COBEM-2017-0019
AN OBJECT-ORIENTED PLATFORM FOR HIGH-FIDELITY MODELING OF POWER PLANTS

Cesar Celis
Mechanical Engineering Section, Pontificia Universidad Católica del Perú
Av. Universitaria 1801, San Miguel, Lima 32, Lima, Peru
ccelis@pucp.edu.pe

Sandro Barros Ferreira
GT2 Energia
R. Hélio de Almeida, s/n, Sala 38, Cidade Universitária UFRJ, Rio de Janeiro, RJ 21941-614, Brazil
sandro@gt2.com.br

Abstract. *This work describes the development of an object-oriented programming (OOP) based platform suitable for building up full scope (full scale or full replica) power plant simulators. These simulators incorporate high-fidelity computer models of whole power plants, including their control room along with all consoles, switches, keys, indicators and other human-machine interfaces. More specifically, details of the rationale behind the conception, design and implementation of the referred OOP-based platform are initially described. An especial emphasis is put on several architectural aspects accounted for and object class hierarchies defined during the development of the platform. Next some the main modules composing the platform are described in detail. One of these modules, known as Simulation module, involves the modelling of the actual power plant components (hardware). Details are thus provided of the component-based design of this Simulation module, along with the inter-component communication approach utilized in the platform under discussion. It is worth noticing that this inter-component communication is key in the platform developed, as this platform is intended to be as generic as possible, such as to allow the construction of any power plant simulator. Finally, as a means of illustration, the use of the platform for creating a design point, off-design point and transient model of an industrial gas turbine, part of an actual full scope power plant simulator, is discussed. This OOP-based tool is expected to be used in future for building up high-fidelity simulators providing practical training to power plant personnel.*

Keywords*: Object-oriented programming, High-fidelity modelling, Teaching and training, Full scope simulators, Power plants*

1. INTRODUCTION

The pursuit of a balanced and sustainable energy matrix has forced governments to promote the diversification of energy primary sources. Countries that had most of their sources based on fossil fuels, nowadays invest in cleaner and renewable sources. Others, like Brazil, which is strongly based on hydroelectric sources, seek energy security by diversifying their energy matrix through the introduction of more wind and thermal power plants (EPE, 2013). The operation of thermal power plants demands a big intervention by the operators, and due to the history of hydro generation in Brazil, few thermal power plant operators have the necessary knowhow to properly operate and train new plant operators. There are two ways of training staff for this kind of job, one is by following senior operators and watching their work, which demands a considerable amount of time, can represent economic losses and does not ensure that trained operators will be exposed to all maneuvers required. The second way is to use technology tools, like full scope power plant simulators (Automation.com, 2012).

Experience shows that an efficient learning process can only be properly achieved, in short term, using simulators to train operators (Krost *et al.*, 2000). The use of simulators for training nuclear power plants operators is already consolidated due to security and availability issues (Bindu *et al.*, 2011). The high cost and long duration of conventional training methods have made the use of simulators essential in both fossil-fuel and biomass-fired power generation units. However, the type of simulator used to provide this training varies widely according to its application; it will depend on the company needs, the level of detail and the fidelity of training. According to the standard that establishes the functional requirements for fossil-fuel power plants there are three types of simulators (ISA, 2005): (i) full scope training simulator - FSTS; (ii) reduced-scope training simulator and (iii) generic simulator. For instance, there may be cases that the operators just need to learn some procedures on the power plant or just get acquainted to a few equipment, so the reduced scope simulator can fit their needs. These three types of systems are based on complex mathematical models that represent the essential physical processes that occur in the plant. And the more faithful the system is, the more complex will be its control systems and screens and the more detailed will be the processes. This complexity explains the small number of full-scope simulators developers worldwide.

A full scope simulator is a high-realism simulator that intends to reproduce the power plant control room, and its results must be identical in time and indication to the responses received in the actual plant under similar conditions (ISA, 2005). Operators can both experience effective training in this type of simulator and test maneuvers before being executed on the actual power plant. The major drawback of a full scope simulator is its cost and long lead time.

2. DEVELOPMENT CONTEXT

Until the year 2000 Brazil had an electricity matrix based on hydropower plants. Thermal power plants existed to supply remote areas, like the northern Amazon region, and some isolated areas that did not justify the installation of very long transmission lines. A severe draught in that year, associated with natural gas availability from neighbor countries like Bolivia and Argentina, supported the launching of the Thermal Power Plants Priority Program, called PPT by the Federal Government. The PPT depicted a scenario in which forty natural gas based power plants were to be constructed and operated. Private money was to be injected in the program in partnership with Petrobras, the national oil company. In total, PPT would add 15,398 MW to the existing system.

