공학박사 학위논문

Development of Integrated Suite of Codes and its Applications to Various Devices

토카막 통합 시뮬레이션 코드의 개발과

여러 장치에 대한 적용 연구

2022 년 8 월

서울대학교 대학원

에너지시스템공학부

이 찬 영

Development of Integrated Suite of Codes and its Applications to Various Devices

지도 교수 나 용 수

이 논문을 공학박사 학위논문으로 제출함
2022 년   8 월

서울대학교 대학원
에너지시스템공학부 원자핵공학전공
이 찬 영

이찬영의 공학박사 학위논문을 인준함
2022 년   6 월

Abstract

The in–depth design and implementation of a newly developed integrated suite of codes, TRIASSIC (tokamak reactor integrated automated suite for simulation and computation), are reported. The suite comprises existing plasma simulation codes, including equilibrium solvers, 1.5D and 2D plasma transport solvers, neoclassical and anomalous transport models, current drive and heating (cooling) models, and 2D grid generators. The components in TRIASSIC could be fully modularized, by adopting a generic data structure as its internal data. Due to a unique interfacing method that does not depend on the generic data itself, legacy codes that are no longer maintained by the original author were easily interfaced. The graphical user interface and the parallel computing of the framework and its components are also addressed. The verification of TRIASSIC in terms of equilibrium, transport, and heating is also shown. Following the data model and definition of the data structure, a declarative programming method was adopted in the core part of the framework. The method was used to keep the internal data consistency of the data by enforcing the reciprocal relations between the data nodes, contributing to extra flexibility and explicitness of the simulations. TRIASSIC was applied on various devices including KSTAR, VEST, and KDEMO, owing to its flexibility in composing a workflow. TRIASSIC was validated against KSTAR plasmas in terms of interpretive and predictive modelings. The prediction and validation on the VEST device using TRIASSIC are also shown. For the applications to the upcoming KDEMO device, the machine design parameters were optimized, targeting an economical fusion demonstration reactor.

Keywords: Tokamak, Plasma, Nuclear Fusion, Integrated Modeling, Integrated Simulation
Student Number: 2016–21305

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

1.1. Background

1.1.1. Fusion Reactor and Modeling

Nuclear fusion is considered as an attractive energy resource in view of sustainability, carbon neutrality, and the absence of high-level radioactive waste [1]. Realizing nuclear fusion on earth requires a container to hold high-temperature plasmas, and the tokamak is considered as the most promising magnetic confinement device concept to stably contain such plasmas for a long duration. In a tokamak, there exists a strong magnetic field applied along the toroidal direction. Owing to the Lorentz force, the toroidal magnetic could constrain the motion of charged particles and plasmas along the toroidal direction. The separation of negative and positive charges due to vertical ∇B drift motion and the resulting outward $E \times B$ drift motion could be prevented by applying an additional magnetic field along the poloidal direction (See Fig. 1.6.1 in [2]); the resulting magnetic field structure in a tokamak plasma is helical.

The helical magnetic field in a tokamak plasma gives rise to a complex geometry called flux surface. The flux surface is the layered surface where the helical magnetic field lays. The structure of the flux surface can be calculated by the balance between the electromagnetic force and the plasma pressure force. The imbalance of these forces could lead to a substantial size of instability, which is called MHD (Magneto-Hydro-Dynamic) instability. The instability can make plasma unstable and could induce substantial heat and particle loss in a short period toward the inner machine wall or even disrupts the plasma. It must be avoided or at least mitigated for the machine' s safety and stable operation, by pre-dicting the onset of the macroscopic instability beforehand.

Due to constrained motion along the helical field line, ideally, the plasmas do not move across the magnetic field. However, consid-ering the collision process between the charged particles, the plasma can be transported across the field lines and the flux sur-faces. This collisional transport is often called neoclassical transport. On the other hand, unstable modes in a plasma due to the wave-particle interaction could lead to an occurrence of a small-scale instability called micro-instability, which could eventually lead to a turbulent cross-field transport during the nonlinear phase of the instability. The origin of the micro-instability is the relax-ation of the free energy comes from the inhomogeneity of the plas-mas (e.g., the pressure gradients or the temperature gradients), which is inevitable in the tokamak plasmas where the plasmas near the machine wall should be relatively cooler while the core plasma is hotter. Calculating how much heat and particles are lost through the transport across the field line and understanding which process dominates the transport is crucial for understanding the behavior of the plasma and predicting the performance of the plasmas.

Plasmas in tokamak showed lower performance than what was expected by the (neo-)classical theory. It was believed that there exists additional "anomalous" transport all over the plasma re-gion, and substantial efforts were conducted to explain the origin of the transport. Empirically, however, it was found that the anom-alous transport is quenched to form a transport barrier near the plasma boundary when sufficient heating is applied. The plasma is then said to be operating in the H-mode (H for high), contrary to the L-mode (L for low). The steep pressure gradient formed by the transport barrier is called a pedestal. The pedestal is essential for increasing the plasma performance and eventually getting higher fusion electricity gain by increasing the fusion reaction cross−section. The H−mode operation is chosen as the baseline scenario for the upcoming international thermonuclear experimental reactor (ITER). Calculating the enhanced performance due to H−mode operation, by predicting the pedestal structure, is necessary for estimating figure of merits of the plasmas for the design of the upcoming reactor.

Plasmas should be heated up to at least 10 keV to get a non−negligible fusion reaction cross−section. Hence, there exist numerous kinds of actuators with different heating mechanisms. The heating mechanisms can be categorized into two parts, one is injecting the high−energy particles and the other is wave heating. When the plasma is sufficiently hot and dense, the charged particle products from a nuclear fusion reaction (e.g., the alpha particle for the D−T fusion reaction) would again heat the plasma so that the plasma can self−sustain its high temperature. Here, the fusion gain (Q) is the most important figure of merit regarding nuclear fusion heating and the self−sustainment of plasma. The fusion gain is defined by the ratio between the fusion−generated charged particle heating and the auxiliary heating. In view of modeling and active operation control, it is important to accurately determine the net heating and fueling applied to the plasmas by the actuators and the fusion products.

Among the actuators, the neutral beam injection (NBI) injects fast neutrals inside the plasmas and the ion cyclotron radio frequency wave (ICRF) accelerates thermal ion to become a non−thermal ion. Moreover, the alpha particle from the D−T fusion reaction should be treated as a non−thermal ion as it carries a huge amount of energy (3.5 MeV) released from the reaction and thus its velocity is much faster than the thermal plasmas. These fast−ions can drive a new set of electromagnetic instabilities that can induce additional transport to itself. Sophisticated modeling of the fast-ion-induced instabilities is crucial, as the instability can make alpha particle rapidly lost before it sufficiently transfers their energy to the background plasmas, while the alpha particle heating should be the dominant heating mechanism in the burning plasmas. Moreover, the fast-ions also can directly affect the onset/evolution of micro-instabilities originated by thermal plasmas. The fast-ions should be properly modeled and the role of fast-ions in turbulent transport of the background plasmas should be understood.

Other than the plasma heating, there also exists actuators for the particle fueling and the intentional disturbance that limits the plasma performance for stable operation. One of the latter, the resonant magnetic perturbation (RMP) applies a weak resonant magnetic field by the external coils. The RMP can suppress the edge localized modes (ELMs), which is the periodic minor disruption of the pedestal structure that occurs during the H-mode operation. While the ELMs and the energy released by the instability in a present medium-sized tokamak is not significant enough to damage the plasma-facing components (PFCs) such as divertor, it can severely damage the PFCs in the large devices such as ITER and DEMO. The ELMs must be suppressed in such devices by calculating the optimized RMP configuration for a given plasma operation condition, or another operation mode or scenario that does not damage the wall should be predicted and suggested beforehand.

Often, the plasma is operated under a diverted configuration (opposite to a limited configuration). Then, the core plasma is surrounded by the outermost magnetic field line called separatrix. The separatrix connects two diverting points, inner and outer striking points. Beyond the plasma boundary, the flux surface cannot be defined as the magnetic field line does not form a closed loop. Here, the parallel transport along the field line matters, as it would eventually meet the PFCs when following a field line. Therefore, the transport process exterior to the plasma boundary is usually treated to be two-dimensional, whereas it is treated to be one-dimensional inside the boundary. The exterior region of interest is called the scrape-off layer (SOL). In SOL, as the low-temperature plasmas interact with the PFCs, so-called plasma-wall interaction (PWI) such as recycling and sputtering occurs. Recycled neutrals and sputtered neutrals/impurities can again affect the heat and particle balance in the SOL region, and even at the core region inside the plasma boundary. It is noteworthy that there exists a maximum heat load limit for the divertor material and its cooling system. Therefore, it is crucial to predict how much heat will be loaded to the divertor through modeling, especially near the two striking points.

A high radiation loss is expected in such a hot and dense burning plasma. Electrons bounded in an ion get excited due to the collisional process with an unbounded electron, and the radiation is emitted during the spontaneous decay of this excited state. Even in the plasmas with fully stripped ions, radiations are emitted due to the recombination process and the acceleration/cyclotron motion of the electrons. Calculating radiative loss from the plasma is important, as the plasma (electrons, predominantly) can be cooled by the radiative loss and the loss is significant under a reactor condition - hot and dense, high-Z divertor material. It is also important to accurately determine the radiative loss in the SOL region, as the heat loss through radiation could reduce the heat load directly exerted on the striking points. The situation where the heat load on the striking points is minimized due to a high radiative loss in the SOL region is called detachment. The condition which enables safe operation by minimizing the heat load to the divertor without the loss of plasma performance should be found through modeling.

As was briefly introduced and shown in Figure 1, there exist plenty of different topics inside the tokamak plasma. Notably, these topics are being stems from the tokamak plasmas, separated by differentiating (1) time scales of governing equations/phenomena, (2) region of interest, and (3) terms (mechanisms) in a governing equation. Models and codes were developed to understand tokamak plasmas in such simplified situations. However, each phenomenon interacts with others to form a substantial level of complexity and nonlinearity. None of the single plasma analysis codes can fully describe the tokamak operation. Hence, the combination of dedi－cated models for each phenomenon is required to aid modelers in considering the complex interactions between different phenomena inside the plasma. Thus, the integrated modeling approach is an appropriate way to investigate these complex nonlinear phenomena self－consistently, helping us understand the physics behind them. Ultimately, a well－validated integrated modeling tool can be uti－lized to predict and further optimize future tokamak devices such as ITER of DEMO to significantly contribute to the realization of the magnetic confinement fusion reactor.

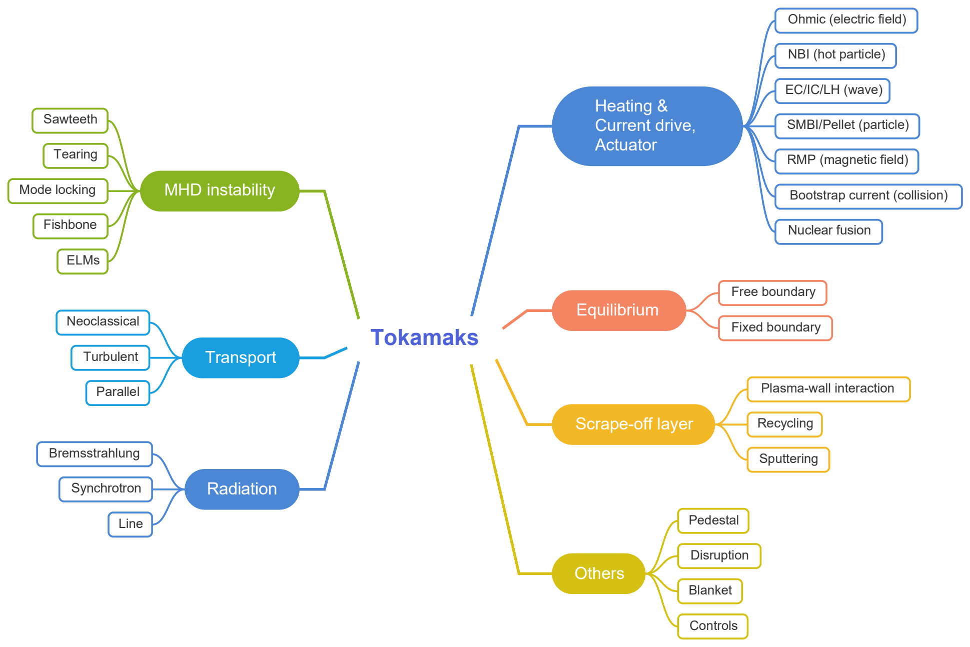


Figure 1. Multiple topics (physics phenomena, actuators, and mod−eling) that are inherent in tokamak plasmas (See chapters in [2]).

