Proceedings of GT2005
ASME Turbo Expo 2005: Power for Land, Sea and Air
June 6-9, 2005, Reno-Tahoe, Nevada, USA

GT2005-68678

DEVELOPMENT OF GAS TURBINE PERFORMANCE MODELS USING A GENERIC SIMULATION TOOL

A. Alexiou
Research Associate
Email: a.alexiou@ltt.ntua.gr

K. Mathioudakis
Associate Professor
Email: kmathiou@ntua.central.gr

Laboratory of Thermal Turbomachines
National Technical University of Athens
PO BOX 64069, Athens 15710, Greece

ABSTRACT

An approach for gas turbine engine modelling using a general purpose object-oriented simulation tool is described. A commercially available such tool that can be adapted to different fields, through the creation of reusable modelling component libraries representing parts or equipment of a physical system, is employed. Libraries are developed using an object-oriented language. The possibility for quick implementation of new models and rapid analysis of results, through the use of a graphical user interface is demonstrated.

A turbofan model, developed for both steady state and transient performance simulation, is used to illustrate the advantages offered by this approach. Results are presented and compared to those produced by an industry-accepted model. The flexibility of incorporating particular features into a model is demonstrated by presenting the implementation of adaptive features and a study of engine frequency response.

Keywords Object-oriented, gas turbine, performance, simulation

NOMENCLATURE

A	Area
A_0	Amplitude
ALT	Altitude
C	Constant
c_p	Specific Heat Capacity
DTAMB	Temperature Difference from Standard Day Temperature
F	Force
f	Frequency
FAR	Fuel Air Ratio
FHV	Fuel Heating Value
$Gain = A_0 / A_{0,f=0.01\ Hz}$	Gain
H	Total Enthalpy
h	Heat Transfer Coefficient
J	Moment of Inertia
K	Pressure Loss Coefficient
k	Thermal Conductivity
L	Length
M	Mass of Metal
Mach	Flow Mach Number
P	Total Pressure
p	Static Pressure
PW	Power
Re	Reynolds Number
RH	Relative Humidity
$SE = \eta / \eta_{ref}$	Efficiency Factor
$SW = (W\sqrt{T}/P) / (W\sqrt{T}/P)_{ref}$	Flow Factor
T	Total Temperature
t	Time
TRQ	Torque
V	Velocity
W	Mass Flow Rate
WAR	Water Air Ratio
WF	Fuel Mass Flow Rate
WW	Water Mass Flow Rate
XM	Flight Mach Number
XMAR	Surge Margin (%)
XN	Rotational Speed
$\Delta\varphi = 2\ \pi\ f\ \Delta t$	Phase Shift
Δ	Difference
η	Efficiency
μ	Dynamic Viscosity
ρ	Density

Subscripts

g	Gas
m	Metal
ref	Reference
s	Surface

INTRODUCTION

Gas turbine engine performance models have evolved to a wide range of applications, providing support from the early stages of preliminary design through to in-service support. Engine models of various fidelity levels are used for design and verification of an engine, support its development and validation, while they play an important role in post-certification and in-service support. For a detailed account of applications as well as other aspects of performance simulation, the reader is referred to [1].

When a new gas turbine engine is designed, an aero-thermodynamic model is created to study its performance at design and off-design conditions. Predictions from this model are then used as input to other analysis codes, e.g. finite element thermo-mechanical stress analysis (FEA), computational fluid dynamics (CFD), etc. The results from these codes often lead to design modifications and improved or new component performance data. These changes must be fed back to the performance model and this procedure is repeated until the design criteria are met. This procedure may be largely facilitated by a flexible, modular environment that allows the user to make the necessary changes rapidly (e.g. incorporate a new compressor design) without affecting the rest of the model. Thus, a user-friendly interface is essential.

Traditionally, these engine cycle models are written in FORTRAN (e.g. [2]), the preferred computer language for scientific programming, because of its robustness and speed of execution. However, the procedural nature of FORTRAN and its programming environment make it difficult to extend, maintain and re-use large codes. In order to support modular design, code re-use, extensibility and user-friendly interface one has to use a modern object-oriented computer language like C++, Java, Borland Delphi, etc. [3].

Currently, there are only a few commercially available gas turbine specific simulation tools based on object-oriented languages. One such tool is GasTurb [4, 5]. Borland Delphi is used to create a user friendly interface through which the user can select among a number of predefined gas turbine configurations. All the mathematical complexity is hidden from the user making this program suitable even for the less experienced user. Although a multitude of parameters give flexibility for modelling a number of engines, it is not possible for the user to build an engine configuration different from those available in the package. This feature can be found in NLR's Gas Turbine Simulation Program, GSP [6, 7]. It is also implemented in Borland Delphi but (compared to GasTurb) makes use of all the object-oriented features offered by it. The user can create different gas turbine configurations by arranging component icons (representing gas turbine engine components) in a model window using a graphical drag & drop interface. When new (non-standard) components are required for specific GSP users, these are supplied in the form of a custom component library; the creation of new components is the job of the developer and not of the user.

A Java-based object-oriented simulation framework developed recently for flexible gas turbine modelling is Onyx [8, 9]. The user can create or modify components representing physical components in the gas turbine engine domain and then combine them either programmatically or visually (like in GSP) to form the required engine model. Higher order analysis methods (like CFD and FEA) can also be integrated within a component object. Support for distributed computing is provided through several common software distribution mechanisms (e.g. CORBA). Onyx was developed to investigate advanced software design techniques as part of NASA's Numerical Propulsion System Simulation (NPSS) project [10, 11]. The aim of NPSS is to reduce time and cost in developing new propulsion systems while increasing confidence and reducing risk in achieving a final design. This is possible due to its open and extensible architecture that fully exploits the capabilities of object-oriented programming. It consists of three layers: the interface layer (using a C++ like language), the object layer and lastly the computing layer for deploying the simulations. Although NPSS is capturing all modern day requirements, what could be considered as a drawback for the wider engine community is the fact that it is only available to the partners involved in its development.

Finally, another option for developing an engine cycle model is to use a generic simulation tool like MATLAB-Simulink [12]. This is especially popular for some types of performance analysis e.g. real time modelling and control system design [13-15]. Simulink has a component-oriented architecture (an engine model is composed of modular blocks representing individual components) but lacks some of the stronger features of the object-oriented programming.

