Architecture Design and Internal Implementation of a Universal Coupling Between Controllers and Physics in a Tokamak Flight Simulator

Chuanren Wu,[a]* Pierre David,[b] Emiliano Fable,[b] Domenico Frattolillo,[c,d] Luigi Emanuel Di Grazia,[c] Massimiliano Mattei,[c,d] Mattia Siccinio,[b,e] Wolfgang Treutterer,[b] and Hartmut Zohm[b,e]

[a]*Karlsruhe Institute of Technology, Institute for Pulsed Power and Microwave Technology, Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany*

[b]*Max Planck Institute for Plasma Physics, 85748 Garching, Germany*
[c]*Consorzio CREATE, Via Claudio 21, Napoli, 80126, Italy*
[d]*Università degli Studi di Napoli Federico II, Dipartimento DIETI, Via Claudio 21, Napoli, 80126, Italy*
[e]*EUROfusion, 85748 Garching, Germany*

Abstract — *The flight simulator predicts the dynamic behavior of a full plasma discharge (described in terms of one-dimensional profiles) by employing multiple control loops based on synthetic diagnostics, which could also emulate realistic sensor and actuator models. It serves as a valuable tool for designing and optimizing plasma scenarios, as well as for assessing the feasibility of controlling discharges. The Fenix flight simulator, originally developed for the ASDEX Upgrade, has been ported to EU-DEMO and is capable of modeling any tokamak.*

One of the essential elements in a flight simulator is the link between the co-simulated plasma physics and the control loops. This element is tightly coupled to the specifications of both the plasma model and the control algorithms to be implemented; but on the other hand, to ensure the portability and applicability of the flight simulator to different scenarios or devices, the coupling between plasma and control algorithms should be neutral to any concrete device and configuration. In addition, as a serial component of the control loop, data exchange takes place at every single step of the control simulation, therefore an efficient implementation is critical for the overall simulation performance. This paper summarizes the universal approach recently implemented in Fenix, which satisfies all the above requirements while remaining lightweight.

Keywords — *Flight simulator, feedback control, full discharge, co-simulation, ASDEX Upgrade.*

Note — *Some figures may be in color only in the electronic version.*

I. INTRODUCTION

Integrated control simulation approaches[1–10] combine scenario modeling with controllability studies in a self-consistent manner. More recently, tools are being developed that also allow for simulating full tokamak discharges and checking the feasibility of the operational and physics goals. The first-of-its-kind flight simulator Fenix[7–11] was originally developed for ASDEX Upgrade (AUG) full-discharge modeling.

Currently, Fenix is also used for studies on the EU-DEMO tokamak reactor prototype,[12] and can be even ported to other existing or planned tokamaks. Such a

*E-mail: chuanren.wu@kit.edu

flight simulator incorporates realistic controller and actuator models with nonlinear responses from the co-simulated plasma dynamics. For the controllers, Fenix builds upon the ITER PCSSP,[13] which has been developed in Simulink®.[14] For the plasma response, Fenix relies on the ASTRA transport code,[15,16] which serves as a general transport simulation framework and includes a broad list of third-party or user-defined modules for modeling particular physics, as is enumerated in Fig. 1 in the next section.

The plasma model consists of a set of one-dimensional (1-D) partial differential equations solving for the main ion and electron temperatures, the ions and electron species densities, and the plasma current density profile. Moreover, a two-dimensional Grad-Shafranov solver computes the magnetostatic equilibrium of the system, and the circuit equations allow for the dynamical evolution of this equilibrium. The integrated simulation of a flat-top scenario or a full discharge on a moderate workstation computer completes in the order of a few minutes of wall time. For example, a full-discharge simulation of AUG typically takes 5 to 10 min, while it takes more time for DEMO only due to the longer discharge duration.

