Computers in Engineering Conference*.

Kumar, S., Bansal, S. (2013). “Comparative Study of Test Driven Development Traditional Techniques”. *International Journal of Soft Computing and Engineering (IJSCE),* Volume 3, Issue 1, 2231-2307.

Molina, A. et al. (1995). “A review of computer aided simultaneous engineering systems”. *Research in Engineering Design*, 7, 28-63.

O’Sullivan, B. (2009). “Making sense of revision-control systems”. *Communications of the ACM*, 52, 9, 56-62.

PHX ModelCenter (2011). Version 10.0, Phoenix Integration, Philadelphia, Pennsylvania.

Reinman, G. et al. (2012). “Design for Variation”. *Quality Engineering*, 24:2, 317-345.

Revision Manager® (2009). Version 2.8.3, Oculus Technologies Corporation, Boston, Massachusetts.

Santner, T. J., Williams, B. J., Notz, W. I. (2003). The Design and Analysis of Computer Experiments. New York: Springer-Verlag.

Savitch, W. (2007). Problem Solving with C++ (6th Edition). Boston: Pearson Education, Inc.

SIMULIA Execution Engine® (2013). Version 5.8, Dassault Systemes, Velizy Villacoublay, France.

Toye, G. et al. (1994). "SHARE: a methodology and environment for collaborative product development". *International Journal of Intelligent & Cooperative Information Systems*, 3, 129-153.

Unigraphics® (2008). Version NX 6.0.5.2, Siemens, Munich, Germany.

Wallace, D. (2001). "Integrated Simulation and Design Synthesis". Technical report. Available at http://dspace.mit.edu/handle/1721.1/3802.

Wellman, M. (1994). "A Computational Market Model for Distributed Configuration Design". *12th National Conference on Artificial Intelligence*. AAAI Press, Menlo Park, CA.

Wong, A. and D. Sriram (1993). "SHARED: an information model for cooperative product development". *Research in Engineering Design*, 5, 21-39.