The objective of the present paper is to discuss the features of object oriented methods for gas turbine modelling and propose an approach for building models, using general purpose object-oriented simulation tools. Using a specific such tool, a gas turbine library is created and its components are used to model a typical civil two-spool turbofan engine layout. Both steady state and transient simulations are performed and the results are compared to an industry-accepted standard. The possibilities offered by the presented approach are further demonstrated by materializing two not-so-commonly encountered applications: model adaptation to specific engine, using measured data, and engine frequency response study. The current approach combines the advantages of the tools described earlier as it is based on commercially available object-oriented simulation tools and allows the mathematical modelling of gas turbine components for all engine configurations in a flexible and user friendly interface.

SIMULATION TOOL FEATURES AND MODELLING ADVANTAGES

Object oriented programming offers some features valuable for building gas turbine models. Some basic notions are summarised here, so that the features of the elements for gas turbine simulation are appreciated.

The basic element is the *object*: a re-usable, self-contained entity that consists of both data and methods to manipulate the data. An object can be accessed (by other objects) only through its external interface. Consequently, its internal (private) data and methods are hidden (*encapsulated*) from the other objects. This facilitates the objects' configuration management and ensures that compatible changes can be made safely, thus making easier code evolution and maintenance. Code re-use is achieved through *inheritance* where an object can inherit data and methods from another object that has previously been designed and tested. This concept also allows compound objects to be created by the *aggregation* of objects

that have already been developed. An additional principle of object-orientation is *polymorphism*, which means that different methods can have the same name, but operate differently depending on the context. On the basis of these basic notions, generic simulation tools are available today, offering the environment for development of models oriented to specific application types. Further features of such tools will be presented in the following, by choosing a specific environment that was employed for materializing the applications dealt with in the paper.

The generic object-oriented simulation tool that will be used to demonstrate the features and steps of the proposed approach is EcosimPro™ [16, 17]. It uses a high-level object-oriented language (EL), for modelling physical systems. The most important concept in EL is the component (equivalent to a class in C++); it contains a mathematical description of the corresponding real-world component. Components are joined together through their ports. Ports define the set of variables to be interchanged between connections. The components and ports are stored in a library. Thus, the first step in a simulation is to create (or re-use) a library and then to code the components and port types either using EL or graphically (see Fig. 2). The next step, simulating a component, is to define its associated mathematical model (called a *Partition*). The built-in mathematical algorithms process the equations symbolically, resolve high index problems, solve algebraic loops, suggest boundary conditions and finally sort the equations for efficient calculation. Finally, an *Experiment* (simulation case) is created in EL for that *Partition*. Here, the user can initialise data, calculate steady states, integrate the model over time, generate reports, etc. Steady states are obtained by solving the non-linear system of algebraic equations that result from automatically declaring the time derivatives of all the dynamic variables to be equal to zero. The default integration method for the system of differential-algebraic equations is DASSL [18]. The solution method is based on replacing any time derivatives with finite difference approximations and then solving the resulting system of equations using implicit Newton iterations.

Contemporary simulation tools have a graphical user interface (GUI) that supports code development and debugging, graphical model creation and visualisation of results. These GUI features for the tool used herein are shown in Figs. 1-3. The main window, Fig. 1, can be used to create libraries, components, ports, functions, experiments, etc. and can also run simulations when no graphical output is required. Another way to create a component is to use the graphical tool shown in Fig. 2. On the left hand side of the window is the palette (library) of available components (one icon for each component). The user can identify objects by their corresponding icon and then 'drag-and-drop' them from the palette onto the canvas to create the necessary engine layout. The objects are then connected together through their specific interfaces (ports), depending on the type of information they exchange. The component attributes (data) can be set by clicking on the corresponding icon. Compiling the schematic will create a new (compound) component and automatically generate the respective EL code.