Fenix demonstrated its capability as a flight simulator by successfully applying it on AUG as its initial application. It was used for validations of the reduced physics models,[11] the controller modules (which emulate the actual AUG control system), and even for prediction of novel discharges as a check on request from experimental leaders.[7–9] As Fenix was ported to EU-DEMO with further developments,[10] the same software architecture was kept, where the basic framework and common code modules were generalized for being shared between these two devices. Among these common modules, the bridge between the control environment and the plasma model has been the focus of this recent development to make it correct, robust, and universal. Interfaces to the two main blocks of the flight simulator are implemented in this coupling module:

1. In the control loop (interface to Simulink), the module takes actuator signals, such as the signal for gas puff, pellets, heating, pumping, coil voltages, etc., as input, while the diagnosed data, such as plasma current, radiation power, density, and temperature profiles in the 1-D-grid, as well as the plasma shape given by the (fixed- or free-boundary) equilibrium solver, e.g. Fable et al.[16] incorporated in ASTRA, are fed back to the controllers as output. In the current implementation, the synthetic diagnostics of 1-D profiles and equilibria are raw simulation data from the physics modeling or directly computed from them. They do not faithfully reflect the experimental ones. Future development will enhance the realism of the diagnostic data by reconstructing the physical quantities. This will entail the integration of real-time tools, similar to those used in the experiments, and potentially might also include real-time techniques like in Felici et al.[6] Although the full set of the 1-D profiles (densities, temperatures, fluxes, conductivities, etc.) are available from the physics calculation and can be logged in Fenix, only a selection of the data is exposed to the controllers depending on the input of the control system.

2. In the physics simulation (interface to ASTRA), the data flow is reciprocal to its counterpart in the control loop, i.e., the coupling module passes all control signals to the physics model, while it is fed with plasma responses. In general, Simulink can work with variable, as well as with fixed time steps individually for each model component. Synchronization with physics is triggered whenever one or multiple components in the Simulink model are stepped. ASTRA and Simulink exchange both the present time and the time step consistently alongside the other signals during the communication, such that they will end up at a synchronous time at the end of their respective steps.

Previously for proof of concept of a flight simulator, this coupling was hardwired for a specific machine assuming specific configurations.[7–9] Each signal was handled (e.g., memory-allocated, copied, unit-converted, specifying the order in the data structure) individually. Therefore, modifying the connections without remembering all details could be quite error prone. A complete refactoring aimed at unifying and universalizing the coupling module for both the DEMO simulator and the original AUG would greatly improve the overall structure and also reduce potential mistakes by modifying the hardwiring. While general multiscale co-simulation frameworks, such as Veen et al.[17], already exist, our approach prioritizes cleanliness, lightweight design, and ease of installation by avoiding the use of third-party libraries for coupling the physics co-simulation. Compared to the previous hardwired implementation, the revamped solution is not only much more maintainable, but the same code without any modification can also be extended to other machines or configurations. The resulting implementation has been applied and validated in recent studies.[10–12,18]

II. ARCHITECTURE CONCEPT AND DESIGN CONSIDERATIONS

Multiple concepts are capable of achieving the same goal. As such, the software architecture was chosen based

on the following considerations. To ensure the robustness of the whole flight simulator, ASTRA was invoked as a standalone process such that it was isolated in its own address space. Specifically, the ASTRA process was spawned by the standard fork–exec scheme, and its lifetime was managed by this coupling module living in the Simulink process. Upon spawning, pipes were established for logging purposes, and the paths of files describing the physics model were passed as command line arguments to the child process, thereby ensuring maximum compatibility with the original ASTRA program.

There were several advantages to doing this. First, whenever the transport solver crashed because of numerical errors or unphysical solutions, the crash would not propagate to Simulink, as the latter is slow to (re-)start and may contain unsaved development progress. If the physics calculation crashed, the aborted ASTRA process was cleaned up by the operating system while its parent process, i.e., the Simulink instance, detected the anomalous stop of the physics process in the coupling module by the waitpid system call. Then, the resources acquired for this simulation case would be released during the aborting of the Simulink simulation loop and control given back to the user again.

Second, since the standard Simulink scheme does not give control back to the user when an element hangs or waits, if an algorithm on the physics side does not terminate, the user of the flight simulator will not be able to operate, e.g., will not be able to evaluate and save the previous data or be able to stop the current simulation except restarting Simulink. The multiprocess architecture easily solves this problem, too. If the elapsed time of a physics calculation step exceeds a user-defined threshold value, the user will be asked whether to stop the simulation or to wait longer. In the first case, the recorded simulation data up to the current step will still be available in Simulink after stopping the unresponsive ASTRA process. Besides, this approach requires only minimum changes to make ASTRA compatible with the flight-simulator design. The feature of Simulink called the "S-function" was chosen to implement this coupling with the co-simulated physics.

