Python-Modelica Framework for Automated Simulation and Optimization

Mareike Leimeister[1,2]

[1]Naval Architecture, Ocean and Marine Engineering, University of Strathclyde, United Kingdom
[2]Fraunhofer IWES, Fraunhofer Institute for Wind Energy Systems, Germany,
mareike.leimeister@iwes.fraunhofer.de

Abstract

Modeling and simulation are essential for the development of complex engineering systems, such as wind turbines. Thus, Fraunhofer IWES (Fraunhofer Institute for Wind Energy Systems) has developed the MoWiT (Modelica for Wind Turbines) library for fully-coupled aero-hydro-servo-elastic simulations of wind turbine systems. To meet the needs for detailed assessment and design development of such sophisticated engineering systems, which imply iterative steps for design optimization, a Python-Modelica framework is set up and presented in this paper. By means of this, the simulation of MoWiT models can easily be managed, including redefinition of model parameters, specification of output sensors and simulation settings, integration of optimization algorithms, post-processing of simulation results, as well as parallel execution of several simulations. The application of this Python-Modelica framework is shown based on the example of a design optimization task of a floating wind turbine support structure.

Keywords: Modelica, OneWind, MoWiT, Python, wind turbines, automated design optimization

1 Introduction

The development process of engineering systems is very complex, labor-intensive, and extensive. System simulation, analysis, and of course optimization are of high importance in, for example, power, control, automotive, aerospace, marine, material, or building engineering. The focus of interest could range from general design optimization, through performance or efficiency enhancement, including for example flow properties or comfort aspects, to a commonly envisaged cost reduction. Regardless of objectives, constraints, criteria, and engineering system, design processes always implicate several iterations, in which the evolving designs are tested, analyzed, and modified accordingly until an optimized design is achieved. Thus, an automated simulation framework is essential to cope with the large number of simulations, required to assess and develop such an engineering system design in detail, but also to support design optimization processes, in which iterative simulations have to be executed.

Good examples for such intricate engineering systems and their extensive development process are wind turbines. These power plants have to comply with requirements from standards, such as IEC 61400-3 (International Electrotechnical Commission, 2009) or DNVGL-ST-0437 (DNV GL AS, 2016), and need to be tested on their performance in various environmental conditions, including loads and system responses. However, the complexity of wind turbine systems, with their non-linear system behavior and couplings between aerodynamics, hydrodynamics (if offshore), control system, and structural dynamics, makes modeling and simulation indispensable.

Thus, at Fraunhofer IWES (Fraunhofer Institute for Wind Energy Systems) a computational model for wind turbine load calculations has been developed in the open-source object-oriented and equation-based modeling language Modelica. This modeling language has the power to deal with multi-physics problems and, hence, can be used for simulation of various engineering systems. The MoWiT (Modelica for Wind Turbines) library[1] is capable of fully-coupled aero-hydro-servo-elastic simulations of wind turbine systems - onshore, bottom-fixed offshore, or even floating offshore. The hierarchical programming and the multibody approach in Modelica allow representation of the entire wind turbine system through models for single components. This component-based structure of the MoWiT library simplifies the adaption and modification of a wind turbine model because single components can easily be exchanged or customized. (Leimeister and Thomas, 2017; Thomas et al., 2014; Strobel et al., 2011)

Even if MoWiT can model the non-linear system behavior, a large number of simulations are required for the design of an optimized wind turbine system. For this purpose a Python-Modelica framework is developed for automated execution of simulations and optimization tasks.

[1]formerly OneWind Modelica library

In this paper, first the simulation framework in Python, interfacing with models created in MoWiT, is presented in detail in Section 2, followed by the extension of the framework for automated optimization applications, covered in Section 3. Afterwards, the application of this Python-Modelica framework is shown exemplarily on the design optimization of a floating wind turbine system (Section 4). Conclusions are given at the end in Section 5.

2 Simulation Framework in Python

The framework for automated simulation of wind turbine models requires

1. a modeling environment, which is the MoWiT library building upon the Modelica modeling language;
2. a tool for executing the time-domain simulations (Dymola[2]);
3. and a programming interface (Python[3]) for external and automated control of the simulations.