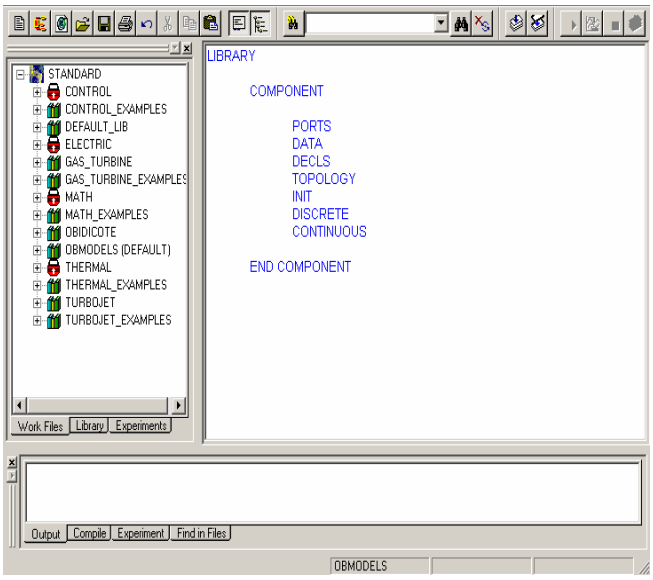

Figure 1: Main Window

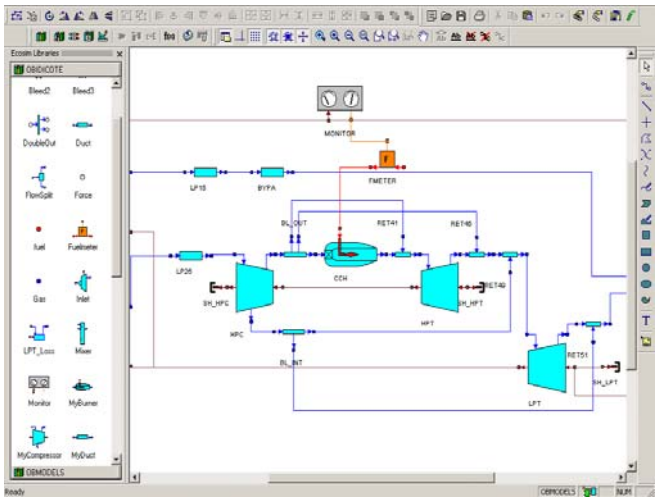

Figure 2: Graphical Component Creation

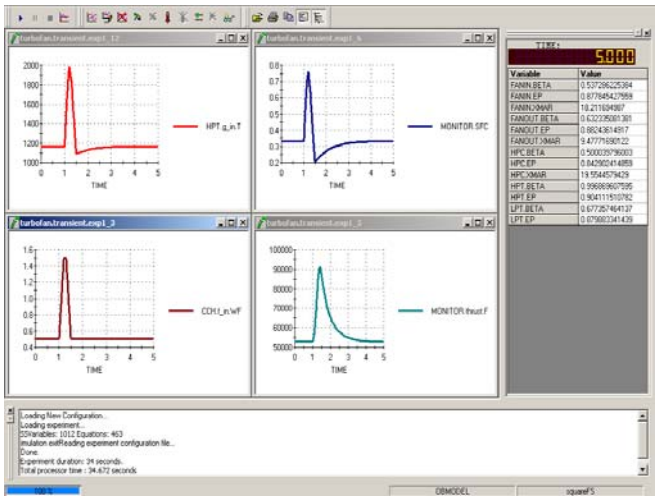

Figure 3: Graphical Output from Simulation

When graphical output is required from a simulation, a separate tool can be launched (Fig. 3). Here the user can create multiple plots, trace variables, calculate steady and transient states and dynamically change data or boundary conditions using sliders. A tool that is not domain-specific, such as this one, is ideal for the simulation of gas turbines, where various disciplines (control, heat transfer, etc) are involved.

MODELLING METHODS

The first step in creating a model of a physical system is to identify the *components* and *ports* to be modelled. Identification of components and ports stems from the nature of the modelled system and the laws that govern its operation.

The modular nature of a gas turbine engine makes this identification a straight forward process. Figure 4 shows the various components and the type of ports (interfaces) of a basic gas turbine library, used in the construction of an engine model. Other components (e.g. propeller, gearbox, heat exchanger, afterburner, generator, etc) can be added to the library to allow modelling any type of gas turbine configuration. The component hierarchy adopted (order of inheritance) aims to reduce code duplication and enhance code re-use of existing and tested components. A key concept in this respect is the use of abstract components; these do not represent physical components but rather hold data and methods that other components inherit from them. It should be noted that a hierarchy tree such as the one shown in Fig. 4 is not static, but it evolves and can be continuously optimised as the library expands, by moving components along the line according to their functionality. This is possible due to the system's flexible architecture that also allows the user to handle the configuration management of the components and the resulting engine models in a transparent and controlled manner.

In the following, the modelling of the ports and components is described and the methods for dynamic modelling are presented. The equations used for modelling the physical processes materialised by the individual components are those described in [19]. Here, the emphasis is put on the way of setting up the various components, exploiting the possibilities offered by the particular programming environment features that are independent from the modelling equations used.

Ports

Ports define the type of information (variables) the components exchange. The direction of a port (in or out of the component) is specified by the user. When a port variable is defined a modifier can be included to describe how that variable behaves when two ports are connected (e.g. a variable can have the same value for all the ports of a connection). Restrictions on how ports can be connected can also be specified (e.g. allow multiple connections). Additionally, EL allows equations to be written inside the ports so that some port variables can be calculated in terms of other port variables. Currently, there are four types of ports in the library, namely Gas, Fuel, Shaft and Force. Table 1 shows the variables and equations for each port.

Table 1: Ports, Variables and Equations

PORT TYPE	VARIABLES	EQUATIONS
GAS	W	WF*(1+FAR+WAR)=FAR*W WW*(1+FAR+WAR)=WAR*W
	P	
	T	
	WF	
	WW	
	FAR	
	WAR	
FUEL	WF	-
	FHV	
SHAFT	PW	TRQ=(PW/XN)*(30/π)
	XN	
	TRQ	
FORCE	F	-

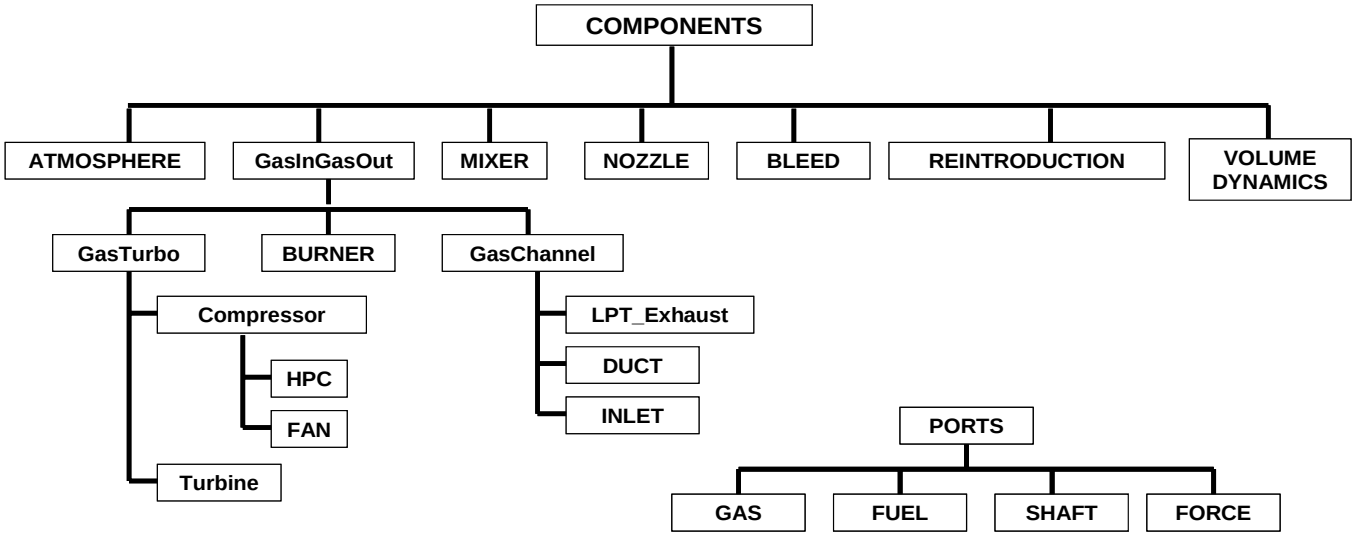

Figure 4: Component Hierarchy

Components

An object-oriented component represents a physical engine component. Common code resulting from similarities in mathematical modelling and/or interface between components is concentrated in abstract components.