1.1.2. Interpretive Analysis and Predictive Modeling

The interpretive and predictive analyses are the type of analysis that utilizes a set of modeling codes/tools. The interpretive analysis targets to interpret the detailed mechanisms or physics behind the observations, such as measurement or phenomenon. For instance, in Figure 2 (left), the interpretation of beam absorption and neutron emission is shown. The plasma stored energy in this experiment was measured by the plasma equilibrium reconstruction by meas−uring the magnetic field from the external magnetic field coils. There is no way to directly measure the contribution of fast−ion energy among the total stored energy. However, the contribution of fast−ion can be calculated by the neutral beam modeling code. Here, the measured stored energy was compared with its modeling counterpart calculated by summing every contribution of plasma species. On the other hand, in the lower panel, the measured neu-tron emission rate was compared with the calculated neutron emission rate from the neutral beam code. It is noteworthy that there exist three different sources of neutron generation mecha-nism (beam-target, beam-beam and thermonuclear) and the in-terpretive analysis shows that the neutrons generated from the beam-target fusion reaction dominate the others, while it is hard to be inferred only from the measurements. Also, from the con-sensus between the experimental measurements and the modeling results in terms of energy and neutron emission rate, it can be concluded that the beam modeling is reliable.

The interpretive analysis becomes more useful and richer when several more codes were integrated. Figure 2 (right) shows the interpretation of ion heat transport. The purple shaded line indi-cates the experimental heat flux. The experimental heat flux can be calculated by solving the ion heat transport equation, after knowing how much net ion heating (and cooling) is deposited and how the ion energy (ion density and temperature) evolves in time. The black region and the red line indicate the neoclassical heat flux and turbulent heat flux, respectively. Overall, the analysis includes equilibrium solver, transport solver, heating models, and neoclas-sical and turbulent transport models (gyrokinetic code, GTS [3]). From the analysis, a plausible conclusion can be made: The exper-imental ion heat transport can be explained by the neoclassical and turbulent heat transport model, and the turbulent transport domi-nates on the outer part of the plasma, while the neoclassical transport dominates on the inner part. If one of the models is absent, the conclusion could not be made; the integration of codes in in-terpretive analysis can aid a deep and comprehensive understand-ing of tokamak plasmas.

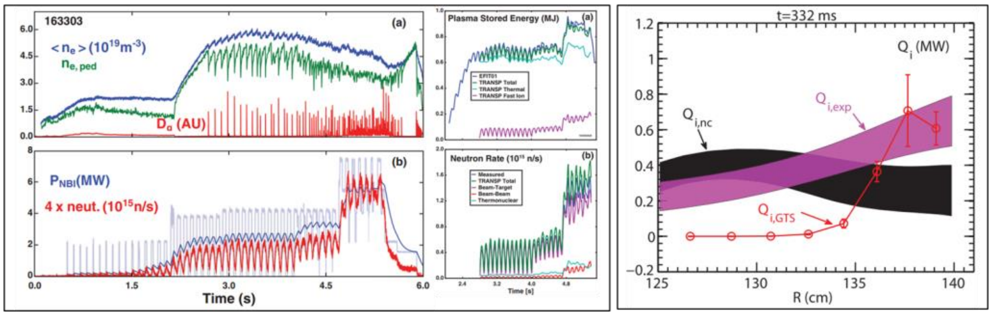


Figure 2. Interpretation of the origin of fusion neutron generation (left) and interpretation of heat transport and its mechanism (right). Figures were reproduced from [4, 5], respectively.

Predictive modeling, distinct from interpretive analysis, focuses on predicting thermal plasma quantities such as density, temperature, and rotation. Thus, the predictive modeling includes a transport solver that solves the transport equation time-dependently or the solver that gives a stationary-state transport solution. The examples of predictive modeling are shown in Figure 3 (two on the upper). The time-dependent prediction of electron and ion temperature with GLF23 [6] and TGLF [7, 8] model in DIII-D tokamak are shown on the left, and the stationary-state prediction solution of density, electron and ion temperatures, angular rotation frequency by QuaLiKiz [9] and TGLF model in the JET tokamak are shown on the right. Note that, although the modeling is represented by the name of the turbulent transport model owing to its importance, predictive modeling should consist of a fully integrated set of models.

The predictive modeling applied to the present device is meaningful as the models, especially the transport models, can be validated by the modeling. For instance, it can be said that the turbulent transport models are validated on the given discharges of the DIII-D and the JET cases shown in Figure 3. Moreover, if the validation is conducted for a multi-discharge or even a multi-device, in principle the integrated suite can be exploited to predict upcoming devices such as ITER or DEMO. However, it is noteworthy that, although the models are validated on the present device, it is not sure whether the models would work properly when applied to future devices. For example, the empirical models or the neural network models, without a proper theoretical understanding or proper input/output data normalizations, would fail when extrapolated. To avoid the issue, theory-based models should be utilized. Even the theory has a validity regime and might break down when applied to large-size, burning plasmas. Hence, the results should be carefully reviewed when extrapolated. Also, the models should be tested on a broad range of parameters by applying to multiple devices from small devices to large devices.

In Figure 3 (lower), the prediction of ITER 12.5 MA hybrid operation and the prediction of temperature profile are shown for the initial phase and the main burn phase, while the density profiles are prescribed. Predictive modeling can give an estimation of future devices, such as estimated temperature, fusion power, neutron generation, and non-inductive current drive fraction. By the results, the device design or the operation scenario can be optimized to reach the target parameters, while reducing the construction cost and risk. Further, the predictive modeling tool can be utilized to generate a digital twin for the upcoming reactor, which will significantly improve the understanding of the reactor construction and its operation. For this, the code suite should be able to consider numerous kinds of phenomena not only for the plasma but also for the machine itself. Many codes/models should be integrated and their calculation and coupling should be extensively validated.

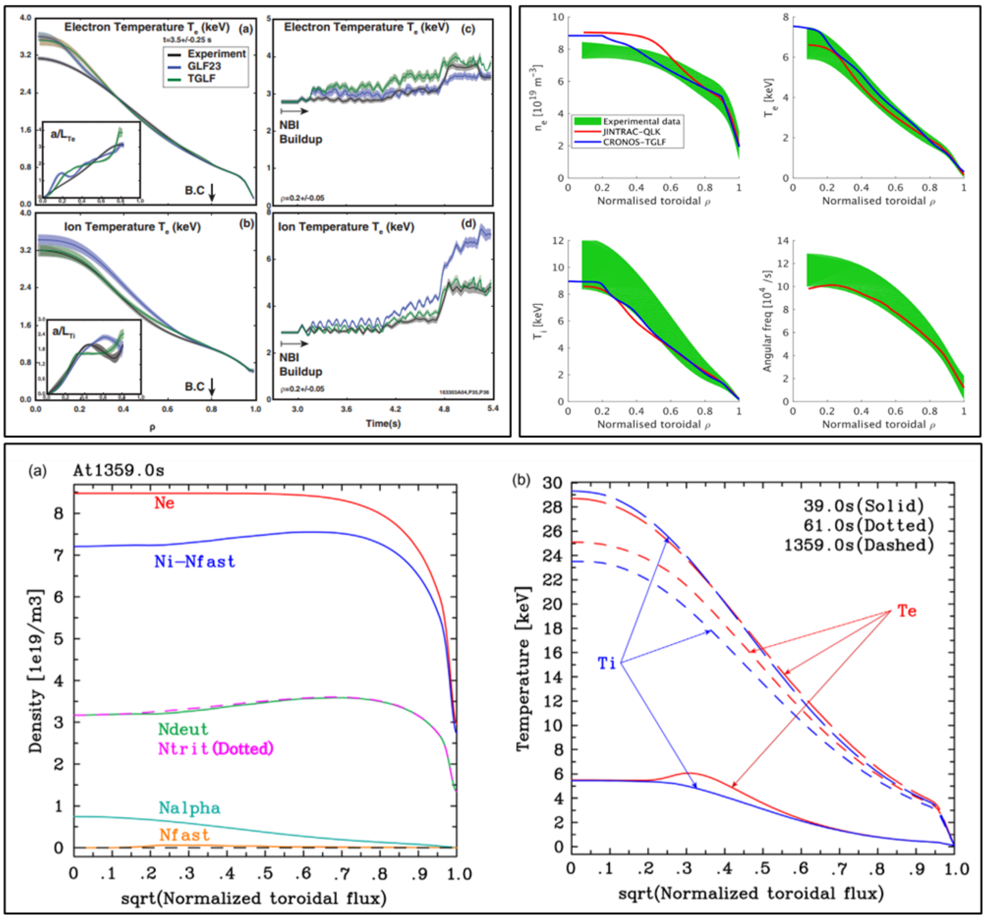


Figure 3. Time−dependent prediction of DIII−D plasmas (upper left), prediction of stationary phase in JET plasmas (upper right), and prediction of ITER 12.5 MA hybrid discharge (lower). Figures were reproduced from [4, 10, 11], respectively.

1.1.3. Modular Approach

Code integration is essential for integrated modeling. In order to integrate the codes, two ways of code integration methods − non−modular and modular approaches − are possible as illustrated in Figure 4. The non−modular approach couples the codes directly with each other. Examples shown in Figure 4 shows that code A, B, and C are mutually coupled, or one of the codes, A, is coupled to the rest of the codes, B and C in the non-modular approach. On the other hand, the codes are not coupled to each other in the modular approach. Instead, codes are coupled to the centralized framework, as shown on the right of Figure 4.

The former has big merit compared to the latter when implementing an implicit method or a synergetic effect. This is because several iterations between the code are required when implementing an implicit method or to see the synergetic effect. Also, owing to the direct coupling between the code, it can be faster than the modular approach. However, adding a new code requires many interfaces between the codes. For instance, if code D is newly adopted and needs to be used within the code group shown at the left of the non-modular approach in Figure 4, it requires at most three more code interfaces, to be coupled with every code in the suite.

The problem can be avoided if the case is the same as the situation at the right of the non-modular approach in Figure 4, where code A acts like a centralized framework. In this case, code D can be interfaced only with code A, without the direct link between codes B or C and satisfying the modularity requirement [12]. However, in this case, as codes are not fully separated, the coupled codes can only be executed through the central code, A. In other words, it is inevitable to use code A, even when only code B is required. Moreover, the functionality of code A becomes ambiguous, as code A should handle the execution of other codes and should properly hold data given by other code components, B, C, and D. Legacy transport frameworks such as TRANSP [13], JETTO [14], and ASTRA [15] correspond to this case. Although the codes adopt the modular approach for the models other than the transport solver, the transport solver or interpreter cannot be separated when composing a workflow. Also, it is almost impossi-ble to switch the transport solver to another.