The tools, which are selected to be incorporated in one framework for automated simulation, stand in perfect mutual complement. The Modelica based modeling environment in combination with the Dymola simulation engine is very suitable for time-domain simulations of complex multi-physics engineering problems. Programming in Python, on the other hand, facilitates the management and handling of simulations, controls the entire simulation process, and creates a set framework for automated application to engineering systems models and problems.

A schematic representation of the simulation framework in Python is presented in Figure 1. In Modelica, using the MoWiT library, the considered wind turbine system is specified and all parameters are set, so that the model can be simulated in Dymola. With setting up the MoWiT model, a Modelica package is created. This is the main input to the Python-Modelica framework as it contains all necessary information about the simulated model (structure, components, parameters, equations, states, ...). Due to the fact that the Python-Modelica framework should also be used to set and modify parameter values according to specific simulation requirements, it is important to add `annotation(Evaluate=false)` to these parameters, when defining them in the MoWiT model.

The simulation framework in Python itself works on different levels, as shown in Figure 1. It contains a `Model Wrapper` for processing the Modelica package and establishing the interface to Modelica based on the Python package `BuildingsPy`[4]. On the next level, the `Simulation Manager` handles the instances from the `Model Wrapper` and manages the simulations. Finally, a main script is required to execute the simulation task and define additional commands, for example for writing result files or for post-processing.

2.1 Processing the Modelica Package

The Modelica package of the created MoWiT wind turbine system model is given as input to the `Model Wrapper`. The Python-Modelica interface is defined based on the available interface between Python, Modelica, and Dymola, provided by the Python package `BuildingsPy`[4]. Within this package, the main class, which is finally required to simulate a Modelica model, is the `Simulator`.

2.1.1 The Simulator

The Python script for the class `Simulator` is taken from the Python package `BuildingsPy` and slightly modified to make it compatible with the used Python 3.x version. The `Simulator` provides the interface between Python and Modelica to run simulations with Dymola. Based on the inputs for model name and the path to the Modelica package of the MoWiT model, the used simulation engine (Dymola) and the path to the executable, as well as optional inputs for working and

[2]https://www.3ds.com/products-services/catia/products/dymola/ (Accessed: 5 October 2018)

[3]https://www.python.org/ (Accessed: 26 October 2018)

[4]http://simulationresearch.lbl.gov/modelica/buildingspy/ (Accessed: 9 October 2018)

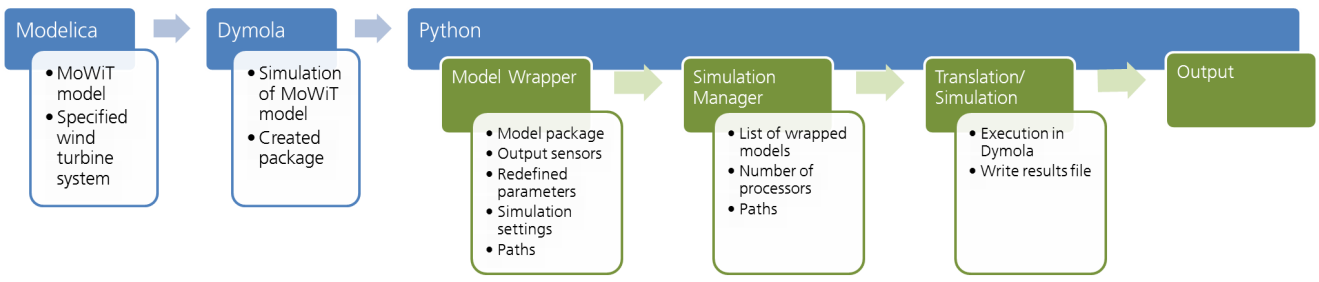


Figure 1. Simulation framework in Python.

output directories, the methods for setting paths, directories, but also simulation parameters and commands are defined. Furthermore, methods for adding pre-processing statements when translating or simulating the model, as well as post-processing statements before writing the log-file are specified. Finally, the methods to simulate a model, translate a model, or simulate a translated model are declared.

2.1.2 The Model Wrapper

The `Simulator` is called within the `Model Wrapper`, which specifies parameters, paths, and simulation settings for Dymola to be used in the `Simulator`. Besides this, also parameters to be set for translation or to be redefined before simulation, pre-processing statements to be added ahead of translation or simulation, as well as a list of output sensor names are defined.