The first component used is *ATMOSPHERE*. This has one outlet *Gas* port which is the inlet to the engine. It calculates the values of the port variables for given values of flight altitude (ALT), flight Mach number (XM), relative humidity (RH) and temperature difference (DTAMB) from standard day temperature.

GasInGasOut is an abstract component that has two *Gas* ports; one inlet and one outlet. It holds (as data) the inlet cross-sectional area of the component and defines the inlet corrected flow. It also contains the method for calculating enthalpy from temperature (and vice versa) and static inlet properties from known total ones by calling the appropriate functions.

GasTurbo is another abstract component that inherits the data and methods of *GasInGasOut*. It is the parent of all the turbomachinery components. It has one inlet and one outlet *Shaft* ports. It stores the value of the moment of inertia (J) and includes the method for modelling shaft dynamics (see Dynamic Modelling section).

Abstract component *Compressor* is a child of *GasTurbo* and parent of *HPC* (High Pressure Compressor) and *FAN* components. It calculates the compressor exit parameter values from the inlet ones and the compressor characteristics supplied by the user. For the model described in this paper, the HPC and Fan maps are of a form typical for current civil aero-engines. Separate characteristics are used for Fan Tip and Fan Root performance. The compressor (Fan and HPC) characteristics are supplied in relative form as the ratio of the value of the parameter to a datum value of that parameter. The parameters used are flow, isentropic *e*fficiency and specific work for different values of rotational speed and beta. The data are supplied in the form of 2-dimensional tables. The surge line is defined from data (1-D table) giving the variation of pressure ratio with flow. Reynolds number effects on performance are included by supplying (as a 1-D table) the fractional change in flow and fractional change in isentropic efficiency with Re ratio. For both 1-D and 2-D tables, the user can select the method of table interpolation (linear, spline, cubic). In this model the Fan was considered of sufficiently large size so that Re effects on its performance can be considered small and thus neglected. Component *HPC* can include as many extra outlet Gas ports as there are bleed locations in the actual component it represents. Interstage bleed temperature is based on a fixed $\Delta T_{BLEED} / \Delta T_{COMPRESSOR}$.

Turbine is used to model both the high pressure (HPT) and the low pressure (LPT) turbines of the turbofan model. For the model presented here, the turbines maps also are of a form typical for current civil engines. The relativised parameters used to define turbine performance are: flow and isentropic efficiency for different values of beta and rotational speed. Reynolds number effects on turbine performance are incorporated in a similar manner to *Compressor*.

BURNER (child of *GasInGasOut*) has a *Fuel* inlet port and holds the data for combustion efficiency (assumed constant here), fuel heating value and pressure loss coefficient, K. A cold pressure loss, ΔP, is calculated from:

$$\Delta P / P = K * Mach^2 \quad (1)$$

where Mach is obtained from the cross sectional area already defined in *GasInGasOut*.

The last child of *GasInGasOut* is *GasChannel* where conservation of mass is defined. This component has three children, namely *DUCT*, *INLET* and *LPT_Exhaust*. *DUCT* is used to model the pressure loss in the Swan Neck (between Fan Root and HPC) and the By-pass duct (between the Fan Tip delivery and the Mixer Entry). The pressure loss, ΔP, is calculated as in the *BURNER* case (Eq. 1). *LPT_Exhaust* calculates a pressure loss based on axial Mach number and swirl angle. The mean blade diameter, blade exit angle and loss coefficient are the data values used in the calculations. Finally, *INLET* provides a ram pressure recovery option and calculates the value of the intake momentum drag which is then fed to an outlet *Force* port.

MIXER has two inlet and three outlet *Gas* ports. The two inlet ports are the Core and By-pass streams and the three outlet ports are the Mixed, Hot and Cold streams generated from the Core and By-pass streams. The Hot and Cold streams are a proportion of the Core and By-pass streams respectively and the mixed stream is calculated so that continuity is maintained in flow, energy and momentum flux. The total pressures of the Hot and Cold streams are assumed to remain constant through the partial mixing process. The condition for engine matching is that the static pressures of the By-pass and Core streams are equal. The Core and By-pass mixing areas as well as the mixing efficiency factors are set by the user.

NOZZLE has three inlet *Gas* ports (Mixed, Hot and Cold) and one outlet *Force* port. The value of the nozzle exit area is defined by the user. The overall ideal thrust and the effective nozzle area for the three streams are calculated by exhausting each stream separately to ambient conditions and adding the three values obtained. The value of the thrust is fed to the *Force* port. Overall thrust and discharge coefficients are obtained from tables depending on the exit pressure ratio.

Apart from the interstage bleeds defined in *Compressor*, a *BLEED* component has also been created with one inlet and one outlet *Gas* ports plus a user specified number of *Gas* ports for bleeding air. The mass flow in these ports is expressed as a fraction of the inlet flow. The air is introduced back to the system using *REINTRODUCTION* (RET). This component also has an inlet and an outlet *Gas* ports and a user specified number of bleed return *Gas* ports. The outlet temperature is calculated from a heat balance. The mainstream pressure is assumed unchanged.

The gas turbine library also contains a number of user-defined functions for calculating the thermodynamic properties of the working fluid (air or gas) as a function of temperature and Fuel-Air Ratio (FAR) according to [20].

Dynamic Modelling

For an engine performance program to produce representative results during transient operation, it must take into account shaft dynamics, heat transfer between gas and metal and gas dynamics.

Shaft dynamics are incorporated in *GasTurbo* through the following equation:

$$\Delta PW = J * XN * \frac{dXN}{dt} * \left(\frac{2\pi}{60}\right)^2 \quad (2)$$

Where ΔPW is the excess shaft power.