The modular approach requires a centralized framework. The codes and modules interact with the centralized framework as shown in Figure 4. Such interaction requires proper data storage used in the framework; the data storage will be explained in the next section. In this approach, unlike the previous approach, there is no overlap between the codes and thus replacing a code compo-nent with another is much easier. For example, if code D should be interfaced, it would only be interfaced with the framework. More-over, if code D does the same calculation as code C but with a dif-ferent model/method, code D can fully replace the role of code C. Also, if a generic data dictionary or data model is defined in the framework, the framework can utilize the full functionality of each code by taking a full set of useful information from the code. This is not the case for the non-modular approach, as each code has a limited variable definition that fits its usage.

Note that there is no absolute advantage between the two ap-proaches. The non-modular approach is rigid but has an advantage on the computation performance. On the other hand, the modular approach merits its versatility by enabling code component re-placement and full functionality utilization. However, considering the current progress of the integrated simulation research, still, the physics codes and models need to be verified with each other and validated against the experiments. There is no complete set of modeling codes that are fully validated. Thus, the interchangeabil-ity of the modular approach is considered as the biggest merit, and the approach will be considered as the better approach in this re-search.

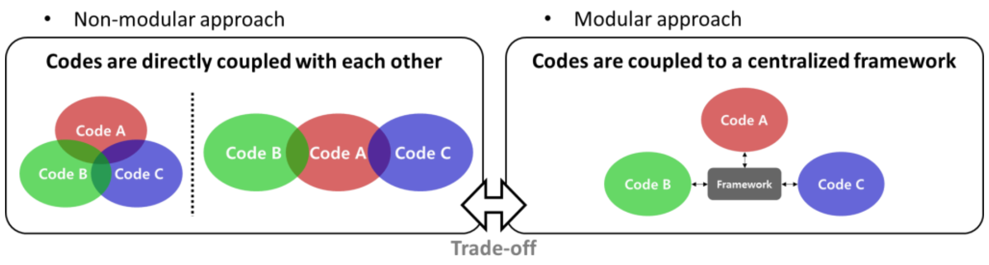


Figure 4. The schematic view and comparison of the non−modular approach and the modular approach.

1.1.4. The Standard Data Structure

In Figure 5, although the center of the framework in the example is not the framework itself but the core transport solver, the frameworks are sufficiently modularized as discussed in the pre−vious section. In such frameworks, it can be noticed that the sim−ulations are conducted by transferring the tokamak and plasma data between the transport solver and the other components. The transport solver is positioned at the center of the framework and the data transferred from/to the other components is represented as arrows. The data would exist in the internal variable space of the transport solvers during the simulation and be written as a file at the end of the simulation. The output format is called $\mathit{Simulation\ output}$ and $\mathit{Plasmaprofiles}$, respectively as shown in the figure.

The verification between the codes and the framework requires cumbersome tasks. For example, when the two frameworks are being verified against each other, as the output format differs be−tween the simulation frameworks as was discussed in the previous paragraph, it requires an additional conversion to a unified format for direct comparison. Also, even though the frameworks have the same analysis target, as each framework has a different imple- mentation structure and an input format, it is hard to make simu- lation settings identical. An example of a benchmark between the transport solvers for the ITER particle transport and a significant amount of effort can be found in the previous study [16]. It casts a need for the definition of the standard data model used in the simulation framework.

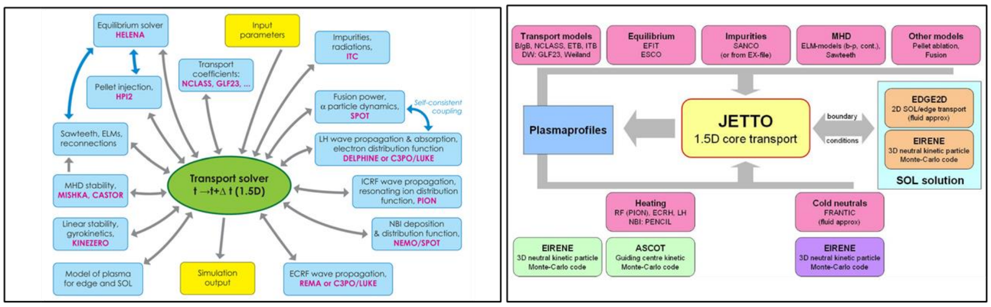

Equilibrium solver HELENA
Input parameters
Impurities, radiations, ITC
Transport coefficients: NCLASS, GLF23, ...
Pellet injection, HPI2
Fusion power, α particle dynamics, SPOT
Self-consistent coupling
Sawteeth, ELMs, reconnections
LH wave propagation & absorption, electron distribution function DELPHINE or C3PO/LUKE
Transport solver t→t+Δt (1.5D)
MHD stability, MISHKA, CASTOR
ICRF wave propagation, resonating ion distribution function, PION
Linear stability, gyrokinetics, KINEZERO
NBI deposition & distribution function, NEMO/SPOT
Model of plasma for edge and SOL
Simulation output
ECRF wave propagation, REMA or C3PO/LUKE
Transport models B/gB, NCLASS, ETB, ITB DW: GLF23, Weiland
Equilibrium EFIT ESCO
Impurities SANCO (or from EX-file)
MHD ELM-models (b-p, cont.), Sawteeth
Other models Pellet ablation, Fusion
Plasmaprofiles
JETTO 1.5D core transport
boundary conditions
EDGE2D 2D SOL/edge transport (fluid approx)
EIRENE 3D neutral kinetic particle Monte-Carlo code
SOL solution
Heating RF (PION), ECRH, LH NBI: PENCIL
Cold neutrals FRANTIC (fluid approx)
EIRENE 3D neutral kinetic particle Monte-Carlo code
ASCOT Guiding centre kinetic Monte-Carlo code
EIRENE 3D neutral kinetic particle Monte-Carlo code

Figure 5. The schematic view of the CRONOS suite of codes [17] (left) and the JINTRAC (JETTO) system of codes [18] (right). Figures were reproduced from the references.

Figure 6 shows the merits of using a standard data model in simulations and experiments. As aforementioned, the standard data model can aid the verification of frameworks and codes. The same structure of input will greatly help running identical settings of simulation and the same structure of output will ease comparing the simulation results. When the experimental data is provided in the standard data format, two additional advantages other than the verification are gained. First, the validation of the framework or the code gets easier as the simulation results can be directly compared with the experimental data. Second, a detailed analysis of plasma or discharge becomes convenient as the experimental data can di- rectly be used in the framework or the code, without the compli- cated and error-prone task of making input files.

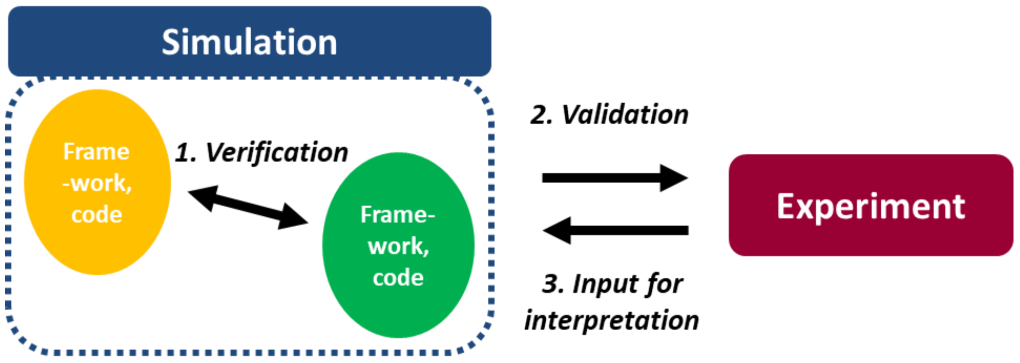
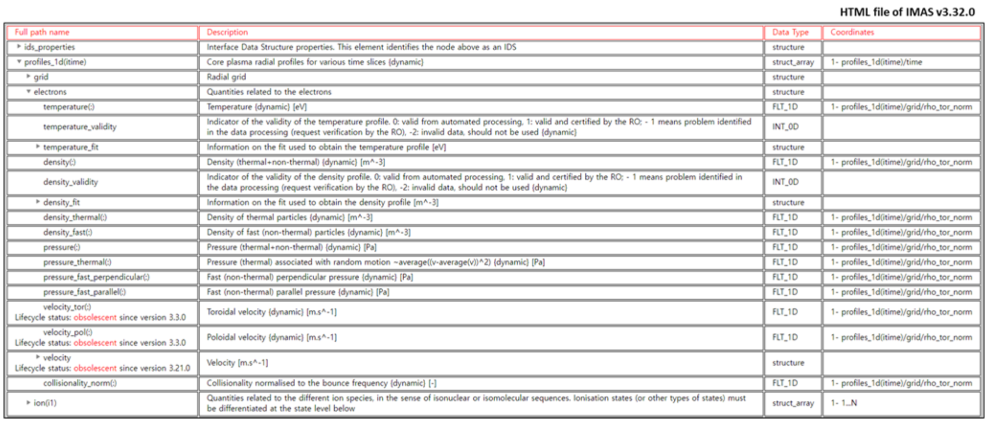


Figure 6. The merits of using standard data in simulations and experiments.

As the first design of a standard data model, the consistent physical object (CPO) [12] was developed. It was then improved in the ITER Integrated Modeling & Analysis Suite (IMAS) [19] with a formal definition of a standard data structure, so-called the interface data structure (IDS), which can be used to exchange data within an integrated simulation framework. The IDS is the format in which future ITER experiments and modeling results are provided. IDS is widely adopted when developing new integrated simulation frameworks [20-23].

The IDS is generic; it ought to contain device information of any tokamak, widely known diagnostic and actuator information, and plasma data from simulation and analysis. Figure 7 shows a part of the documentation of *core_profiles* IDS, where in total there exists around 70 separate IDSs in the full IMAS data model. The documentation is well-organized, by showing the description, data type, and coordinates of a given data node. The structure is visualized in Figure 8. It can be noticed that the structure is hierarchical, which eases researchers to understand the data structure. The *core_profiles* IDS can hold 1D profiles of core plasma quantity and some global (0D) quantities such as plasma current and loop voltage in time. In *profiles_1d* array of structure (AoS), there exists placeholders allotted for the information of species like electron and generic ion. The species can hold density and temperature, as in general, the temperature is not identical for two different ion species. A more detailed design of IDS can be found in the previous study [19].