The basic `Model Wrapper` class directly processes a Modelica package of a MoWiT model and modifies, if required, specified parameters and settings. Additionally, a method is defined to write Dymola commands for generating a `*csv` output file after completion of the simulation. Furthermore, the total number of simulations to be executed is specified. This is especially relevant when running several simulations, which could be processed in parallel or successively. This is managed by the `Simulation Manager`, which is introduced in the next Subsection 2.2.

2.2 Managing the Simulation

Wrapped models are then further processed in the `Simulation Manager`. The input list of wrapped models could contain

- one instance of a `Model Wrapper` class, corresponding to just one MoWiT model;
- one instance of a `Model Wrapper` class, based on one and the same model, however, comprising several simulations with different parameter settings;
- or several different instances of a `Model Wrapper` class for working with various MoWiT models.

These models in the list of wrapped models can be handled either successively or in parallel, while for the latter the number of processors used for multi-processing in a pool can be specified as additional input to the `Simulation Manager`. Both forms of management are available for different processing methods:

- translating a wrapped model;
- simulating a translated wrapped model;
- creating a turbulent wind file.

The first two methods are calling functions in the `Simulator`. The latter method is only relevant for simulations with turbulent wind. This is defined through the turbulence intensity, as well as the wind spectrum type (Kaimal, von Karman, or Mann). In this Python-Modelica framework application, TurbSim (Jonkman, 2009) is used for generating a turbulent wind field. A file containing the time series of the wind speed could

- either already exist and the path to this file has to be specified directly in the MoWiT model or is given as input to the `Simulation Manager`;
- or still has to be generated, which requires the path to the TurbSim executable, as well as the wind turbine and simulation case specific TurbSim input file.

2.3 Executing the Task

Finally, a main script is required to execute the simulation task and define additional commands, for example for writing result files or post-processing calculations. This script highly depends on the application case. Hence, for running a large number of simulations, such as in the case of design load case simulations, only the simulations to be executed, as well as simulation settings, paths, and input parameters are to be specified. However, for applying the Python-Modelica framework to automated optimization tasks, additional code, in which design variables and objective functions are defined and linked to existing Python packages for optimization, has to be written in the main script. More detailed information on the Python-Modelica framework extension for the use for automated optimization is given in the following Section 3.

3 Extension for Automated Optimization

The Python-Modelica framework, as presented in Section 2, serves as basis for further applications, apart from automated simulation, such as the realization of optimizations. The extension of the Python-Modelica framework for automated optimization is of significant importance, as optimization tasks are highly iterative. This finds profitable use in the design and optimization of wind turbine systems. Due to the complexity of an (offshore) wind turbine model, comprising a huge number of parameters, and the non-linear system behavior, optimization problems cannot directly be solved and a large number of iterations has to be gone through.

The wind turbine system model, which should be used for optimization purposes, has to be wrapped and processed with the `Model Wrapper` and `Simulation Manager`, respectively, according to the explanations

in Subsections 2.1 and 2.2. This, together with further definitions regarding the optimization process, is passed to the main script, by which means finally the execution of the optimization algorithm is started (see Subsection 2.3). Additional information on the optimization itself is provided by separate classes, clustered into the optimization problem, the optimizer, and the optimization algorithm, which are introduced in the following Subsections 3.1 to 3.3.

3.1 The Optimization Problem

Based on the model from the `Simulation Manager`, the optimization problem has to be described. This comprises definitions of design (also called optimization) variables, objective functions, as well as additional constraints. As the Python-Modelica framework works without a GUI, all input has to be provided in the programming scripts. These, however, are coded in such a way, that they all internally use variables, which are only once defined in the main script and assigned their values by means of the user input.

The optimization variables are the design parameters of the wind turbine model, which are to be modified during the optimization iterations. The parameter names must be provided according to the Modelica dot-notation and following the structure within the MoWiT model. Since these parameters are assigned new values during the optimization, it is important that they are still existing in the compiled model (see the remark at the beginning of Section 2).