The components making up the Gas Generator (i.e. *HPC*, *BURNER* and *Turbine*) include a method to account for heat transfer between the gas and the metal through the following equation:

$$c_p * M * (dT_m/dt) = h * A_s * (T_g - T_m) \quad (3)$$

The convective heat transfer coefficient h is modelled using an expression for forced convection in a duct [21]:

$$h = C * Re^{0.8} * k / L \quad (4)$$

where the Reynolds number Re is defined as:

$$Re = (W_g * L)/(\mu * A) \quad (5)$$

The heat transfer equation is applied separately to compressor and turbine blades and casings. The user can switch this calculation ON or OFF from within the component.

Finally, a component was created (*VOLUME DYNAMICS*) to model the dynamics of the volumes representing the individual engine components, through the following conservation equations [22]:

Conservation of Mass:

$$\frac{d\rho}{dt} = \frac{W_{in} - W_{out}}{Volume} \quad (6)$$

where in and out refer to the inlet and outlet of the volume being considered.

Conservation of Momentum:

$$\frac{dw}{dt} = \frac{1}{L} * \left[(W * V + p * A)_{in} - (W * V + p * A)_{out} + F_{body}\right] \quad (7)$$

where F_{body} is the force exerted by solid boundaries in the volume.

Conservation of Energy:

$$\frac{d}{dt}(\rho * H - p) = \left[(W * H)_{in} - (W * H)_{out} + PW\right]/Volume \quad (8)$$

where PW is shaft or thermal power exchanged with the volume.

VOLUME DYNAMICS has two inlet Gas ports (the inlet and outlet of the component being modelled) and one outlet Gas port (see HPC volume dynamics in Fig. 5). The geometric data of the engine component (length, volume and cross sectional areas) are set by the user. A switch can activate the component ON/OFF.

APPLICATION EXAMPLE, RESULTS & DISCUSSION

Using the components from the Gas Turbine Library created, the turbofan model depicted in Fig. 5 was produced. The mathematical modelling of the components presented in the previous section is based on the Real Time Model (RTM) reported in [19]. The RTM is used for steady state and transient performance and is considered to represent current modelling practices for a typical civil two-spool turbofan engine. It does not refer to a specific commercial turbofan engine in service, but its structure reflects fully the principles and the technology of contemporary turbofan engines in civil aviation. In comparison with the Object-Oriented Model (OOM) described here, it does not include gas dynamics effects. Additionally, the two models use different routines for

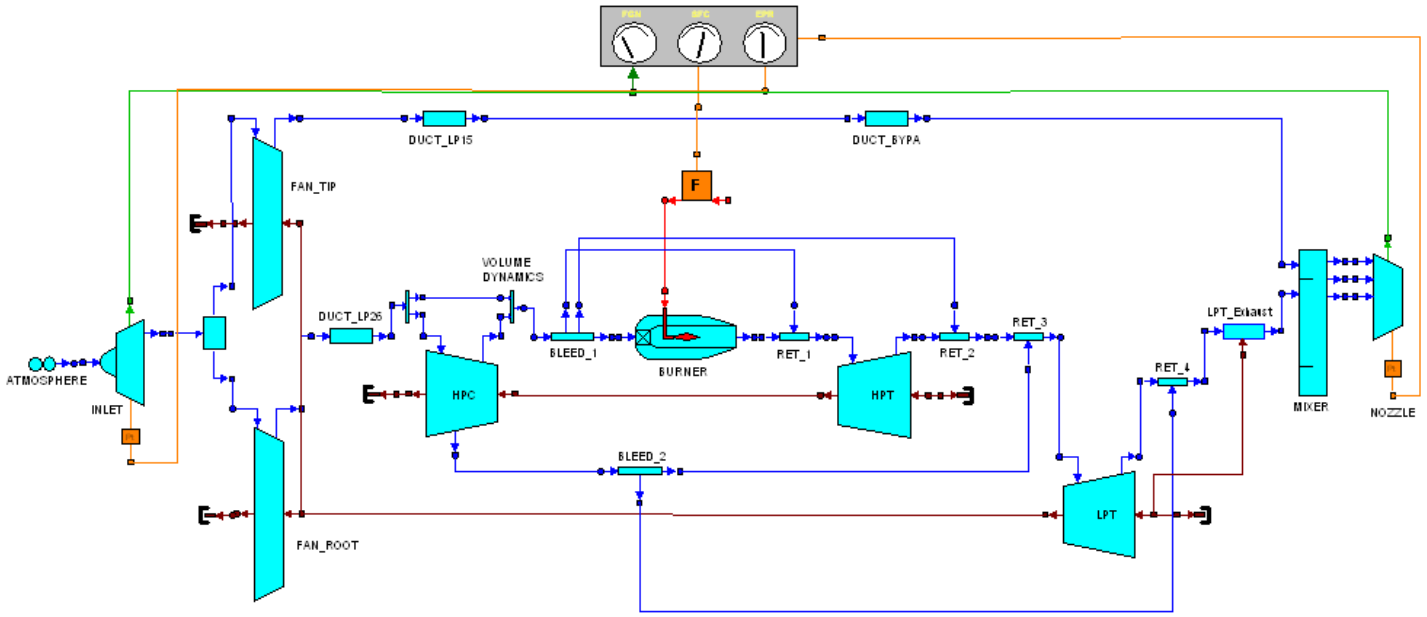

Figure 5: Turbofan Model Layout

calculating the thermodynamic properties of the working fluid and for table interpolation. Finally, different solvers are employed in the two models. With these differences in mind, a number of steady state and transient cases were simulated with both models. The results are compared to ensure that the OOM behaves consistently.

Figure 6 shows the percentage difference between the two models for a number of parameters and for two steady state cases: a) at Top-of-Climb (TOC) and b) at Seal-Level-Static (SLS). The engine's operating point is defined by specifying in the *Experiment* the values of ALT, XM, RH, DTAMB (standard boundary conditions) and anyone of the following parameters: fuel mass flow rate, thrust, shaft speed, engine pressure ratio, turbine inlet temperature, compressor delivery pressure or temperature. For the TOC case (design case), the difference between the values predicted by the two models is within ±0.1% except for the Thrust (FGN) for which the difference is 0.4%. For the SLS case, all the values are within ±0.15%.

A series of steady state simulations was also carried out at SLS conditions for different values of WF. The results, as a series of points on the high pressure compressor map are shown in Figure 7. For the whole range tested, there is less that 0.1% difference between the two models.