Based on the decision made previously, resource management is straightforward. The operating system will recycle the resources (e.g., acquired memory, opened files, and semaphores) of the ASTRA process after each completed or terminated simulation. On the Simulink side, the mechanism of S-function provides a RAII (Resource Acquisition Is Initialization) interface to manage the resources during each normal start-stop cycle of simulations, as well as for exceptional termination of a simulation, where the forked ASTRA process belongs also to a special resource being managed in this way. Hence, the lifetime and visibility of the "state" data are only limited to each individual simulation rather than as global variables living in Simulink like in the previous proof-of-concept implementation, which again enhances the robustness and correctness by mitigating the risk of forgetting to re-initialize any reusable global variable between two consecutive simulations.

Error handling is mainly on the Simulink side. The diagnostic messages from the physics calculations, previously printed on the screen (stdout and stderr), are redirected to and logged in Simulink. In the case of failed communications or a crashed ASTRA process, the Simulink part of the coupling module, see Fig. 1, detects and handles these events when the responses time out, while control is given back to user, who will be asked to decide whether to terminate the ongoing run or further wait for a potential response.

In the design being established from the previous discussions, inter-process communications will be needed for the data exchange. The standard methods are pipes, sockets, and shared memories. The last one was chosen to achieve the maximum performance of the flight simulator. Being portable and lightweight, the implementation of the shared memory communications does not necessitate any specific library apart from the standard POSIX interface. The block diagram of the established design is shown in Fig. 1.

The architecture was designed with flexibility in mind, which means that these modules can be employed in different tokamak configurations as is, without modifying the code infrastructure. Therefore, the physical meanings, units, and types (scalar or vector), as well as the number of diagnostic and control signals, should be transparent to the code. As such, these metadata are described in a simple table as input of the configuration, which will be elaborated in the next section.

For simplicity, all signals are uniformly stored and exchanged only in the binary format of the Institute of Electrical and Electronics Engineers (IEEE) 754 double precision, which has sufficient mantissa to exactly represent also a large range of integers. There is no endianness issue when doing this, since in the current version both Simulink and ASTRA run on the same computer. To keep the interface on the ASTRA side clean and simple, the module only exposes two functions for reading and writing data respectively, both of which take a variable number of arguments where each of them corresponds to a signal in the order specified by the configuration.