As important as optimization variables for an optimization procedure are objective functions. These describe the goals, which are to be obtained by means of the optimization. Mostly, optimization routines are defined to minimize the objective functions, thus, these have to be provided accordingly. Depending on the optimization routine type, only one or several objective functions can be processed. For multi-objective optimizers, each goal can be defined separately. However, if the optimizer can handle only one objective function, all goals have to be combined in one expression, in which weight can be incorporated to rank the importance of the single objectives.

The two key elements of the optimization problem are already specified by means of the optimization variables and the objective functions; however, further input can be given in form of constraints. These apply either for the goals and specify if only certain values are allowed or if dependencies or relations exist, or for the design parameters and define the allowable ranges of values which they can take on.

3.2 The Optimizer

The optimization problem is given to an optimizer, which then executes the optimization task and algorithm. There are several open-source optimizers available for the use in a Python environment, such as optimization routines from OpenMDAO (Multi-disciplinary Design, Analysis, and Optimization), an open-source framework for efficient multi-disciplinary optimization, (openmdao.org, 2016); PyGMO (Python Parallel Global Multi-objective Optimizer), focussing on multi-objective (MO) optimization, (Izzo and Biscani, 2015); or Platypus with a special focus on MOEAs (MO Evolutionary Algorithms) (Hadka, 2015) - just to name a few examples.

In the presented Python-Modelica framework, optimization routines from Platypus (Hadka, 2015) and OpenMDAO (openmdao.org, 2016) are implemented. Only gradient-free optimizers can be used for the application to wind turbine models in MoWiT, as these models represent too complex systems, which cannot be reduced to one single equation by means of minimization techniques. Furthermore, the high complexity also attributes greater importance to multi-objective optimizers.

Apart from optimizer-specific inputs, a criterion has to be specified for limiting the number of iterations within the optimization process. This could be defined for instance through the number of optimization cycles to be performed or a convergence tolerance for the results.

3.3 The Optimization Algorithm

Using the defined optimization problem and the specified optimizer, as described in Subsections 3.1 and 3.2, respectively, the optimization algorithm is executed. In each run, the design variables are modified, based on the objective results from previous simulations, complying with the defined value ranges of the optimization variables, and following the optimizer-specific routine. The iterative optimization simulations are terminated as soon as the specified stop criterion is fulfilled. Figure 2 visualizes this process schematically. Furthermore, depending on the specified processing method, as set in the `Simulation Manager` (see Subsection 2.2), several simulations within the optimization routine may be executed in parallel.

Due to the fact that - especially at the beginning of the optimization routine - also suboptimal settings might be selected by the optimizer, it could happen that simulations of individual models are aborted before the specified simulation duration. To handle these or similar failures a query condition can be incorporated when analysing the results for evaluating the objective functions. One possible approach is to check if the simulation was successful by evaluating the last entry in the time output. In case of aborted simulations, the goals might not be

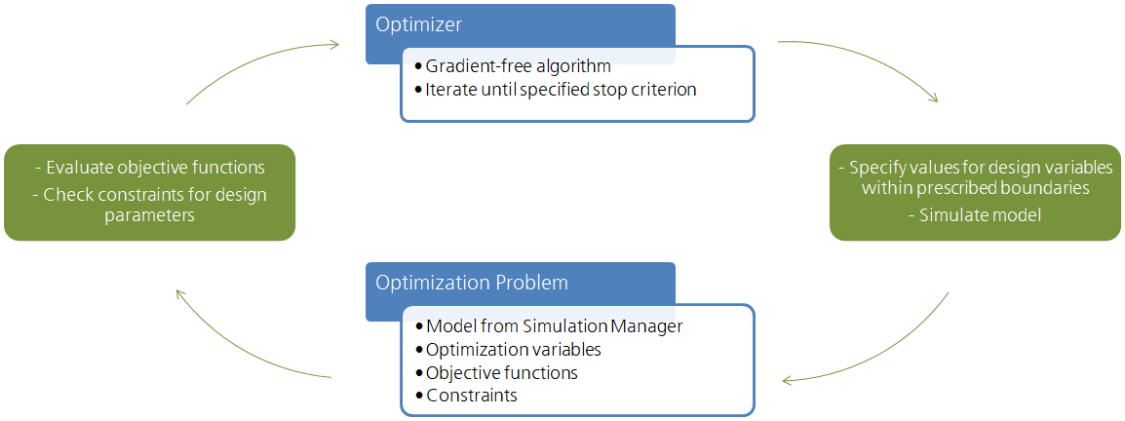


Figure 2. Automated optimization algorithm in Python.

derived as defined, but set to undesireable values to ensure that these unsuccessful and thus suboptimal individuals are excluded and not considered further by the optimizer.