Having established that the OOM performs satisfactory at steady state, a number of transient cases were simulated. Figure 8 compares the predictions of the two models for two transient cases: a) acceleration and b) deceleration. For both cases, the fuel flow rate is the controlling parameter (its variation with time is depicted in the insert on the upper left hand side of the figure). Only shaft dynamics were taken into account in the simulations. Both models behave similarly over the entire range tested.

In terms of speed of execution of OOM, a steady state case takes less than 50 ms to compute using a PC with a P4 processor at 1.8 GHz with 512 Mb RAM, under Windows XP. For the acceleration case, the duration of the simulation was 112s for a time step of 20 ms. For both steady and transient cases, there were 1012 variables, 463 equations and the accuracy was set to 10^{-6}.

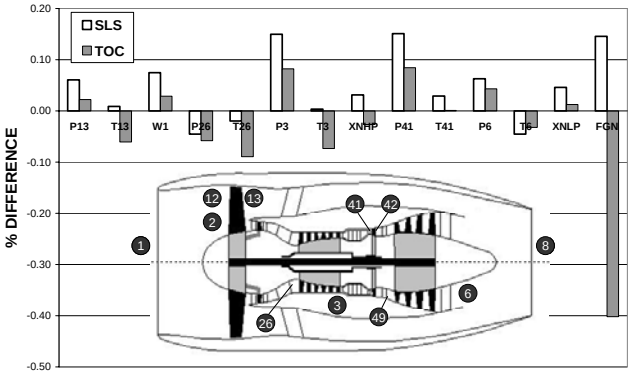

Figure 6: Model Comparison for Static-Sea-Level (SLS) and Top-Of-Climb (TOC) Cases

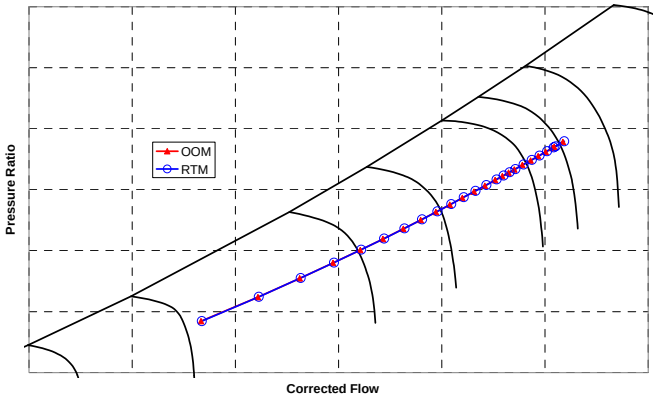

Figure 7: Model Comparison for a Series of Steady State Cases at Static, Sea-Level Conditions

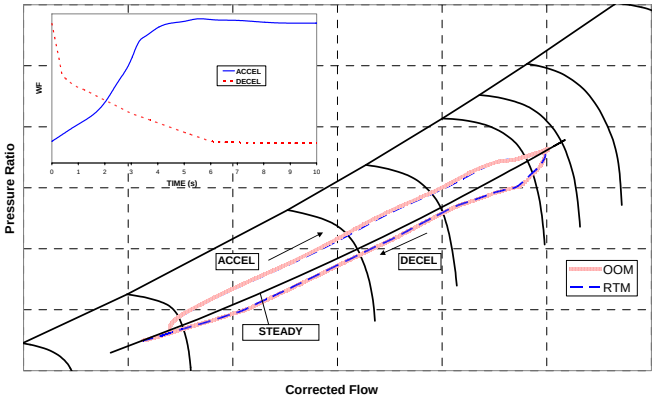

Figure 8: Model Comparison for Deceleration and Acceleration Cases

Engine Dynamics

If the dynamic terms are not included in an engine cycle model then its predictions will differ from the true engine behaviour under transient conditions. This will lead to incorrect assumptions regarding stable operation, fault diagnosis, controller design, etc. The OOM is used to illustrate the point.

Figure 9 shows the temporal variation of thrust when various dynamic effects are taken into account in the OOM model. This is for the acceleration case described in the previous figure. For this simulation, the difference in engine thrust can be up to 6% when all dynamic effects are taken into account compared to the case with only shaft dynamics.

In the case of gross transients such as fuel-spiking, gas dynamics effects are very important. Such a case was simulated with the OOM and the results are shown in Fig.10. Inclusion of gas dynamics in the model has a profound effect on the compressor surge margin, increasing it by almost 10% during the acceleration phase, compared to the simulation where gas dynamics were not considered.

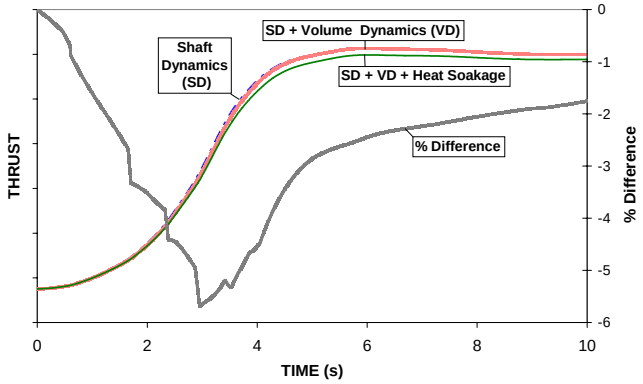

Figure 9: The Effects of Shaft, Gas and Heat Transfer Dynamics on Thrust for an Acceleration

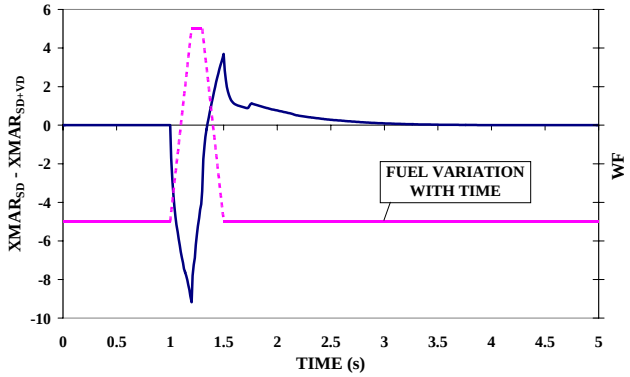

Figure 10: The Effect of Gas Dynamics on Compressor Surge Margin during a Fuel Spike