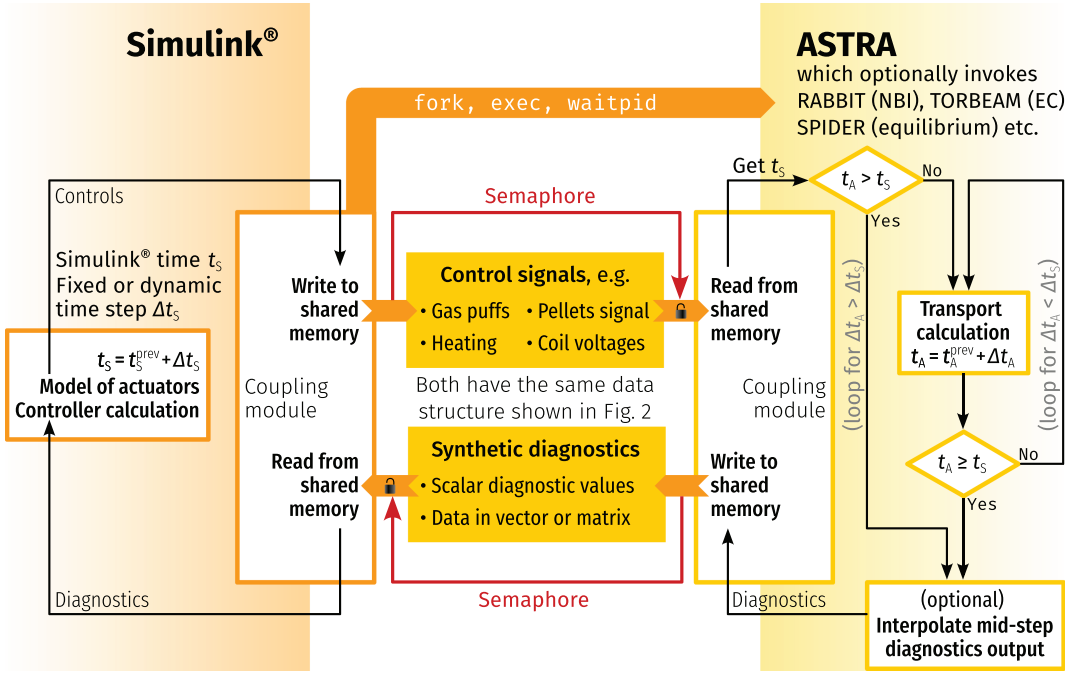

Fig. 1. Block diagram of the coupling and synchronization in Fenix.

To ensure the portability, a minimum requirement is considered: the code is written in C89 and it requires only a small subset of POSIX. Furthermore, it can be seen in Fig. 1 that both modules on the Simulink side and the ASTRA side are symmetric to some degree. Indeed, the symmetric parts, especially the procedures for establishing and performing communications, are unique but shared in both modules for consistency.

III. DATA STRUCTURE AND FLEXIBLE DEFINITION OF THE EXCHANGED DATA

As limited by the internal mechanism of Simulink, the ports on the block representing the input and output of tokamak physics are internally identified only by the order they appear on the left (for input) and right (for output) side of this S-function block. Therefore, each port has common metadata that basically consist of the port width (dimension) and the order in the port list. With this information, the data structure of the shared memory and the procedure for configuring the exchanged data can be designed.

Both the diagnostic and the control signals have the same layout of shared memory, which is described in Fig. 2. It contains only the necessary information, which can be understood by both Simulink and ASTRA independent of the exact tokamak configuration. The layout is designed to be backward compatible for future extensions. As given in Fig. 1, the two instances of this table are protected by their own semaphores for data consistency and time-step synchronization.

The data structure in Fig. 2 was created during run-time by code that takes Table I as input, where the latter is a MATLAB® table defined by user. The "Port Name" columns in this table contain arbitrary text as notes, and these texts are tagged on the corresponding ports of the Simulink block for indication of the port usages. Moreover, Table I was generated by user-defined MATLAB functions to ensure the consistency of the data definition and also to further increase the flexibility. For instance, one could use the raw mesh grid of the temperature and density profiles as synthesized diagnostic data. In this case, they will have the same width, which is the number of grid points, e.g., 100. This number can be specified as a unique global constant in Simulink, which will be evaluated and in all relevant fields of Table I consistently.

During this process, no recompilation is needed. Hence, the port configuration is fully under the user's control rather than a part of the hard-coded program. Once a simulation is initialized, the only run-time overhead is copying (and optionally scaling) the uniform data in memory spaces, whose performance is restricted by the memory bandwidth and should not be the bottleneck of the entire simulation. To summarize, the flexible approach of defining exchanged data is visualized in Fig. 3.

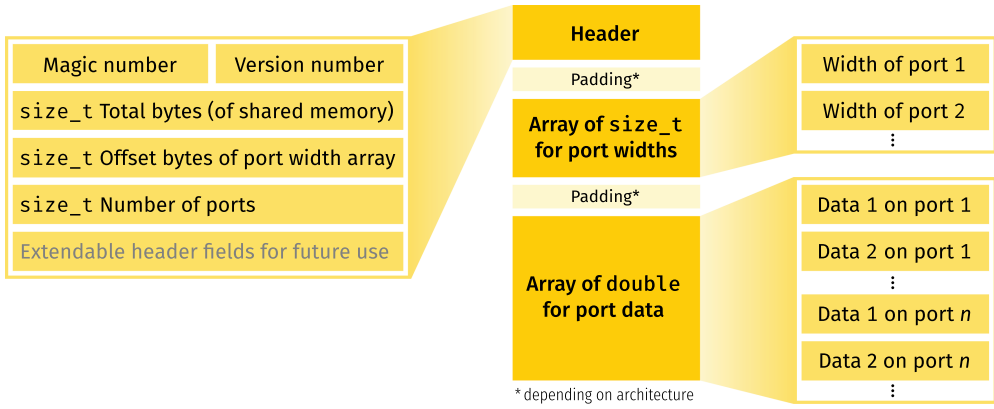

Fig. 2. Layout of a shared-memory block for diagnostic data or control signals in Figs. 1 and 3.