During the execution of the optimization algorithm, the simulation results are written in *csv output files according to the defined method in the `Model Wrapper`, as outlined in Paragraph 2.1.2. By means of supplementary code, additional outputs, such as the objectives of each solution, can be written and exported subsequent to the optimization.

4 Application Example: Design Optimization of Wind Turbine Systems

Optimization tasks in the development of wind turbine systems are wide-ranging. Mostly costs, and thus indirectly also performance and material demand, are the main drivers, but optimization problems can for instance as well be related to noise emissions, dimensions, and lifetime. In the following the Python-Modelica framework is exemplarily applied to automated design optimization of a floating offshore wind turbine system.

In this optimization task, the floating spar-buoy wind turbine system from phase IV of the Offshore Code Comparison Collaboration project OC3 (Jonkman, 2010) is used. The floating wind turbine system consists of a spar-buoy platform, which supports the NREL 5 MW reference wind turbine (Jonkman et al., 2009). The visualization of the MoWiT model in Dymola is presented in Figure 3. There, also the coordinate system of the wind turbine, as well as the nomenclature of the six degrees of freedom of movement are introduced.

The optimization algorithm is defined based on the following problem and settings:

- Three parameters of the spar-buoy floating platform are selected as design variables with their corresponding allowable value ranges: the diameter (between 6.5 m and 10.0 m) and the height (between 68.0 m and 108.0 m) of the spar-buoy column, as well as the density of the ballast (between 1281.0 kg/m^3 and 2600.0 kg/m^3). A fourth indirect variable, the amount of ballast (filling height in the column), is internally determined and adjusted, based on the design parameters and to ensure floatation of the system.

- Three objective functions and corresponding constraints are defined to limit the maximum system inclination (pitch) to 10°, limit the maximum nacelle or tower-top acceleration to 1.962 m/s^2, and to minimize the floater translational motion (combined surge, sway, and heave).

- The optimizer NSGAII[5] from Platypus (Hadka, 2015) is used due to the multi-objective optimization task and the optimization algorithm is executed for 25 generations with 36 individuals each.

The variation of the floater design within the optimization algorithm is shown in Figure 4. The black shape represents the original geometry and corresponding ballast height (indicated by the dashed line). A few exemplary geometries of individuals obtained during the optimization are presented in different green tones and reveal the ranges of the design variables.

A more detailed analysis of the simulation results shows that both the spread of the design parameters and the spread of the optimization objectives converge, with having a minimum spread in generation number 13, as indicated in Figure 5. From this, the final optimum geometry is selected, which is displayed in red in Figure 4. With this design, the objectives are achieved, while still fulfilling the prescribed boundaries and constraints for the design parameters.