The program envisaged both simple and combined cycle power plants. Many power plants were not built, but the ones currently running had to learn all what they need on the fly, as the country need more power availability. The training, though supported by many universities, was more theoretical and the only full scope simulator purchased for the endurance does not represent any of the built power plants. As the machinery is aging and more thermal power plants are to be built in the country, an opportunity to develop local knowledge and technology on the production of full scope simulators has emerged. Thus, this work describes one of the results of this effort. After successfully building the first specific full scope simulator for an 820 MW combined cycle power plant, there is now a work set to create an object-oriented platform to allow the construction of as many as necessary full scope simulators for any power plant that desires to own one.

3. OOP-BASED PLATFORM DEVELOPMENT

The development of an OOP-based platform suitable for building up full scope (full scale or full replica) power plant simulators is described in this section. More specifically, its requirements along with details of the rationale behind its conception, design and implementation are initially highlighted. Emphasis is put on several architectural aspects accounted for and object class hierarchies defined during the development of the platform. Next, one of the main modules composing the platform, i.e., Simulation module, is described next in detail. This module is responsible for modeling the actual power plant components (hardware). Details are thus provided of the Simulation module component-based design, along with the inter-component communication approach used in the platform developed.

3.1. Requirements and Overview

As usually occurs when developing new models or computational tools in general, the first step in the development of the OOP-based platform involved the specification of its main requirements. In the case of the platform discussed here, these requirements include the following: (i) The platform and its associated computer code should present reusability, extensibility and ease of maintenance features. (ii) The computer code included in the platform should be fully compilable; i.e., it should be possible to convert the source code into computer-readable machine code that can be executed directly by a computer's central processing unit (CPU). (iii) The platform should allow in future the creation of computer models, representative of power plant components, through a GUI (graphical user interface).

From the first requirement, it was decided to implement the platform following the OOP paradigm. OOP was chosen because it presents, through classical concepts such as abstraction, encapsulation, inheritance and polymorphism (Parsons, 2000), intrinsic characteristics of ease of maintenance and extension, as well as code reusability. From the second requirement, since this programming language is fully compilable, C++ was chosen as the main programming language. This choice is also supported by the natural affinity that exists between C++ and OOP. Finally, after the third requirement and following previous works (Takai, 1990; Curlett and Felder, 1995), the OOP-based platform was conceived and designed. A schematic representation of the designed platform including its main modules is shown in Figure 1. As it can be observed from this figure, the platform includes three main modules, Simulation Data and GUI. Briefly, when using the OOP-based platform, the module Simulation performs all required simulations (calculations). The results from such simulations are stored in the module Data, from which they are used in input/output (I/O) procedures, as well as for graphic display via the GUI module. Following this platform design, the communication between the Simulation and GUI modules is made only through the Data module, which includes data shared by the other two modules. This allows that the development of the Simulation module is completely independent of that associated with the GUI module, and vice versa. The first two modules, Simulation and Data, are discussed in this work. The development of the third module (GUI) corresponds to future work.

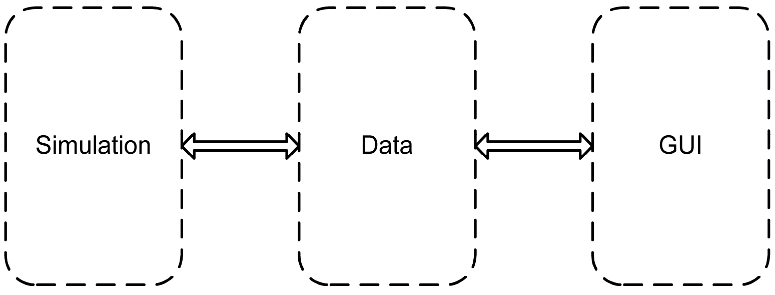

Figure 1. Main modules of OOP-based platform.

Accounting for that power plants may be conceived as sets of interconnected components (hardware specific parts), the Simulation module is based on models of components and their interconnections. Following (Curlett and Felder, 1995) therefore, this module includes three basic concepts: (i) Component, (ii) Port and (iii) Connector. The idea here is that any system, power plants for instance, can be represented by a set of components connected to each other. The interconnection between components (Component object) is carried out through connectors (Connector object) connecting the ports (Port object) included in the components. The ports, which may be of different types depending on the specific application, contain standard information of interest to the other components. That is, the relevant information of each component that needs to be externalized is contained in the ports of the components. Per this approach, the model of a given system, a gas turbine for instance, is constructed using connectors that connect the ports of the different components that comprise the system. It is worth noticing that the connectors are not hard coded within the components. The specification of the connections between the components of a system is currently performed during the assembly of the system and this will be done through the GUI module in future. This component-port-connector approach does not involve the use of global variables and so it can be extended indefinitely.