HTML file of IMAS v3.32.0

Full path name	Description	Data Type	Coordinates
▸ ids_properties	Interface Data Structure properties. This element identifies the node above as an IDS	structure	
▾ profiles_1d(itime)	Core plasma radial profiles for various time slices {dynamic}	struct_array	1- profiles_1d(itime)/time
▸ grid	Radial grid	structure	
▾ electrons	Quantities related to the electrons	structure	
temperature(:)	Temperature {dynamic} [eV]	FLT_1D	1- profiles_1d(itime)/grid/rho_tor_norm
temperature_validity	Indicator of the validity of the temperature profile. 0: valid from automated processing, 1: valid and certified by the RO; - 1 means problem identified in the data processing (request verification by the RO), -2: invalid data, should not be used {dynamic}	INT_0D	
▸ temperature_fit	Information on the fit used to obtain the temperature profile [eV]	structure	
density(:)	Density (thermal+non-thermal) {dynamic} [m^-3]	FLT_1D	1- profiles_1d(itime)/grid/rho_tor_norm
density_validity	Indicator of the validity of the density profile. 0: valid from automated processing, 1: valid and certified by the RO; - 1 means problem identified in the data processing (request verification by the RO), -2: invalid data, should not be used {dynamic}	INT_0D	
▸ density_fit	Information on the fit used to obtain the density profile [m^-3]	structure	
density_thermal(:)	Density of thermal particles {dynamic} [m^-3]	FLT_1D	1- profiles_1d(itime)/grid/rho_tor_norm
density_fast(:)	Density of fast (non-thermal) particles {dynamic} [m^-3]	FLT_1D	1- profiles_1d(itime)/grid/rho_tor_norm
pressure(:)	Pressure (thermal+non-thermal) {dynamic} [Pa]	FLT_1D	1- profiles_1d(itime)/grid/rho_tor_norm
pressure_thermal(:)	Pressure (thermal) associated with random motion ~average((v-average(v))^2) {dynamic} [Pa]	FLT_1D	1- profiles_1d(itime)/grid/rho_tor_norm
pressure_fast_perpendicular(:)	Fast (non-thermal) perpendicular pressure {dynamic} [Pa]	FLT_1D	1- profiles_1d(itime)/grid/rho_tor_norm
pressure_fast_parallel(:)	Fast (non-thermal) parallel pressure {dynamic} [Pa]	FLT_1D	1- profiles_1d(itime)/grid/rho_tor_norm
velocity_tor(:) Lifecycle status: obsolescent since version 3.3.0	Toroidal velocity {dynamic} [m.s^-1]	FLT_1D	1- profiles_1d(itime)/grid/rho_tor_norm
velocity_pol(:) Lifecycle status: obsolescent since version 3.3.0	Poloidal velocity {dynamic} [m.s^-1]	FLT_1D	1- profiles_1d(itime)/grid/rho_tor_norm
▸ velocity Lifecycle status: obsolescent since version 3.21.0	Velocity [m.s^-1]	structure	
collisionality_norm(:)	Collisionality normalised to the bounce frequency {dynamic} [-]	FLT_1D	1- profiles_1d(itime)/grid/rho_tor_norm
▸ ion(i1)	Quantities related to the different ion species, in the sense of isonuclear or isomolecular sequences. Ionisation states (or other types of states) must be differentiated at the state level below	struct_array	1- 1...N

Figure 7. A part of the documentation of *core_profiles* IDS. The IDS is well−documented with a description, data type, and coordinate.

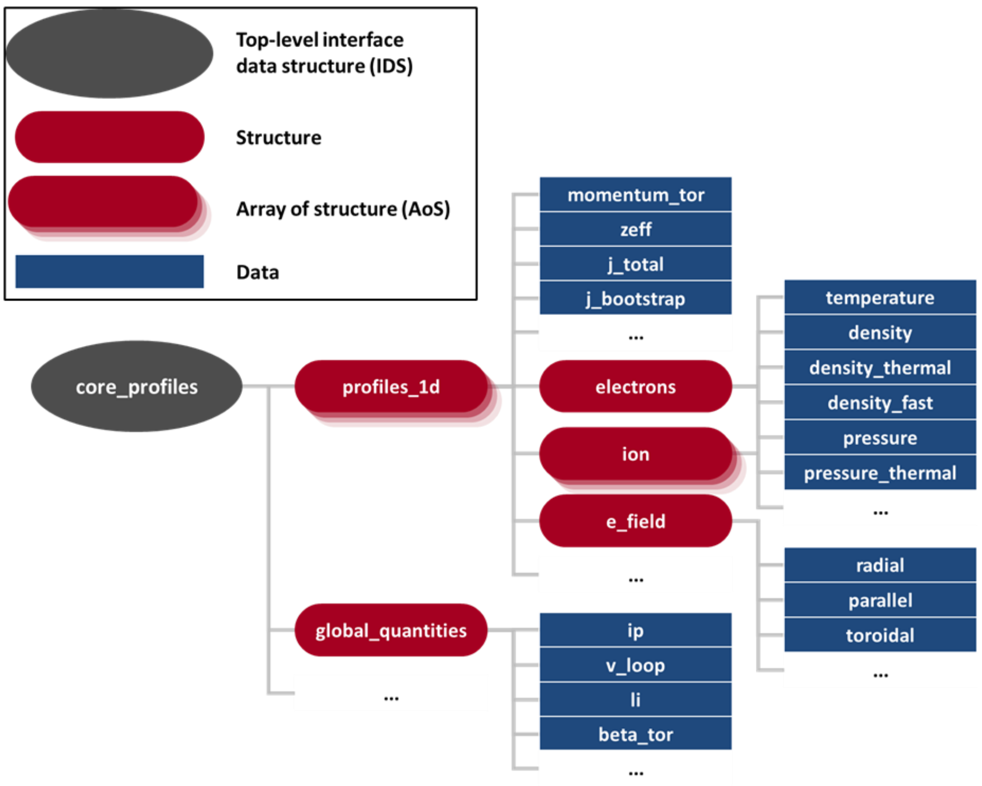


Figure 8. A part of the structure of *core_profiles* IDS. The IDS is generic and hierarchical.

1.1.5. The Internal Data Consistency in a Generic Data

Some data nodes in an IDS can be expressed by other data nodes, due to their definitions or constraints. For example, in Fig－ure 8, the *zeff* in *profiles_1d* can be expressed by the combina－tion of the electron density, ion densities, and ion charge (*z_ion_1d*, not shown in the figure) following the definition of the data nodes. Also, the *density* (total density of a specie) can be expressed by the summation of *density_thermal* and *density_fast* for every spe－cie. The relations are reciprocal; not only the *density* but also the *density_thermal* and *density_fast* can be calculated when the other two are given. Moreover, when considering the quasi-neutrality constraint which is the basic property of the plasma, the electron density, ion densities, and ion charges can form a reciprocal relation.

It is noteworthy that the relations are not limited to the *core_profiles* IDS. The relations exist within other IDSs, such as *equilibrium*, *core_sources*, *core_transport*, and even between the IDSs. For instance, one of the *source* or *model* AoS in *core_sources* or *core_transport* IDS should contain the total heating and current drive source or the total transport amount. In other words, data nodes in a *source* or a *model* AoS are related. Moreover, the electric conductivity, which is usually calculated by the neoclassical transport model, is natural to be stored in the *core_transport* IDS. However, the *core_profiles* IDS also has the node named *conductivity_parallel* for the interpretation of parallel electric field and current density. The two quantities should be consistent considering different grids in the two IDSs. Also, the non-inductive current drive sources calculated by the current drive source models should be properly considered to calculate *j_bootstrap* and *j_non_inductive* in *core_profiles* IDS.

It is obvious that erroneous simulation results would appear if the consistency in the centralized data is not kept. The situation gets even worse when the user cannot notice the erroneous simulation result. Moreover, it is hard for a simulation code developer to pinpoint the origin of the error and to debug the code when the error originates from broken consistency. Therefore, keeping the internal data consistency is crucial.

1.1.6. Integration of Physics Codes into IDS

The previous research [23] categorizes the physics code integration into three tiers, as shown in Figure 9. The tier 1 approach fully embraces the adoption of the IDS. The IDS participates not only in the I/O (input/output) routine but also in the physics routine. The data stored in the IDS is directly used and written by the physics routine. The tier 2 approach has a difference in communication between the IDS and the physics routine. In this approach, the IDS is only accessed by the I/O routine. It should be noted that as the I/O routine contains the IDS in the tiers 1 and 2 approaches, the IDS libraries that can be imported in the given code language are required.

In the tier 3 approach, on the other hand, the IDS is fully separated from the physics code. Note that the medium of the coupling, which can be one of file I/O or routine I/O, does not differentiate the tiers. As is demonstrated by the colors of the data node in an IDS in Figure 9, the consistency of the data is not guaranteed in the tier 3 approach, whereas the consistency is kept by the I/O routines in tiers 1 and 2. This also indicates that, when an original physics code is given, the I/O routine of the physics code should be significantly modified to fill in the data nodes in an IDS with proper values and keep the consistency between the data nodes. On the other hand, the tier 3 approach preserves the original code itself without or with minimal modification of the I/O routine.

The tier 3 approach is preferable to the other approaches when interfacing legacy codes that are no longer supported by the code developer. After knowing some distinctive output data from the legacy physics code, the code can be interfaced easily by updating a few IDS nodes that correspond to the outputs. However, as was previously noted, it requires some further calculations to keep the data consistency.

Tier 2 approach is usually taken by the EU-IM codes, such as ETS [21] and H&CD workflow [24]. The physics code and their modified I/O routine that embraces the IDS is called an IMAS actor. For the tier 3 approach, the OMFIT [23] is the most appropriate example. The output of the physics code is originally stored by the inherent format and then converted to the IDS format through the Python library called OMAS [25].

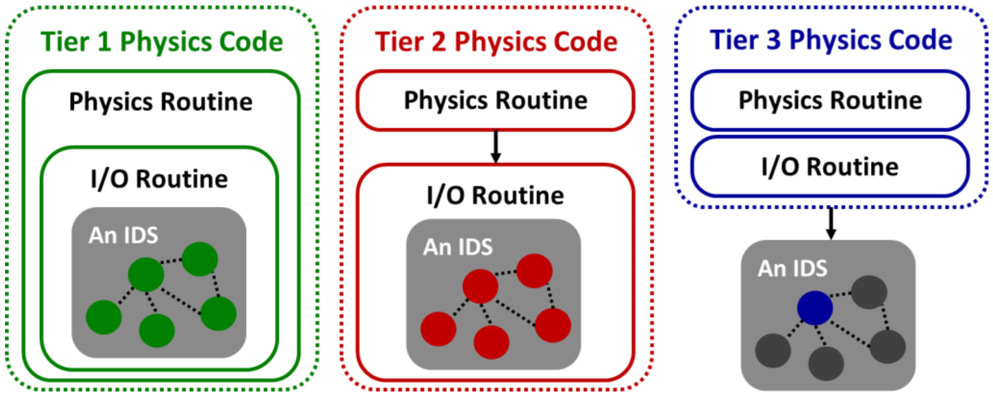


Figure 9. Three levels of integration of physics codes into IDS [23]. The physics codes belonging to tiers 1 and 2 contain IDS in their I/O routines. On the other hand, the tier 3 physics codes are separated from the IDS. In terms of data consistency, while tiers 1 and 2 guarantee data consistency in an IDS as shown in green and red circles, the tier 3 approach does not guarantee data consistency within an IDS.

1.2. Overview of the Research

The research is associated with the development of a new integrated simulation framework, TRIASSIC [26]. TRIASSIC was designed to perform various interpretive and predictive analyses. A full modular approach was adopted for interfacing the code components in the framework; thus, the replacement is convenient for every code component even for the transport solver. Full modularization was enabled by adopting the interface data structure as a main data storage for the simulation framework. It also gave merits by standardizing code inputs and outputs, making validation and verification much easier. The framework chooses the tier 3 approach to interfacing the codes and so the internal data consistency of the generic data structure was kept by the framework.

The research is composed as follows. In chapter 2, a detailed design, comparison with other frameworks/workflows, and code description will be shown. The verification of TRIASSIC with standalone code execution will also be shown. In chapter 3, the possible problems in the framework related to data consistency will be raised and the solutions implemented in TRIASSIC will be addressed. In chapter 4, the application of TRIASSIC on various devices - KSTAR, VEST, and KDEMO - will be addressed in the aspects of various workflows viable in various devices. Finally, in chapter 5, the summary and conclusion will be drawn, and future works will be suggested.

Chapter 2. Development of Integrated Suite of Codes

2.1. Development of TRIASSIC

2.1.1. Design Requirements

Full modularization was considered as an essential feature for a new integrated simulation framework, as was discussed in section 1.1.3. It is noteworthy that full modularization implies the separa-tion of code and minimization of each code functionality. In other words, the functionality of a code component should be minimized for full modularization. For instance, if a coupled set of a transport solver and a transport model is integrated into a framework, the codes are not fully modularized; they should be separated and in-terfaced independently for full modularization. This way, the entire useful output from each code can fully be utilized.