[5]Non-dominated Sorting Genetic Algorithm II

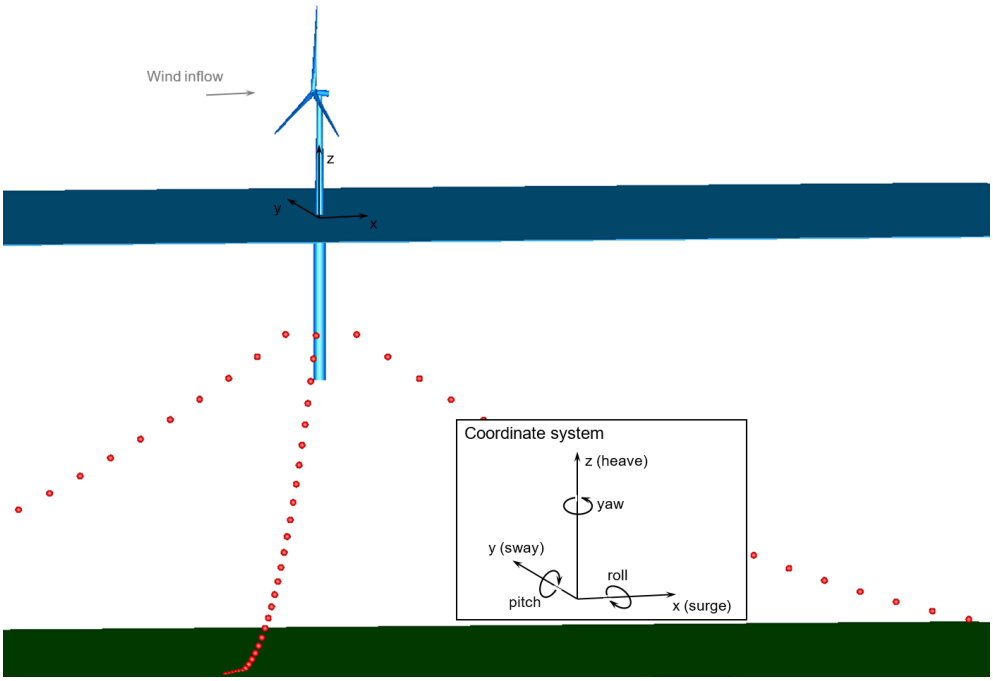


Figure 3. Floating spar-buoy wind turbine system in MoWiT, including coordinate system and system degrees of freedom, as well as wind inflow direction.

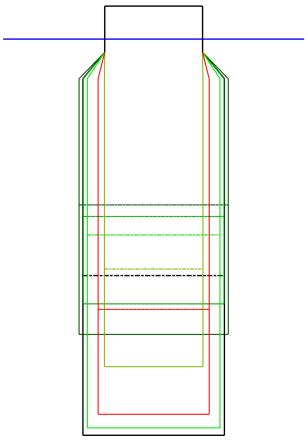
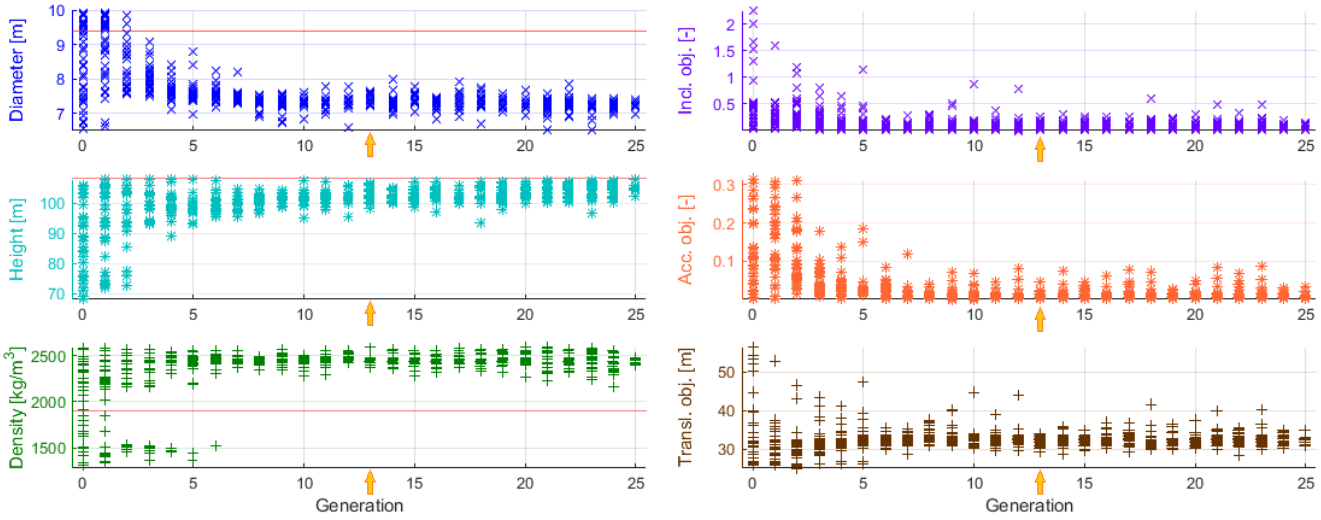


Figure 4. Interim (green tones) and final (red) results from the floater design optimization procedure, in comparison with the original design (black).



(a) Development of the design variables column diameter (blue), column height (cyan), and ballast density (green), together with the original values (red).

(b) Development of the objectives for system inclination (violet), nacelle acceleration (orange), and floater translation (brown).