The Data module is in turn responsible for both data management and I/O procedures. The communication between the two modules Simulation and GUI is performed via this Data module. The Data module implementation is based on what is known in computer science as dictionaries. More specifically, associative containers, storing elements defined by the pair (key, value) are used for storage and efficient retrieval of the main parameters characterizing individual components. Each of the component ports includes a container whose elements are variables that represent particular attributes of the simulation. Only numeric variables are currently used as elements of the containers. Other types of variables will be however considered in future, via the use of variable tables (Curlett and Gould, 1994) fulfilling the same role of the containers currently utilized. Notice that, in addition to allowing data cataloging and I/O procedures, the use of variable containers also provides an efficient mechanism for data transfer between components, through copy functions between variables. This is how the connector functions of data push/pull are implemented on the OOP-based platform. Some additional details regarding the use of the Simulation and Data modules are provided in the case study discussed in last part of this work. The class hierarchy used for modeling a system, implemented in the Simulation module, is discussed below.

3.2. Simulation Module Class Hierarchy

As its name implies, the Simulation module is responsible for the simulations in the platform. In this context, a simulation is understood as the execution of the computational model representing a given system. That is, the resolution of the governing equations described in the system model. When the models of the components comprising a given system are mutually independent, the resolution of these equations is made internally in each of the models. Nevertheless, when there are dependencies between the models of the system components, specific tools (solvers) for solving the dependent models governing equations need to be utilized. In the initial development of the OOP-based platform, only independent models have been considered. Solvers will be introduced in future in the platform after their need. Models of independent components and their interconnections are described in this section.

As stated before, for modeling a system, the Simulation module involves in general three types of objects, component, port and connector. Following this approach, the model of any system is comprised of components (component) connected (connector) to each other through their ports (port). The main class hierarchy associated with these three main objects, component, port and connector, are illustrated in Figure 2 using a UML (unified modeling language) diagram. As it can be seen in this figure, the classes involving these objects (SiComponent, SiPort and SiConnector) are derived from the same generic class (SiPart). The generic class has no physical meaning and it is used here to include information and procedures common to all classes derived from it only.