The minimization of functionality reduces code duplication and possible errors from the problem. One of the main examples of minimization can be the separation of the pedestal model and the ELM model, as the pedestal structure seemingly ought to be de-termined by a unified pedestal/ELM model. While the pedestal width and height are predicted by the pedestal model, the ELM model determines the plasma transport at the pedestal region. The pedestal structure can be determined by various models [27-29], and the transport phenomena at the pedestal region can be mim-icked by various methods such as the direct or delayed assignment of a given pedestal shape, continuous ELMs [30], and transient ELMs [18, 31]. The choice of the pedestal model and ELM model should be independent, hence the two models should be separated. Other codes that require minimization – equilibrium and transport, transport solver and transport model, 2D grid and 2D plasma transport, and plasma heat exchange and transport solver – are all separated in TRIASSIC.

As was previously noted, the use of the standard interface is a necessary condition for the fully modularized simulation framework. It also gives plenty of merits on experimental validation and verifications. Moreover, a generic, machine-independent data structure of IDS made the framework applicable to any tokamak device. The use of the standard interface (IDS) was considered as one of the most important requirements.

The simulation framework should be user-friendly; a user should be able to compose a workflow and understand the code easily. For user-friendliness, the graphical user interface (GUI) can be utilized to organize a workflow. The GUI should help understand the input parameters for the simulation workflow and code components. The data fed into the simulation should be easily treated, even for the arrays of 1D time-varying scalar, 2D time-varying profile, and 3D time-varying contour data. The GUI should also help deal with the multi-dimensional data by copy-paste and by checking the data fed into the application by visualization. The functionality that loads data from a well-known file format such as G-EQDSK (file format for storing the tokamak plasma magnetic equilibrium data) should exist to help make simulation input. The data should be able to be saved by the input file in a human-readable form. Finally, the simulation code should be in a form that is easy to read. The use of high-level language can substantially enhance the readability of the code.

A good performance of the simulation is also a crucial virtue of the integrated simulation framework. The plasma analysis codes should be interfaced directly rather than a file-based coupling, on exchanging the data with the internal data storage. The file-based coupling can cause significant file I/O (input/output) resulting in a high computation load on the simulation server. Moreover, the framework should provide parallel computing of the code components. Code components which have native support of the MPI (message passing interface) should be able to be implemented in a parallel way, and the other way of parallelization should be implemented for the codes without the native MPI support.

The code suite should have minimal dependency on the system. The compiling system of the code suite should not depend on the compiler vendor, and the simulation should be able to be conducted even when the simulation workflow is composed of codes with different compiler vendors, as the simulation workflow is managed by a high-level language interpreter such as Python. Also, the simulation should not depend on the other program other than the main executor. Finally, the input and the output of the simulation should be straightforward. The simulation should not require any extra input file other than the designated, nor give extra output other than the designated data structure, except for the special case.

2.1.2. Overview of TRIASSIC

The TRIASSIC is an acronym for Tokamak Reactor Integrated Analysis System for SImulation and Computation. TRIASSIC is a Python framework containing a group of interfaces for existing plasma analysis codes. It targets comprehensive interpretive/predictive tokamak simulations.

TRIASSIC comprises four components: The simulation (work-flow) manager, IDS from ITER-IMAS, GUI, and physics code components, as shown in Figure 10. Here, the IDS was adopted as the internal data storage of the simulation framework. On the simulation initialization, an IDS can be generated via using the data in user-provided input files, or the simulation can start without generating an IDS when a reference IDS is given. The functionality of generating a new IDS is required to support using individual post-processed data and using old data that are already stored in another format.

The code components that are interfaced in the framework will be briefly explained with references in section 2.2.1. The GUI was developed to aid in generating the input files. The GUI can also fetch useful plasma equilibrium data from external formats such as G-EQDSK files and can get up to 2D (spatiotemporal) information from copy-paste. A more detailed explanation of GUI will be addressed in section 2.2.3. Finally, the role of the framework itself and a more detailed description of the internal re-calculation routine of the core of the framework will be addressed in the following paragraph and chapter 3.

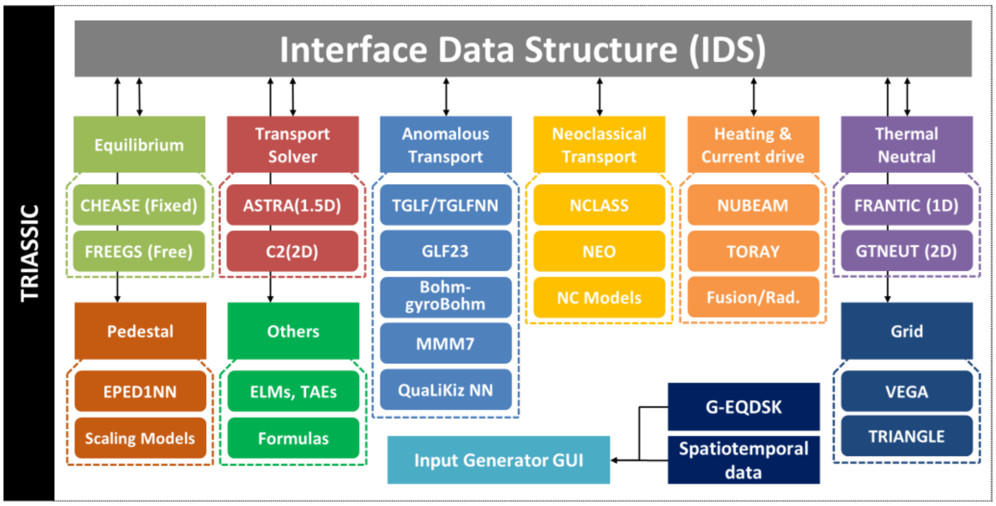


Figure 10. The schematic view of TRIASSIC integrated simulation framework. TRIASSIC incorporates the interface data structure as its data storage, GUI which eases the input generation, and the plasma analysis code components.

Figure 11 shows the input files used in TRIASSIC and the over–all simulation flow. The simulation can be started if and only if the three input files are prepared: ***data***, ***task***, and ***sim***. Those are the Fortran namelist formatted human–readable plain text files. The reason for adopting the ASCII type Fortran namelist format as an input file format is to ease writing and putting various data types, ranging from characters or strings to the 2D spatiotemporal float–ing–point data. This namelist formatted data can easily be con–verted to a Python dictionary by an external module. The input files can be generated either manually or by the graphical user interface.

The ***data*** file consists of the shot and run information that de–termines where to save the IDS datafile and the specific identifi–cation of the reference IDS from where to start the simulation. It can also contain some of the frequently used plasma and actuator quantities such as densities, diffusivities, and NBI power/energy and let the simulator fill the time-varying data into the IDS as prescribed. Although this is not a standard way of using IMAS/IDS in the integrated modeling approach, in this way, a user can freely put the simulation data without the need for existing IDS data. This non-standard approach differentiates the TRIASSIC from other modeling frameworks and made modeling much easier when evaluating a new actuator design or predicting a device that does not exist yet.

The ***task*** file contains general options for the simulation and code-specific options for each component. The former includes the generic information for the overall simulation flow, such as the number of default radial computation grids, plasma ion species configuration, or from which ion species the radial electric field would be calculated. The task-specific option specifies the exact functionality of a component, such as a switch that determines whether to solve the heat transport equation or to follow the experimental data. These options must be well-defined and should have clear objectives.

Lastly, the ***sim*** file contains invocation settings for all the components specified in the ***task*** file. Any kind of simulation whose component call is periodic in time can be designed by the sim file. The invocation of each component is determined by choosing the start time, end time, invocation interval, and which interface to be called. Additionally, there is a rank system to determine the order of execution when there are two or more components at the same invocation time. It explicitly specifies which one has the higher priority of invocation.

Based on the ***data*** file, TRIASSIC determines where to find the input data among the existing reference IDS datafile and the input file. If a reference IDS is given, TRIASSIC reads the data from IDS by the *get_slice* operation. After that, TRIASSIC sets a plan for the simulation based on the simulation time configuration written in the *task* file and the invocation setting written in the *sim* file. During the time advancing, TRIASSIC invokes the Python interface by giving the tokamak or plasma data contained in IDS with some task-specific simulation options. The Python interface then translates the standard data format to a driver-specific format of the code. After the code calculation, the result is again translated to a standard data format and used to update the IDS data.

Updating the IDS data requires a further re-calculation, such as re-calculating the total pressure, or the radial electric field followed by an update on the electron temperature or the poloidal rotation. The internal data consistency of the simulation is kept by re-calculating the IDS data based on the general simulation options after every component call. The details of this internal data consistency will be addressed in the following chapter. For every data save interval, the simulation data is stored as an IDS datafile by the *put_slice* operation.

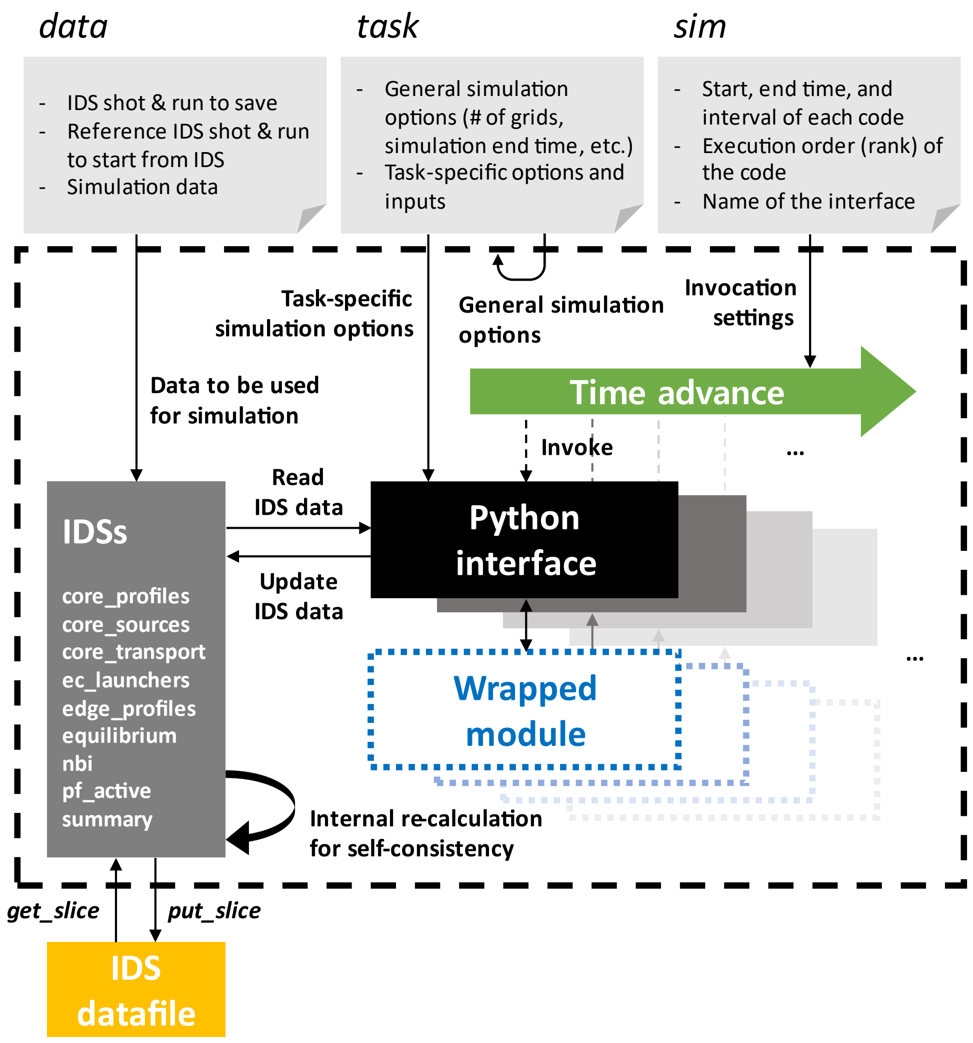

data
task
sim
- IDS shot & run to save
- Reference IDS shot & run to start from IDS
- Simulation data
- General simulation options (# of grids, simulation end time, etc.)
- Task-specific options and inputs
- Start, end time, and interval of each code
- Execution order (rank) of the code
- Name of the interface
Task-specific simulation options
General simulation options
Invocation settings
Data to be used for simulation
Time advance
Invoke
...
Read IDS data
Update IDS data
IDSs
core_profiles
core_sources
core_transport
ec_launchers
edge_profiles
equilibrium
nbi
pf_active
summary
Python interface
Wrapped module
...
Internal re-calculation for self-consistency
get_slice
put_slice
IDS datafile

Figure 11. The overall simulation flow of TRIASSIC. The simulation can be controlled by three input files – *data*, *task*, and *sim*. TRIASSIC orchestrates the execution of each component with time advance, while the component interfaces communicate with IDS for its input and output data.