Figure 5. Results from the floater design optimization procedure, arrows indicate the generation from which the final optimum design is selected.

5 Conclusion

In this paper, a Python-Modelica framework is presented, by which means Modelica models can be managed and simulations executed automatically, using scripts programmed in Python. Models for entire wind turbine systems (onshore, bottom-fixed offshore, or even floating offshore) are created in the MoWiT library, which are then simulated in Dymola. The external and automated control of the simulations is taken over by various Python scripts. These are split up into methods for processing the Modelica package of the MoWiT model, methods for managing the simulation, and the main script for executing the task and performing further (post-)processing. By means of this Python-Modelica framework iterative simulations can automatically be performed, which is very relevant for the assessment, design, and optimization of wind turbine systems. For the latter application, the framework is extended to cover also definitions for optimization algorithms, including optimization problem and optimizer.

An exemplary optimization task for design optimization of a floating wind turbine support structure demonstrates that the presented Python-Modelica framework automates the execution of a large number of simulations, is capable of handling non-linear system behaviors, and thus is a valuable tool for detailed assessment of wind turbine system designs.

Acknowledgements

This work was partially supported by grant EP/L016303/1 for Cranfield University, University of Oxford and University of Strathclyde, Centre for Doctoral Training in Renewable Energy Marine Structures - REMS (http://www.rems-cdt.ac.uk/) from the UK Engineering and Physical Sciences Research Council (EPSRC). The author also wants to thank Philipp Thomas and Niklas Requate from Fraunhofer IWES, as well as Athanasios Kolios and Maurizio Collu from University of Strathclyde for their contribution.

References

DNV GL AS. *Loads and site conditions for wind turbines: Standard DNVGL-ST-0437*. November 2016 edition, 2016. URL `https://www.dnvgl.com/`.

David Hadka. *Platypus Documentation, Release*. 2015. URL `https://platypus.readthedocs.io/en/latest/`.

International Electrotechnical Commission. *Wind turbines – Part 3: Design requirements for offshore wind turbines: International standard IEC 61400-3*. 1.0 edition, 2009.

Dario Izzo and Francesco Biscani. Welcome to PyGMO, 2015. URL `https://esa.github.io/pygmo/index.html`.

Bonnie J. Jonkman. *TurbSim User's Guide: Version 1.50: Technical Report NREL/TP-500-46198*. National Renewable Energy Laboratory, 2009.

Jason Jonkman. *Definition of the Floating System for Phase IV of OC3: Technical Report NREL/TP-500-47535*. National Renewable Energy Laboratory, 2010.

Jason Jonkman, Sandy Butterfield, Walt Musial, and George Scott. *Definition of a 5-MW Reference Wind Turbine for Offshore System Development: Technical Report NREL/TP-500-38060*. National Renewable Energy Laboratory, 2009.

Mareike Leimeister and Philipp Thomas. The OneWind Modelica Library for Floating Offshore Wind Turbine Simulations with Flexible Structures. In *Proceedings of the 12th International Modelica Conference*, Linköping Electronic Conference Proceedings, pages 633–642. Linköping University Electronic Press, 2017. doi:10.3384/ecp17132633.

openmdao.org. OpenMDAO 2.4.0 Beta documentation: Optimizer, 2016. URL `http://openmdao.org/twodocs/versions/latest/tags/Optimizer.html#optimizer`.

Michael Strobel, Fabian Vorpahl, Claudio Hillmann, Xin Gu, Adam Zuga, and Urs Wihlfahrt. The OnWind Modelica Library for Offshore Wind Turbines - Implementation and First Results. In *Proceedings of the 8th International Modelica Conference*, Linköping Electronic Conference Proceedings, pages 603–609. Linköping University Electronic Press, 2011. doi:10.3384/ecp11063603.

Philipp Thomas, Xin Gu, Roland Samlaus, Claudio Hillmann, and Urs Wihlfahrt. The OneWind Modelica Library for Wind Turbine Simulation with Flexible Structure - Modal Reduction Method in Modelica. In *Proceedings of the 10th International Modelica Conference*, Linköping Electronic Conference Proceedings, pages 939–948. Linköping University Electronic Press, 2014. doi:10.3384/ECP14096939.