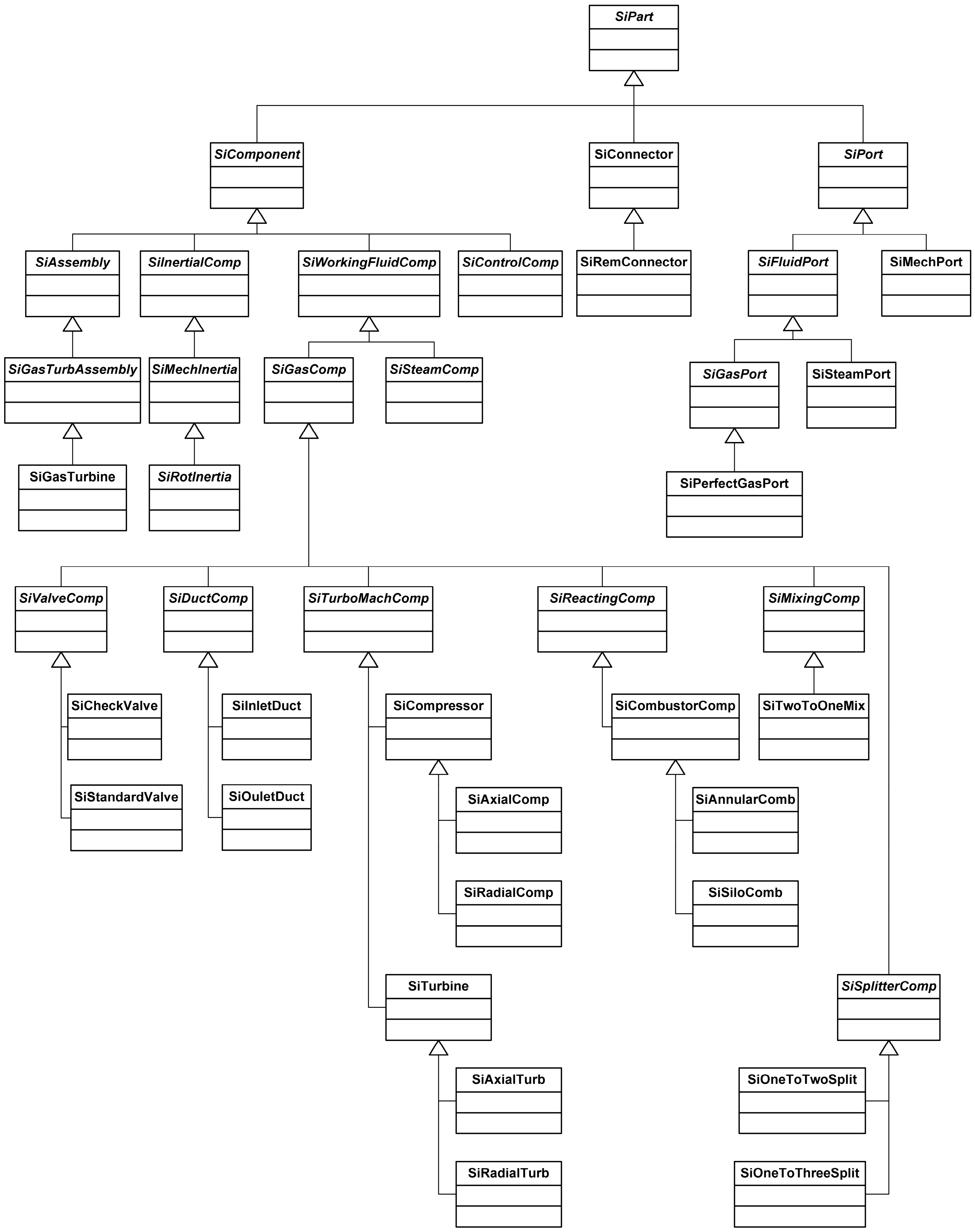

Figure 2. Class hierarchy of OOP-based platform Simulation module.

The SiConnector class allows the communication between two system components by passing the information contained at their ports. This class checks the compatibility of the ports connected by the connector by counting the number of variables communicated through the ports. If the number of exchanged variables is zero, an incompatibility of ports is declared and an error message is produced. The SiRemConnector class is similar to the SiConnector one, but this allows the communication between different computer processors through the MPI (message passing interface) protocol. This class will be useful in the future when using distributed computer systems for executing the different computational models utilized.

The SiPort class is the base class used to define the various types of ports contained in the system components. This class allows the flow of information in the components. There is no limit regarding the number of ports that a component can have. This number is set according to the needs of each component. In a way, components may be regarded as control volumes and ports as control surfaces delimiting these volumes (Curlett and Felder, 1995). Two main port types have been initially implemented, SiFluidPort and SiMechPort. These ports allow the connection between components through the fluid flow (SiFluidPort) and mechanically (SiMechPort). The fluid-related ports have been in turn divided into those of gas type and those associated with pure substances. Additional subdivisions of port types can be performed when required.

Finally, the SiComponent class is used to model the different components of a system, a power plant for instance. In this context components are understood in general as being specific parts of a system hardware. There are no restrictions however as to what a component does or what it is exactly. Components can be then, for example, an annular combustor in an aero gas turbine engine or a condenser in a power plant. This generic nature of the components in the OOP-based platform allows their reuse in different system models and simulations. It is also worth noticing that there are no restrictions in terms of the internal structure of the components. That is, the model inside the components can be as simple as multiplying two parameters characterizing the component; or as highly complex as solving the 3D Navier-Stokes equations using a CFD (computational fluid dynamics) numerical approach. These features of the OOP-based platform would allow eventually carrying out numerical zooming-related analyses (Follen and auBuchon, 2000), increasing in this way the fidelity level of the virtual tests performed using this platform.

The structure of the classes derived from the base class SiComponent is based on previous works developed in other contexts (Curlett and Felder, 1995; Visser and Broomhead, 2000). Accordingly, three classes are initially derived from the SiComponent base class, SiWorkingFluidComp, SiInertialComp and SiControlComp. From these classes, the first one (SiWorkingFluidComp) involves components through which there is the passage of working fluid. The second one (SiInertialComp) is related to inertial components, such as those involving thermal (e.g. engine casings) and mechanical inertia (e.g. shafts). Finally, the third one (SiControlComp) includes components related to the system control such as actuators. Two other classes are derived from the SiWorkingFluidComp class, SiGasComp and SiPureSubsComp. As their names indicate, these last two classes regard to cases in which the working fluid is a gas and a pure substance, respectively. The case study discussed in this work, as a means of illustration of usage of the OOP-based platform, focus on gas turbines, i.e., components whose working fluid is a gas. Therefore, in accordance with Figure 2, the description of the class hierarchy will be restricted in this section to these specific component classes.