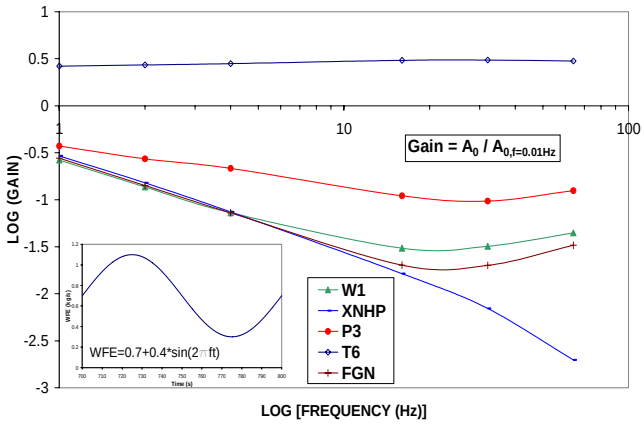

Figure 11: Gain Variation with Frequency for Sinusoidal Fuel Input

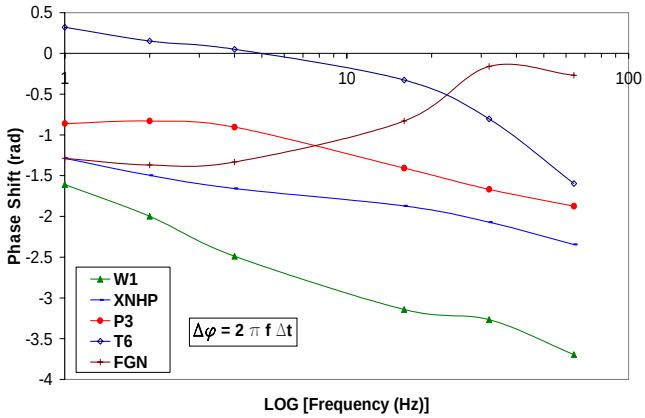

Figure 12: Phase Shift Variation with Frequency for Sinusoidal Fuel Input

Frequency Response

The OOM was also used to study the frequency response of a turbofan engine. No thermal effects were included in these simulations. The frequency of a sinusoidal fuel schedule was varied between 0.01 and 64 Hz. The time length of the input (fuel) signal was chosen long enough to achieve stabilised sinusoidal output. Figure 11 shows the frequency variation of gain for various parameters while Fig. 12 shows the phase shift, $\Delta\varphi$, for the same parameters.

For the case when only shaft dynamics are considered (not shown here), the gain of pressures and temperatures initially reduces with increasing frequency up to about 20 Hz where it starts to level off. The gain of spool speeds continue to decrease with increasing frequency for the whole frequency range tested. When volume dynamics are also considered, there is a change of slope at higher values of frequency (>20 Hz). This is expected as gas dynamic effects become significant at higher frequency ranges. It is interesting to note that the gain of the temperatures after the combustion chamber is positive. In general, there is no consistent variation of the phase difference between input and output signals with frequency.

Adaptation to Test Data

Both *Compressor* and *Turbine* components (Fan Tip, Fan Root, HPC, HPT and LPT in Turbofan model) incorporate adaptation factors (SW and SE). These factors are expressing flow and efficiency variations in these components.

An engine in reference condition is characterised by values of the factors equal to unity. The values of the factors for an engine in condition which has to be determined are derived using data from measurements. The incorporation of these factors is useful for customizing an engine as well as for condition monitoring and diagnostic purposes (for more details of adaptation factors see [23]).

In the OOM, when the values of the adaptation factors are known these are supplied as data from the user for each component. When the factors are to be determined from measured quantities then a special *Partition* (Design) is created where they are declared as unknowns (Fig. 13). The program then asks (Fig. 14) for an equal number of boundary conditions (to form an NxN system); here the user can specify the parameters corresponding to the measured quantities. An *Experiment* is then created for this *Partition* to find the values of the adaptation factors for the specified boundary conditions. Figure 15 shows the values of the adaptation factors estimated

by the OOM from measurements produced by the model of a deteriorated engine.

As seen from this description, the adaptivity feature is very easy to implement as a result of the inherent capabilities of the object-oriented environment. This approach offers thus more flexibility, compared to previously presented methods [24, 25].