TABLE I

MATLAB Tables Containing the Port Configurations

Example of Control Signals			Example of Diagnostic Data		
Port Name	Width	Scaling Factor[a]	Port Name	Width	Scaling Factor[a]
D-pellet source	1	10^{-19}	I_{plasma}	1	10^{6}
Pump speed	1	1.0	T_e profile	100	1.0
Coil voltages	10	1.0	n_e profile	100	10^{19}

[a]The units themselves are not explicitly stored in the metadata; instead, during the configuration phase, the user defines the scaling factors. The utilization of these scaling factors stems from legacy variable handling, and in future development, the goal is for them to converge to a uniform value of 1 and ultimately be removed entirely. Presently, the scaling is calculated efficiently by optimized code only on the Simulink side.

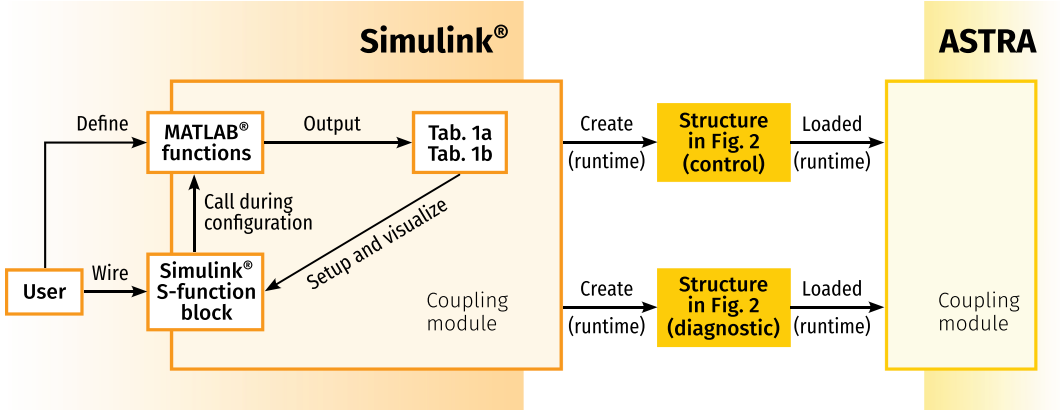

Fig. 3. Human interface and internal handling of metadata.

IV. CONCLUSION

The design and implementation details of the coupling between the plasma model and control environment in the tokamak flight simulator Fenix were presented. The completely refactored module is now robust, universal, and flexible, where all data are handled consistently. Moreover, the scheme is valid for any code coupling of this sort, also beyond the flight simulators for fusion.

Acknowledgments

This work was carried out within the framework of the EUROfusion Consortium, funded by the European Union via

the Euratom Research and Training Programme (grant agreement no. 101052200-EUROfusion). The views and opinions expressed, however, are those of the authors only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

Disclosure Statement

No potential conflict of interest was reported by the authors.

ORCID

Chuanren Wu http://orcid.org/0000-0002-2523-1476
Pierre David http://orcid.org/0000-0003-4837-8507
Emiliano Fable http://orcid.org/0000-0001-5019-9685
Domenico Frattolillo http://orcid.org/0000-0003-3648-8186
Massimiliano Mattei http://orcid.org/0000-0001-7951-6584
Hartmut Zohm http://orcid.org/0000-0002-8870-7806