Six main component classes whose working fluid is a gas, i.e., derived from the SiGasComp base class, have been firstly implemented. Following the order in which they are shown in Figure 2, these component classes are: (i) SiValveComp, (ii) SiDuctComp, (iii) SiTurbMachComp, (iv) SiReactingComp, (v) SiMixingComp and (vi) SiSplitterComp. From these classes, the first two refer to, respectively, valves and ducts through which the gases flow. The third class includes turbomachinery related components. Compressors and expanders (turbines), which can be of axial or radial type, have thus been initially implemented in this classes' category. The fourth class includes components involving chemical reactions. Only combustors, which can be of annular or silo type have been initially implemented in the platform. Other components with chemical reactions will be included in future according to their need. The last two classes, SiMixingComp and SiSplitterComp, are related to components involving gases mixture and split flow processes, respectively.

Following a composite design pattern, the simulation module class hierarchy includes as well component classes that hold other components. In Figure 2 this type of classes are represented by the SiAssembly base class, from which other classes such as the SiGasTurbAssembly and SiGasTurbine are derived. Notice that like previous works (Curlett and Gould, 1994) no assumptions are made on the type of components contained in an assembly. Accordingly, since SiAssembly is derived from SiComponent, an assembly may be part of another assembly, allowing thus the use of as many subassemblies as required. Finally, it is important to highlight that, for the sake of conciseness, only the main classes required for modeling gas turbines have been shown in Figure 2. Other classes thus, such as those derived from the SiGasComp base class related to nozzles and heat exchangers, do not appear explicitly in this figure.

4. CASE STUDY: OOP-BASED PLATFORM USAGE

The focus of this work is on the OOP-based platform development and not on the results obtained using such platform. Accordingly when discussing the case study accounted for in this section, an emphasis is put on the platform objects used for the system modeling rather than on the results obtained from the simulations. To do so a model of an Alstom's GT11N2-like industrial gas turbine engine has been created using the OOP-based platform and the details of such modeling are addressed here. For the sake of completeness however, design and off design point results of such engine modeling are briefly discussed as well.

4.1. General Description

As stated above a model of an Alstom's GT11N2-like engine was created using the platform under discussion in this work. This modeled gas turbine engine features a 14-stage compressor, a silo type combustor and a 4-stage turbine, along with several cooling/sealing bleeds and blow-off valves related flows. A schematic representation of the created model is shown in Figure 3. All connections between engine components indicated in this figure imply the use of platform objects of type connector. These object connectors 'connect' the modeled components through their ports. Notice that connections between fluid ports only have been included in this figure. To avoid polluting this figure, connections through mechanical ports, between compressor-turbine-shaft for instance, do not appear in this figure. What has been included in Figure 3 should be enough for given the reader a flavor of the models that can be created using the developed OOP-based platform.