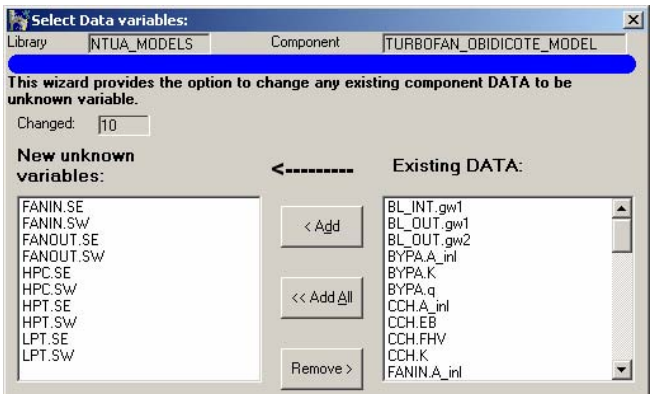

Figure 13: Changing Adaptation Factors from Data to Unknown Variables

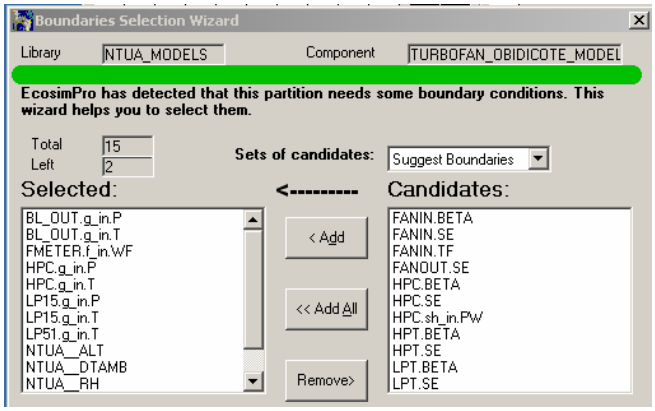

Figure 14: Selecting Measured Quantities as Boundary Conditions

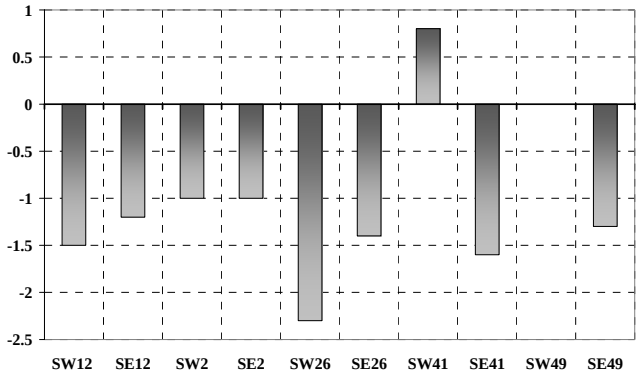

Figure 15: OOM Estimated Values of Adaptation Factors from Measurements Simulating a Deteriorated Engine

CONCLUSIONS

The use of an object-oriented simulation environment to create a re-usable library of gas turbine engine components has been presented. A representative application consisting of an aero-thermodynamic model of a typical civil two-spool turbofan engine was developed, by connecting the appropriate components from the library through an advanced graphical user interface. Steady state and transient simulations results were shown to agree well with those produced by an industry-accepted model. It was demonstrated that the various types of dynamic effects can be easily incorporated into the model and sample results of a related sensitivity study were presented. This feature was also used to derive the frequency response of the engine, in terms of performance variables over a fuel input. Finally, the ease of implementation of adaptation factors (for matching a model to given measurements) was demonstrated.

The flexibility and speed of developing and re-using component/models, the powerful mathematical and debugging facilities along with an advanced graphical user interface make this environment a valuable tool for gas turbine system performance analysis.

ACKNOWLEDGEMENTS

The work described in this paper has been carried out as part of the integrated project VIVACE (AIP3-CT-2003-502917) and financial support of the European Union Commission is gratefully acknowledged. The authors would like also to express their thanks to the partners involved in Work Package 2.4 of VIVACE for their comments and discussions and especially EA International for supplying the software and providing technical support. The assistance of Ph. Kamboukos as well as C. Romesis and G. Sieros in parts of the work is also acknowledged.

REFERENCES

[1] Research and Technology Organisation, 2002, "Performance Prediction and Simulation of Gas Turbine Engine Operation", RTO-TR-044.

[2] Mathioudakis, K., Stamatis, A., Tsalavoutas, A., Aretakis, N., 2000, "Instructing the Principles of Gas Turbine Performance Monitoring and Diagnostics by Means of Interactive Computer Models" ASME 2000-GT-584.

[3] Booch, G., 1994, "Object-Oriented Analysis and Design with Applications", 2nd edition, Benjamin Cummings, Redwood City.

[4] Kurzke, J., "Advanced User-friendly Gas Turbine Performance Calculations on a Personal Computer", ASME 95-GT-147.

[5] GasTurb website: www.gasturb.de

[6] Visser, W.P.J., and Broomhead, M.J., 2002, "GSP, A Generic Object-Oriented Gas Turbine Simulation Environment", ASME 2000-GT-2.

[7] GSP website: www.gspteam.com

[8] Reed, J. A. and Afjeh, A.A., "Computational Simulation of Gas Turbines: Part I - Foundations of Component based Models", ASME 99-GT-346.

[9] Reed, J. A. and Afjeh, A.A., "Computational Simulation of Gas Turbines: Part II - Extensible Domain Frameworks", ASME 99-GT-347.

[10] Lytle, J.K., 2001, "The Numerical Propulsion System Simulation: An Advanced Engineering Tool for Airbreathing Engines", ISABE 2001-1216.
[11] Follen, G.J., 2002, "An Object-Oriented Extensible Architecture for Affordable Aerospace Propulsion Systems", RTO-MP-089.
[12] Math Works, 1997, "SIMULINK: Dynamic System Simulation for MATLAB, Ver. 2", The Math Works Inc., USA.
[13] Visser, W.P.J., Broomhead, M.J. and van der Vorst, J., 2001, "TERTS, A Generic Real-Time Gas Turbine Simulation Environment", ASME 2001-GT-446.
[14] Kong, C. and Roh, H., 2002, "Performance Simulation of Turboprop Engine Using Simulink Model", ASME 2002-GT-30516.
[15] Camporeale, S.M., Fortunato, B and Mastrovito, M., 2002, "A High-Fidelity Real-Time Simulation Code of Gas Turbine Dynamics for control Applications", ASME 2002-GT-30039.
[16] E.A. International, 2004, "EcosimPro: User Manual".
[17] EcosimPro Website: www.ecosimpro.com
[18] Petzold, L.R., 1983, "A Description of DASSL: A Differential/Algebraic System Solver", Scientific Computing, pp. 65-68.
[19] Stamatis, A., Mathioudakis, K., Ruiz, J. And Curnock, B., 2001, "Real Time Engine Model Implementation for Adaptive Control & Performance Monitoring of Large Civil Turbofans", ASME 2001-GT-362.
[20] Walsh, P.P. and Fletcher, p., 1998, "Gas Turbine Performance", Blackwell Science, Oxford.
[21] Long, C.A., 1999, "Essential Heat Transfer", Longman, London.
[22] Biraud, B., Despierre, A., Gayraud, S., 2001, "Simulation of the WR-21 Advanced Cycle Engine", ASME 2001-GT-0020.
[23] Stamatis A., Mathioudakis K., Papailiou K., 1990, "Adaptive Simulation of Gas Turbine Performance", J. of Engineering for Gas Turbines and Power, **112**, pp. 168-175.
[24] Stamatis A., Kamboukos Ph., Aretakis N., Mathioudakis K., 2002, "On Board Adaptive Models: A General Framework and Implementation Aspects", ASME 2002-GT-30622.
[25] Visser, W.P.J., Kogenhop, O. Oostveen, M., 2004, "A Generic Approach for Gas Turbine Adaptive Modelling", ASME 2004-GT-53721.