2.1.3. Comparison of Integrated Simulation Codes

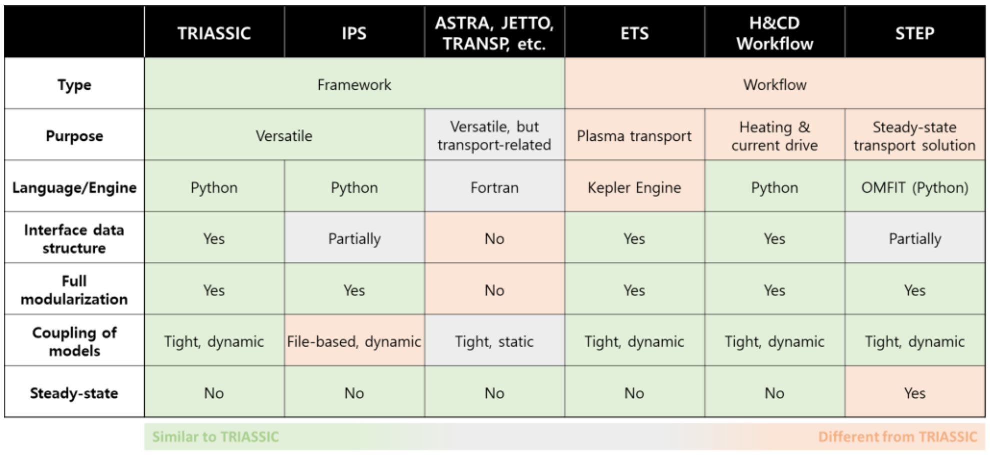
Figure 12 shows the comparison of TRIASSIC and other frame-works such as IPS (integrated plasma simulator) [32], ASTRA [15], JETTO [14], and TRANSP [33] or workflows such as ETS (European transport simulator) [21], H&CD workflow [24], and STEP [23]. While the workflow has a specific purpose for the sim-ulation, the framework should be able to compose various work-flows to investigate a variety of phenomena in tokamak plasmas. Therefore, while ETS, H&CD workflow, and STEP target to inves-tigate plasma transport, heating & current drive, and steady-state transport solution, respectively, the frameworks are versatile. Note that the transport codes such as ASTRA, JETTO, and TRANSP are deeply related to plasma transport, so they do not satisfy the full modularization addressed in section 1.1.3. In other words, those frameworks cannot conduct other (heating & current drive or neutral modeling, etc.) simulations solely without con-ducting the transport steps.

Note that the OMFIT (one modeling framework for integrated tasks) [22], which is one of the most famous integrated simulation codes, was not included in the comparison as the simulation through OMFIT accompanies other frameworks or workflows such as IPS, TRANSP, or STEP. In other words, OMFIT is exploited to execute the simulations by using other codes. It is impossible to compare the normal simulation frameworks with the modeling framework of simulation frameworks.

Considering the overall features and comparisons shown in Fig-ure 12, it can be noticed that TRIASSIC is somewhat very similar to the IPS. They both are written in a high-level language, Python, adopt full modularization, and can conduct time-dependent simu-lations. The main differences between those two are the usage of IDS and the coupling of models. While the TRIASSIC natively uses IDS as an endpoint that the Python actors directly communicate, the common data storage for the IPS is the *Plasma State* [34]. The *Plasma State* is also a generic data structure and has undergone a long period of maintenance, however, the structure is mono-blocked so that it is not hierarchical and does not have the capability of having multiple occurrences for the same data node [12]. It is noteworthy that although the actors in IPS do not directly communicate with IDS, IPS partially supports IDS as the *Plasma State* can be converted to IDS through the OMFIT.

For the coupling of models, while both inevitably utilize dynamic coupling of the codes due to the use of high-level language for the simulation, TRIASSIC uses tight coupling whereas the IPS basically uses file-based coupling. The importance of tight coupling for performance was already stressed in section 2.1.1 and a detailed explanation of this coupling will be addressed in the following section.

It can also be said that TRIASSIC is similar to the EU-IM (European framework for Integrated Modeling) workflows such as ETS and H&CD workflow when the type and purpose of the simulation codes are neglected. The main difference is the simulation engine. While TRIASSIC and H&CD workflow is being executed through the Python interpreter, ETS runs through the Kepler engine. Recently, there is a move to switch the Kepler-based workflows into Python workflows in ITER-IMAS, aiming for easy programming and high portability to extensively adapt to many analysis codes worldwide. The H&CD workflow is a result of this move when compared to the ETS workflow. The design of TRIASSIC can indeed be an appropriate goal that a fully modular simulation framework should take.



	TRIASSIC	IPS	ASTRA, JETTO, TRANSP, etc.	ETS	H&CD Workflow	STEP
Type	Framework			Workflow		
Purpose	Versatile		Versatile, but transport-related	Plasma transport	Heating & current drive	Steady-state transport solution
Language/Engine	Python	Python	Fortran	Kepler Engine	Python	OMFIT (Python)
Interface data structure	Yes	Partially	No	Yes	Yes	Partially
Full modularization	Yes	Yes	No	Yes	Yes	Yes
Coupling of models	Tight, dynamic	File-based, dynamic	Tight, static	Tight, dynamic	Tight, dynamic	Tight, dynamic
Steady-state	No	No	No	No	No	Yes

Similar to TRIASSIC … Different from TRIASSIC

Figure 12. The comparison of TRIASSIC and other integrated simulation frameworks and workflows. The green color indicates that the feature is similar to TRIASSIC, while the orange color indicates that the feature is contrary to TRIASSIC.

2.2. Components in the Framework

2.2.1. Physics Codes Interfaced with the Framework

The codes that are interfaced are shown in Figure 10. TRIASSIC incorporates the CHEASE [35] code for solving the Grad-Shafranov equation inside the plasma boundary when the boundary was given (fixed-boundary equilibrium). CHEASE code provides the calculation of various magnetic equilibrium metrics such as the profiles of flux surface averaged quantities (quantities that are once distributed on a 2D plane are averaged on a flux surface to become a 1D profile) that are useful in solving so-called 1.5D transport equations [36]. Meanwhile, the FREEGS [37] code was adopted to solve free-boundary equilibrium by considering the external coil currents such as poloidal field (PF) currents or inversely to estimate PF currents and resulting magnetic field structures inside or outside the plasma boundary based on the prescribed plasma boundary.

For the transport solver, the ASTRA [15] code was adopted for solving 1.5D flux surface averaged transport equations and the C2 [38] code was adopted for solving the entire 2D transport equations from the plasma core to the plasma-facing components. It should be noted that although the ASTRA code itself is an integrated suite of codes as already discussed in section 2.1.3 and is widely being used and upgraded as an integrated modeling framework [39], only the transport solver part of the ASTRA was adopted to satisfy the full modularization condition discussed in section 1.1.3. Hence, in this research, the ASTRA usually refers only to the transport solver part of the entire code.

For the anomalous transport models, theory-based models such as TGLF [8], GLF23 [6], and MMM7 [40, 41] were adopted. To overcome the so-called transport shortfall problem [42] under high q_{95} condition in gyro-fluid models such as TGLF and GLF23, an additional transport model called $non-linear\ enhancement$ was also adopted referring to the previous researches [43, 44]. TRIASSIC also incorporates the experimental scaling-based Bohm-gyroBohm [45, 46] model and the neural-network accelerated models such as TGLF-NN [29] or QuaLiKiz-NN [47, 48]. For the neoclassical transport models related to collisional transport with spontaneous flows and bootstrap currents, NCLASS [49] and NEO [50] codes were adopted. For a more sophisticated bootstrap current modeling, various bootstrap current models [51-53] were also coupled in the framework.

NUBEAM [54] and TORAY [55] codes were employed for the neutral beam injection (NBI) and the high-frequency wave H&CD

calculations, respectively. Note that the NUBEAM is also capable of calculating the collisional radial transport and slowing down of the fusion product and its resulting heating exerted on the thermal plasmas. Simple formulas for calculating the fusion alpha-particle heating [56, 57], the radiative cooling of hydrogen neutrals, and the fractional abundance and the radiative cooling of the impurities [58] are also available. Moreover, the atomic reaction cross-sections from $adf11$ data in ADAS (Atomic Data and Analysis Structure) [59, 60] are also utilized in the framework. The ADAS database can be used to precisely calculate the fractional abundance of the impurities and the radiative cooling of the impurities, under the steady-state assumption of the collisional-radiative model. For the remaining radiative losses, the Bremsstrahlung radiation loss is calculated by the fitting formula from the NRL formulary [61] and the synchrotron radiation loss with consideration on radiation transport is calculated by the formula [62] and additionally referring to its adaptation on the CRONOS suite of codes [17].

The steady-state neutral distribution in the core plasmas can be calculated by two codes, FRANTIC [63] and GTNEUT [64, 65]. The codes calculate the neutral population of the main ion species by taking the charge exchange and the ionization reactions of neutrals into account. While the FRANTIC code is 1D as it calculates on the cylindrical coordinate considering the effective volume of the flux surfaces, the GTNEUT code is 2D and runs on an externally provided 2D mesh. GTNEUT code can be utilized to calculate both inside and outside of the plasma boundary while the FRANTIC code only considers inside the plasma boundary as it is 1D. Given the neutral density inside or outside the plasma, reaction cross sections, and plasma densities, particle source rates and ionization cooling rate can be derived.

The calculation of neutral distribution and 2D plasma transport in the SOL region or even inside the plasma boundary requires a 2D mesh. Mesh generators are separated from the neutral code or the 2D transport code and interfaced as a separate model in the framework. TRIASSIC incorporates the VEGA [66] mesh model to generate a high-quality, field-aligned quadrilateral mesh and TRIANGLE [67] code to generate Delaunay triangular mesh, both inside or outside the plasma boundary separatrix. Modeling of PFCs is important for investigating PWI. The triangular mesh generator is required to fully consider the complex structure of PFCs, while properly capturing the fast transport process along the magnetic field line through field-aligned mesh.

For the study of the pedestal and to consider enhanced performances of H-mode plasmas, TRIASSIC contains a neural network accelerated model, EPED1NN [23, 29], which predicts the pedestal width and height. The formation and collapse of the pedestal require an additional model that manages the plasma transport on the pedestal region; the transport models to achieve the desired pedestal width and height in a brute (plasma densities or temperatures are directly adjusted by the model instantaneously or gradually) method or by plausible modeling that mimics ELMs or so-called continuous ELMs [68] by adjusting the particle or heat diffusivities. The occurrence of the H-mode can be judged by the loss power scaling laws from the reference [69].