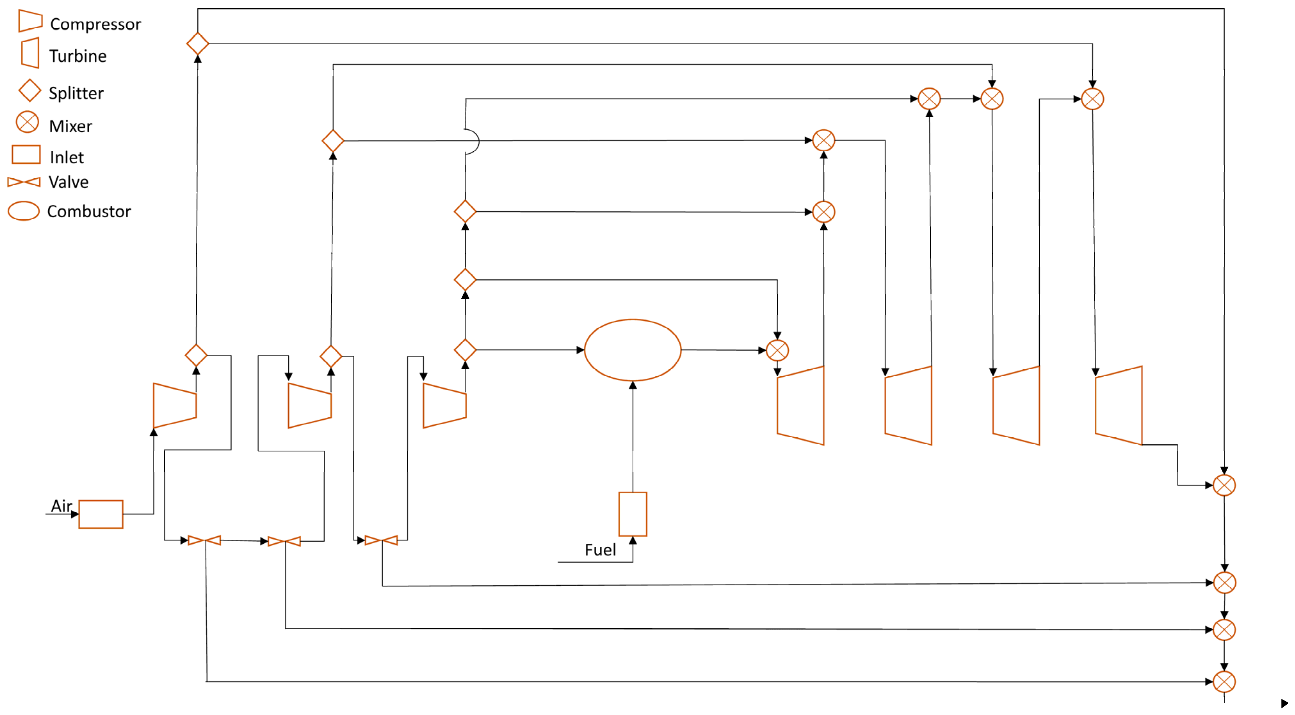

Figure 3. Gas turbine scheme including cooling bleeds and blows-off valves related flows.

It is first noticed from Figure 3 that two object inlets are utilized, one for the air entering and the other for the fuel. The compressor is modeled through three compressor sections indirectly interconnected each other via splitters and blow-off valves objects. It is worth highlighting that the modeled compressor blow-off system works during the engine start-up/shut-down only. During normal operation then, e.g., steady state operation at a given load, the compressor blow-off valves are fully closed. The compressor has been divided in sections such as to allow extracting air for cooling/sealing purposes. These compressor bleeds are modeled in the platform with the help of flow splitters that, as their names indicates, split the flow in two streams, the main stream and the bleed-related one. Since a typical gas turbine engine includes several cooling/sealing bleeds, as many splitters as required can be used for modeling these sub systems.

As expected the gas turbine combustor is modeled using two inlet streams, one corresponding to the oxidant (air) and the other to the fuel. After combustion the working fluid is cooled down using air from the compressor discharge. The complex turbine cooling system, relaying on different heat transfer mechanisms, utilized for protecting the first turbine vane/blade rows is modeled using a simplified mixer component the reduces the working fluid temperature before entering the first turbine stage. Similarly, other mixer components are utilized for mixing the remaining cooling bleeds with the flows leaving the different turbine sections. The reduction of the working fluid temperature throughout the expander has been possible thanks to use of four turbine sections, which allow the connection of these sections to the mixers and these last ones to the splitters distributed along the gas turbine compressor. Notice as well that mixers are also utilized when required for mixing the compressor blow-off flows and the main exhaust flow from the gas turbine.

To finalize this section it is important to emphasize that gas turbine-based power generation plants include many more components that those main ones illustrated in Figure 3. These other components include for instance turning gears, shafts, bearings, electrical generators, inlet air cooling systems and heat exchangers. Even though the objects corresponding to these plant components do not appear explicitly in Figure 2 and Figure 3, they are already available in the OOP-based platform developed. These objects are used when building up high fidelity full scope power plant simulators dealing with all operational details present in actual power plants. Next some results obtained from the use of the gas turbine model described in Figure 3 are briefly discussed.

4.2. Simulation Results

The gas turbine model shown in Figure 3 has been run at both design and off-design point conditions. For the off-design point case five points have been considered only. The main results from the simulations at design point are summarized in Table 1. This table shows that the differences between the results obtained from the model and the engine data available in open literature (Soares, 2008) are small. Notice that for obtaining these design point results educated guesses of initially unknown parameters such as component efficiencies, cooling bleeds and pressure losses have been carried out.

Table 1. Design point results.

Parameter	Data (Soares, 2008)	Model	Difference
Power output [MW]	115.4	115.9	0.5%
Thermal efficiency	0.336	0.335	0.3%
Heat rate [BTU/kWh]	10150	10184	0.3%
Exhaust temperature [K]	531	530	0.3%

Regarding the off-design point simulations, it is worth highlighting first that, as this usually represents proprietary data, no component characteristics were available for the gas turbine modeled. Accordingly, in the case of the compressor, a properly scaled generic compressor map was utilized. For the turbine, choking conditions were assumed. Since the off-design points accounted for lie close to the design point one this assumption seems to be reasonable.