References

1. S. KIM et al., "Full Tokamak Discharge Simulation of ITER by Combining DINA-CH and CRONOS," *Plasma Phys. Control Fusion*, **51**, *10*, 105007 (2019); https://doi.org/10.1088/0741-3335/51/10/105007.
2. M. ROMANELLI et al., "JINTRAC: A System of Codes for Integrated Simulation of Tokamak Scenarios," *Plasma Fusion Res.*, **9**, 3403023 (2014); https://doi.org/10.1585/pfr.9.3403023.
3. V. PARAIL et al., "Self-Consistent Simulation of Plasma Scenarios for ITER Using a Combination of 1.5D Transport Codes and Free-Boundary Equilibrium Codes," *Nucl. Fusion*, **53**, *11*, 113002 (2013); https://doi.org/10.1088/0029-5515/53/11/113002.
4. P. MOREAU et al., "Development of a Generic Multipurpose Tokamak Plasma Discharge Flight Simulator," *Fusion Eng. Des.*, **86**, *6–8*, 535 (2011); https://doi.org/10.1016/j.fusengdes.2011.01.013.
5. B. MAVKOV et al., "Experimental Validation of a Lyapunov-Based Controller for the Plasma Safety Factor and Plasma Pressure in the TCV Tokamak," *Nucl. Fusion*, **58**, *5*, 056011 (2018); https://doi.org/10.1088/1741-4326/aab16a.
6. F. FELICI et al., "Real-Time Physics-Model-Based Simulation of the Current Density Profile in Tokamak Plasmas," *Nucl. Fusion*, **51**, *8*, 083052 (2011); https://doi.org/10.1088/0029-5515/51/8/083052.
7. F. JANKY et al., "Simulation of Burn Control for DEMO Using ASTRA Coupled with Simulink," *Fusion Eng. Des.*, **123**, 555 (2017); https://doi.org/10.1016/j.fusengdes.2017.04.043.
8. F. JANKY et al., "ASDEX Upgrade Flight Simulator Development," *Fusion Eng. Des.*, **146B**, 1926 (2019); https://doi.org/10.1016/j.fusengdes.2019.03.067.
9. F. JANKY et al., "Validation of the Fenix ASDEX Upgrade Flight Simulator," *Fusion Eng. Des.*, **163**, 112126 (2021); https://doi.org/10.1016/j.fusengdes.2020.112126.
10. E. FABLE et al., "The Modeling of a Tokamak Plasma Discharge, from First Principles to a Flight Simulator," *Plasma Phys. Control. Fusion*, **64**, 044002 (2022).
11. E. FABLE et al., "A Practical Protocol to Emulate a Reactor Scenario on Present Machines, with Application to the ASDEX Upgrade Tokamak via Predictive Modeling," *Nucl. Fusion*, **63**, *7*, 074001 (2023); https://doi.org/10.1088/1741-4326/acd205.
12. M. SICCINIO et al., "Development of the Plasma Scenario for EU-DEMO: Status and Plans," *Fusion Eng. Des.*, **176**, 113047 (2022); https://doi.org/10.1016/j.fusengdes.2022.113047.
13. M. WALKER et al., "The ITER Plasma Control System Simulation Platform," *Fusion Eng. Des.*, **96–97**, 716 (2015); https://doi.org/10.1016/j.fusengdes.2015.01.009.
14. "Simulation and Model-Based Design," MathWorks®; https://www.mathworks.com/products/simulink.html.
15. G. V. PEREVERZEV and P. N. YUSHMANOV, "ASTRA Automated System for TRansport Analysis," Garching: Max-Planck-Institut für Plasmaphysik (2002); https://hdl.handle.net/11858/00-001M-0000-0027-4510-D.
16. E. FABLE et al., "Novel Free-Boundary Equilibrium and Transport Solver with Theory-Based Models and Its Validation Against ASDEX Upgrade Current Ramp Scenarios," *Plasma Phys. Control. Fusion*, **55**, *12*, 124028 (2013); https://doi.org/10.1088/0741-3335/55/12/124028.
17. L. E. VEEN et al., "Easing Multiscale Model Design and Coupling with MUSCLE 3," *Int. Conf. Comput. Sci.*, **12142**, 425 (2020).
18. L. E. DI GRAZIA et al., "Development of Magnetic Control for the EU-DEMO Flight Simulator and Application to Transient Phenomena," *Fusion Eng. Des.*, **191**, 113579 (2023); https://doi.org/10.1016/j.fusengdes.2023.113579.