Furthermore, TRIASSIC incorporates a linear toroidal Alfvén eigenmode (TAE) model [70], which finds the linear stability of TAEs. It also contains simple formulas of Ohmic (Joule) heating and collisional heat exchange between electron and ion based on the NRL formulary.

2.2.2. Physics Code Interfacings

Most of the physics codes are written in low−level languages such as Fortran, C, and C++ for performance. Note that, unlike common sense in the programming language, in this research, the word $low - level$ indicates a low level of user−friendliness and machine dependency in compiler language so the low−level lan−guage includes Fortran, C, and C++. Here, the high−level language only refers to Python. Direct interfacing of low−level physics codes and their usage in the high−level language required an ad−ditional wrapper.

Figure 13 shows the interfacing procedure of the tier 2 physics code. The tiers in code integration and their difference were ad−dressed in 1.1.6 and Figure 9. When a legacy physics code is given, at first the driver routine is added to the code to make the code a tier 3 physics code. After that, the IDS is imported into the I/O routine to make code use the IDS inherent in the I/O. Finally, as the output from the physics routine (represented as a black circle) does not satisfy the consistency between the data nodes, the I/O routine should update other nodes (represented as red, blue, green, and purple circles) in the IDS to make the data consistent. The original physics code should be significantly modified. Moreover, due to these modifications, interfacing with a tier 2 physics code requires a significant effort. Especially, a developer should have an understanding of both physics code and the structure of the IDS.

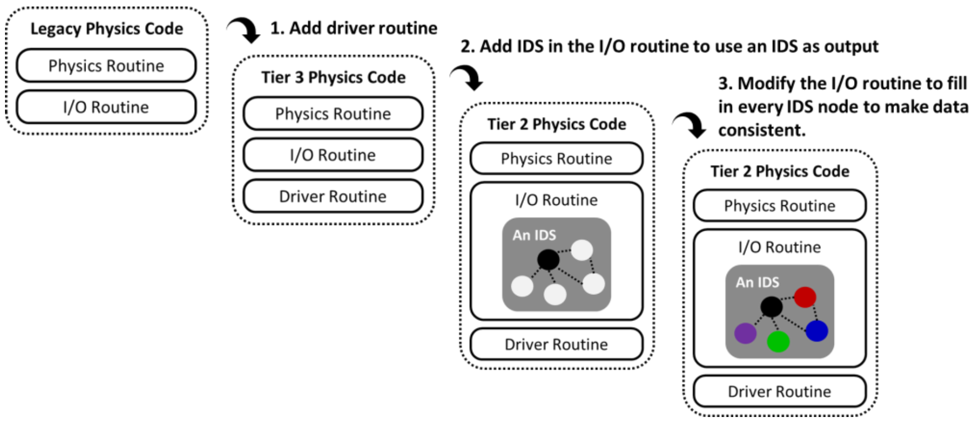


Figure 13. The development procedure of legacy physics code to−ward the tier 2 physics code.

To overcome the hardship of code interfacing, TRIASSIC takes the tier 3 approach to interface the legacy physics codes with the framework. Figure 14 shows a detailed view of low−level legacy physics code coupling in TRIASSIC. When an original code is given, only a driver routine is made to utilize the code in the framework in a proper way. The F2PY [71] or the SWIG [72] was used to create a wrapper for the low−level language code, to utilize the routine written in a low−level language in the Python framework. Then, the original code, driver routine, and wrapper were compiled and linked together to form a shared (dynamic) library that can directly be imported into the framework.

During the simulation runtime, as was briefly mentioned in sec−tion 2.1.2 and shown in Figure 11, when the simulator invokes the Python interface, the interface gets the plasma/tokamak data from the IDS. The Python interface then converts the standard data into a suitable form for code execution with a simple calculation. Re−ferring to the pre−compiled library, the driver routine was called by transferring code inputs to the subroutine or the function. The calculation result of the physics code is then returned to the Python interface, and the interface again converts the simulation result into the standard form to put data in the IDS.

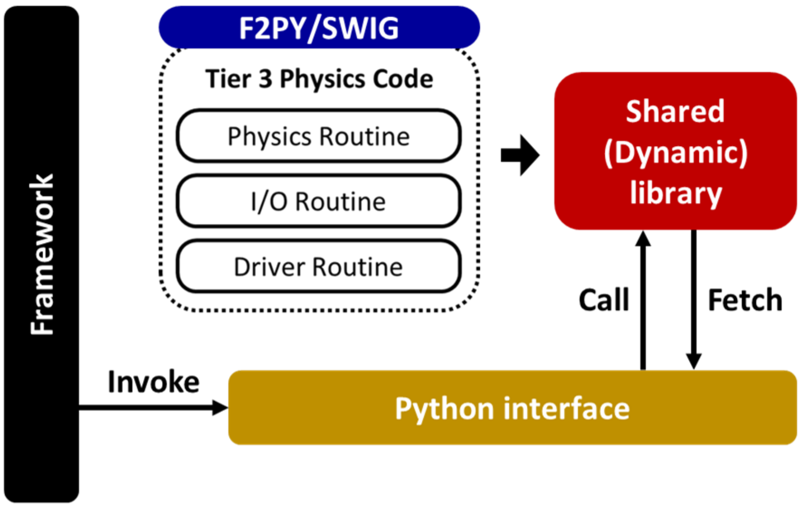


Figure 14. A detailed view of the tier 3 code interfacing in TRIAS–SIC. The original code is compiled by F2PY or SWIG with an additional driver routine to form a shared (dynamic) library. The library communicates with the IDS through the Python interface.

There are some notable features that the interfacing method has. First, there are no or only minor modifications to the original physics code, due to the tier 3 approach. Only the driver routine should be added to the original physics code or even there are cases where the driver routine is already well–defined and does not require an additional one. Second, the interfacing of physics codes is dynamic. the whole set of physics codes does not have to be completely prepared for a simulation and the libraries are imported during the simulation runtime. In other words, when there is a minor modification of a driver routine, there is no need to re–compile the entire code to make a new executable. It only requires re–compile the library that has the modifications on the original code or the driver routine. This indeed can be significant merit when the size of the code suite becomes considerable. The two merits of the interfacing method could significantly reduce the time spent on development.

Third, the interfacing of the original code is direct; the Python interface directly calls the driver routine through the wrapper. Moreover, the simulation does not need to spawn additional calculation jobs. The method is much more efficient, safer, and less error-prone than transferring the data between the programs through the network protocol or the file I/O.

Finally, the framework does not depend on external programs or machines. If a unified version of Python and its consistent dependency of low-level memory support of arrays is given, any compiler vendor supported by the dependency can be utilized to create the dynamic library. Moreover, it is possible to compose and execute a single workflow with different compiler vendors. It is also noteworthy that the framework does not depend on the external program except for the framework interpreter (Python) itself, for the execution of the simulation.

The code interfacing can be compared with the other frameworks as shown in Figure 15 and Figure 16. Figure 15 shows the file-based coupling of a physics code. In this case, the Python interface communicates with the original code by the file I/O. It can burden a high load on the machine due to high file I/O when a large amount of data is frequently exchanged. Also, as a new process is spawned by launching the executable in the Python interface, it is error-prone and unsafe in terms of managing the job processes. On the other hand, according to the tier of code integration introduced in 1.1.6, the case shares the tier 3 approach with TRIASSIC. Due to the approach, there are advantages in that there are minor or no original code modifications, due to the absence of the additional I/O routine. It enables the use of legacy codes that are not supported by the original developer, and ultimately efficient development and interface of various analysis codes.

Figure 16 shows the coupling of the physics code through the external program, Kepler. In this case, the workflow is launched through the external program; it can be a disadvantage as it requires an additional program and an adaptation on it other than the well-known scientific analysis tool such as Python. The case corresponds to the tier 2 approach. The original physics code gets significantly modified by adopting the IDS in the I/O routine or even in the physics routine and is compiled with the FC2K to generate the Kepler actor. It is noteworthy that the communication of the IDS data is somewhat different between the approach and TRIASSIC. While the Python interface in TRIASSIC merely gets and puts some data nodes from and to the IDS, the Kepler actor gets and returns a whole set of IDSs. Precisely, TRIASSIC does the modification via an in-place method while the Kepler approach does not. The in-place modification is more efficient as it does not have to create a new IDS object when implemented. However, the consistency in the IDSs can be broken and further modifications are required to keep the consistency. The details of the problems in internal data consistency followed by the interfacing method and the role of the TRIASSIC framework will be further addressed in the next chapter.

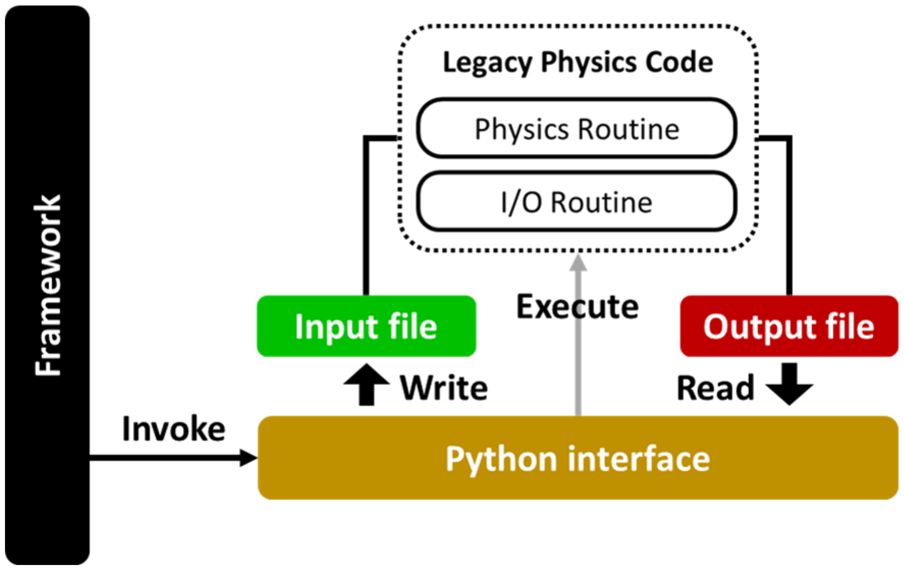


Figure 15. The tier 3, file−based code interfacing of the Python framework.

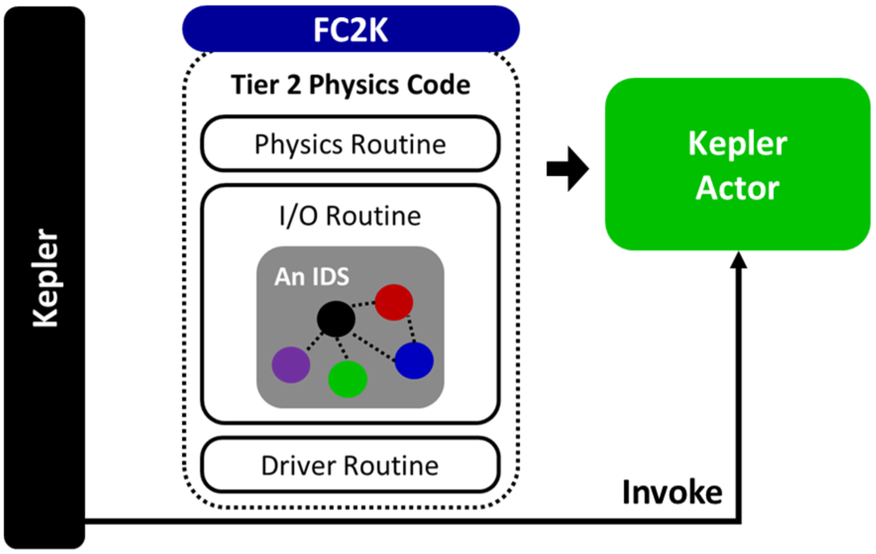


Figure 16. The tier 2 code interfacing which depends on the exter− nal program, Kepler.