The main results from the off-design simulations are shown in Figure 4 and Figure 5 below. More specifically, Figure 4 shows the variation of gas turbine thermal efficiency and power output as a function of turbine entry temperature (TET). As expected both parameters increase with the increase in TET. In a similar fashion, Figure 5 shows the variation of compressor pressure ratio and gas turbine exhaust temperature with TET. Since the off-design point calculations were carried out considering constant compressor entry conditions and engine rotational speed, it is natural to expect an increase in compressor pressure ratio as TET increases. This increase in pressure ratio leads of course to reductions in compressor surge margin and all risks that this involves. The increase in exhaust temperature is in turn a direct consequence of the increase in TET.

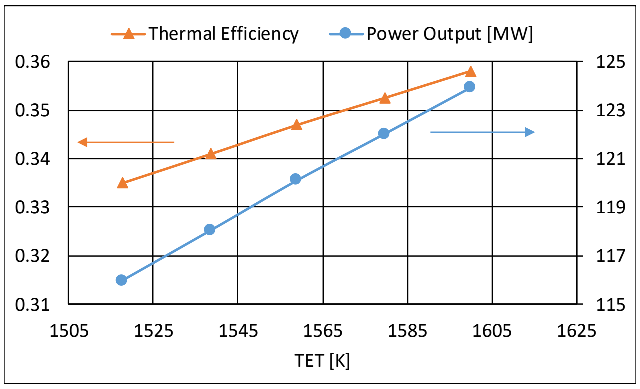

Figure 4. Thermal efficiency and power output as a function of TET.

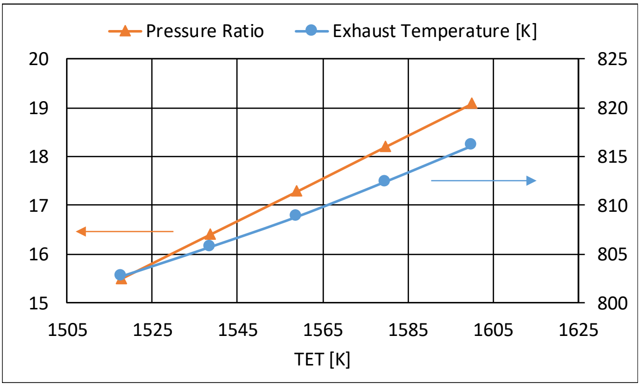

Figure 5. Compressor pressure ratio and gas turbine exhaust temperature as a function of TET.

4.3. Further Results

To complement the results discussed in the previous section, first results obtained from the dynamic simulation of an actual power plant are described here. The referred results correspond to outcomes from an industrial gas turbine transient model belonging to an actual full scope power plant simulator under construction using the OOP-based platform developed. For the sake of illustrating the capabilities of the transient model and the utility of the OOP-based platform, a 30-minutes scenario involving a change load from about 80 to 100% of nominal power is simulated. Notice that in the plots showing the comparisons between the parameters computed and those measured in the power plant, lines identify the results coming from the models, whereas symbols identify the power plant data. In addition, a single color is utilized to characterize a given parameter.

Accordingly, the normalized values of the parameters that characterize the gas turbine main components – compressor, combustor and expander, besides the power output, are highlighted in Figure 6 and Figure 7. The reference parameters used for normalization here correspond to those characterizing the gas turbine full load condition. As it can be readily verified from these figures, the results obtained from the transient model follow the trends of the parameters recorded by the power plant data acquisition system. It implies then that the model can properly reproduce the transient behavior of the gas turbine. Indeed, in the first part of the simulated period, the relative discrepancies between computed and measured parameters are relatively small, about 0.5 to 1%. When the change load start taking place, the referred discrepancies increase, but they remain of the order of 2-3%. Even though they are not shown in this work for the sake of brevity, it is worth noticing that similar results are obtained when other operating scenarios such as start up and shut down are accounted for. The detailed results from the referred scenarios will be published later elsewhere.

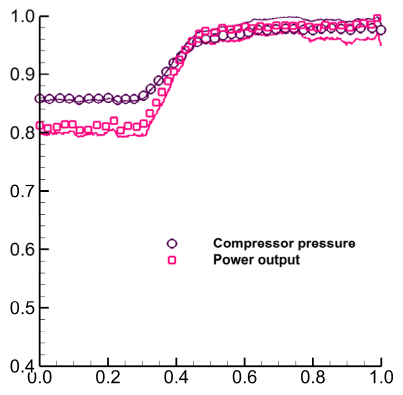

Figure 6. Temporal evolution comparison of computed and measured parameters – Compressor exit pressure and power output.

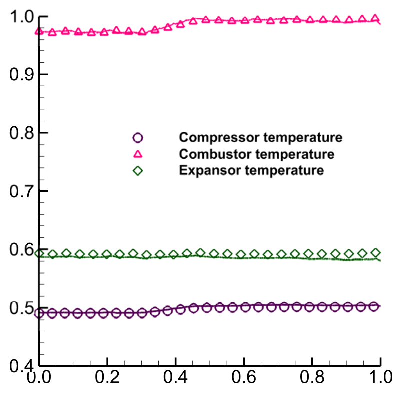

Figure 7. Temporal evolution comparison of computed and measured parameters – Compressor, combustor and turbine exit temperatures.

It is expected that the case study results and those related to the dynamic simulation of an actual power plant discussed in this Section 4 provide to the reader an insight of the type of models that can be created using the OOP-based platform under discussion. Finally, it is worth mentioning that the quality of results to be obtained using this platform depends, as always, on the quality of the models included in each of the platform objects representing the power plant components. The higher the fidelity of these component models, the higher the quality of the results obtained from the simulations.

5. CONCLUSIONS

In this work, the development of a generic OOP-based platform suitable for building up full scope power plant simulators has been described. Specifically, details of the rationale behind the conception, design and implementation of the referred OOP-based platform have been initially provided. An especial emphasis has been put on several architectural aspects accounted for and object class hierarchies defined during the development of the platform. Some of the main modules composing the platform have been described thoroughly. Specifics have been thus provided of the component-based design of these modules, along with the inter-component communication approach utilized in the developed platform.

Since the OOP-based platform is intended to be as generic as possible, allowing the construction of any power plant simulator, details of the inter-component communication, key aspect in the platform discussed, have been emphasized. As a means of illustration then, the developed platform has been utilized for creating a simple industrial gas turbine engine model. Simulations at design and off-design point conditions using this model have been carried out and the results discussed. With this same purpose, transient results obtained from the simulation of an actual power plant have been also included. The coherence of the obtained results highlight the usefulness of the platform whose development has been described in this work. It is concluded thus that the development of the platform is following the right track. This OOP-based platform is expected to be used in future for building up many high-fidelity simulators providing practical training to power plant personnel.

6. REFERENCES

Automation.com, 2012. “Operator Training Simulators market to grow”. 3 Sep 2017 <http://www.automation.com/portals/manufacturing-operations-management/plant-asset-management/operator-training-simulators-market-to-grow>.

Bindu, S., Jayanthi, T., SatyaMurty, S.A.V., Swaminathan, P., Raj, B., 2011. “Role of Animated Human Machine Interface in Nuclear Power Plant Simulation”. *International Journal of Simulation Modelling*, Vol. 10, pp. 5-16.

Curlett, B.P., and Felder, J.L., 1995. “Object-Oriented Approach for Gas Turbine Engine Simulation”. *NASA Technical Memorandum* 106970, Cleveland, USA.

Curlett, B.P., and Gould, J.J., 1994. “Flexible Method for Inter-Object Communication in C++”. NASA Technical Memorandum 106315, Cleveland, USA.

EPE (Empresa de Pesquisa Energética), 2013. “Plano Decenal de Expansão de Energia 2021”. 3 Sep 2017 <http://www.epe.gov.br/PDEE/20130326_1.pdf>.

Follen, G., and auBuchon, M., 2000. “Numerical Zooming Between a NPSS Engine System Simulation and a One-Dimensional High Compressor Analysis Code”. *NASA Technical Memorandum* 209913, Cleveland, USA.

ISA, 2005, “Fossil Fuel Power Plant Simulator – Functional Requirements”. ANSI/ISA-77.20-1993 (R2005), North Carolina, USA.

Krost, G., Allamby, S., and Lehtonen, P., 2000, “Organization and justification of power system operators training”. CIGRE SC 39 Session, Paris, France.

Parsons, D., 2000. *Object-oriented programming with C++* (2nd Ed.). Continuum, London, UK.

Soares, C., 2008. *Gas Turbines, A Handbook of Air, Land and Sea Applications* (1st Ed.). Butterworth-Heinemann, London, UK.

Takai, M., 1990. *A new architecture and expert system for aircraft design synthesis*. PhD Thesis, Stanford University.

Visser, W.P.J., and Broomhead, M.J., 2000. “GSP, a Generic Object-Oriented Gas Turbine Simulation Environment”. *ASME Turbo Expo 2000: Power for Land, Sea, and Air*, Munich, Germany.

7. RESPONSIBILITY NOTICE

The authors are the only responsible for the printed material included in this paper.