2.2.3. Graphical User Interface

As was addressed in section 2.1.2, three input files are required for the simulation. The GUI was developed to help generate these input files, as it is hard to produce entire files from the scratch due to a certain data structure that the three input files have. The GUI should guide generating input files via a user–friendly environment and could be used to validate the schema of the files. Especially, for the *data* file, where most of the contents therein are unneces–sary when an input IDS is given, the GUI should ease making the input file by letting the user know which shape and type a toka–mak/plasma data has.

The most frequently used inputs in a variety of plasma simula–tions are the plasma current, toroidal magnetic field, plasma density, and temperature profiles. Naturally, those data are distributed in time; hence, the data are usually 1D in time or 2D in time and space. On the other hand, it is frequent that a modeler has the plasma data by comma–separated values (CSV) or a spreadsheet format. The data in such formats could be easily imported to the GUI through interactive interfaces, such as copy–paste. Besides, well–known file formats in the field of fusion plasmas such as G–EQDSK should be able to be imported through the GUI. A user should be able to choose which data to be fetched from such well–known file formats.

Apart from the schema of the data, it is also important to check the sanity of the multi–dimensional data. Most of the sanity checks will be conducted by the simulation user. The checking, however, is inefficient when looking at the data shown in the text format. Therefore, the GUI should provide the functionality to visualize the spatiotemporal data in an appropriate form.

It is noteworthy that, despite the user–friendliness of the GUI, a text−type environment is often useful. Specifically, the text−type environment is useful when running a set of simulations by varying the value of a parameter (e.g., the parameter scan). Also, storing the simulation data by converting the input data into the input files is necessary for recording the simulation history. Therefore, the simulation was enforced to be executed only by the input files, not by the GUI as shown in Figure 17. Nonetheless, the user−friendly environment and the text−type environment should be easily convertible in both directions. An example of a TRIASSIC GUI is shown in Figure 18.

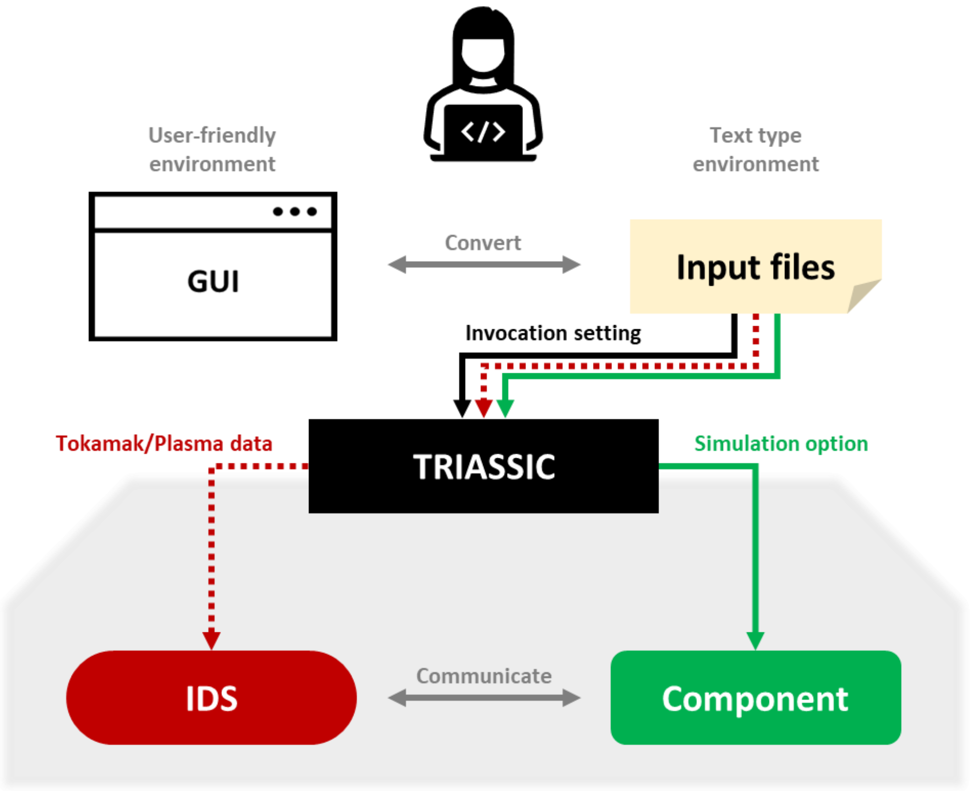
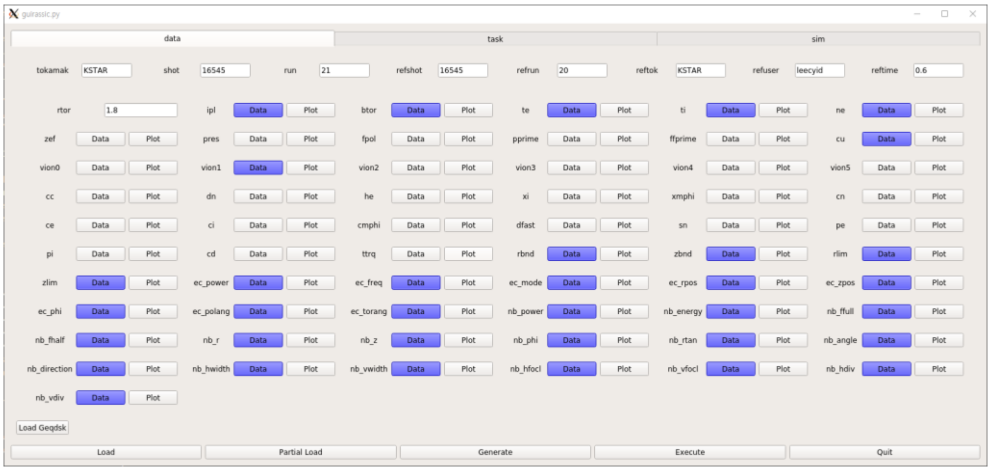


Figure 17. Schematic view of three input files (*data*, *task*, and *sim* shown in Figure 11 are shown as red, green, and black lines, re−spectively) and the role of the graphical user interface in TRIAS−SIC.


tokamak KSTAR shot 16545 run 21 refshot 16545 refrun 20 reftok KSTAR refuser leecyid reftime 0.6

rtor 1.8

guirassic.py
data
task
sim
ipl btor te ti ne
zef pres fpol pprime ffprime cu
vion0 vion1 vion2 vion3 vion4 vion5
cc dn he xi xmphi cn
ce ci cmphi dfast sn pe
pi cd ttrq rbnd zbnd rlim
zlim ec_power ec_freq ec_mode ec_rpos ec_zpos
ec_phi ec_polang ec_torang nb_power nb_energy nb_ffull
nb_fhalf nb_r nb_z nb_phi nb_rtan nb_angle
nb_direction nb_hwidth nb_vwidth nb_hfocl nb_vfocl nb_hdiv
nb_vdiv
Data
Plot
Load Geqdsk
Load
Partial Load
Generate
Execute
Quit

Figure 18. The ***data*** tab of TRIASSIC GUI. A user can manipulate simulation input data for the three parts – ***data***, ***task***, and ***sim*** – as represented in the GUI.

2.2.4. Jobs Scheduler and MPI

The simulation tasks are executed by the job scheduler such as Sun Grid Engine or SLURM through a shell script. Based on the load balancing through the job scheduler, the simulation of physics components in TRIASSIC was accelerated by adopting the message passing interface (MPI). Though being an integrated simulation framework where there are frequent that some code components can only be run on a single processor while the other component should be run in parallel for a feasible simulation, TRIASSIC does not support spawning new processors. Job processors are re-served at the initial phase of the simulation and all of them proceed together until the end of the simulation. There are two types of MPI implementation in TRIASSIC.

When the original code natively supports the MPI, basically all the processors join the main calculation routine through the Python interface. The original code then properly distributes the given processors. In this case, the MPI scalability depends on the scala–bility of the legacy code. The NUBEAM (Monte–Carlo neutral par–ticle simulation) and C2 (2D transport with multiple separate zones) correspond to this case.

On the other hand, there is a case when the code requires many independent calculations. The local transport models such as TGLF, GLF23, NCLASS, and NEO correspond to this case. The plasma transport is evaluated independently on a local flux surface position. During the simulation, the Python interface distributes the calcu–lation on a worker pool by different flux surface positions. Figure 19 shows the MPI scalability of the TGLF code, calculated on 100 different radial grids. With the increase of the number of proces–sors, the speedup was increased quite well. The speedup efficiency was about 80% when the number of processors was equal to half of the number of radial grids.

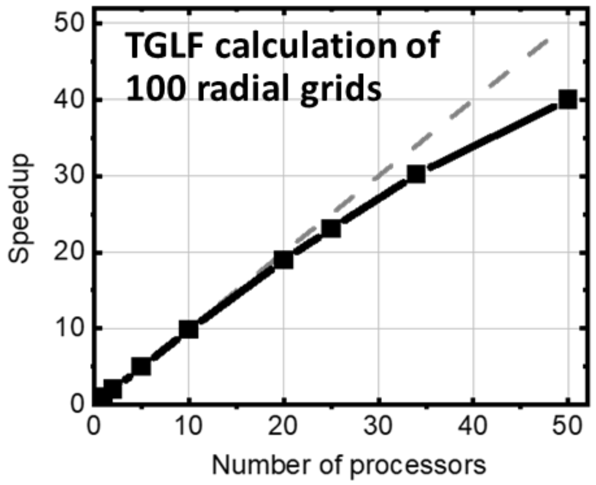


Figure 19. The MPI scalability of the TGLF calculation of 100 radial grids.

2.3. Verifications

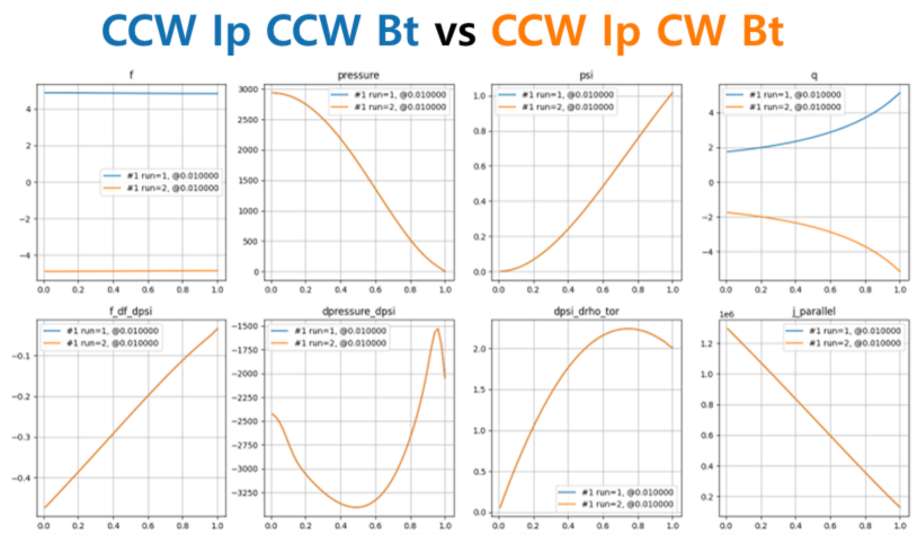
2.3.1. The Coordinate Conventions

The coordinate conventions (COCOS) [73] define a set of numbers to define the coordinates of various tokamaks and modeling codes. The COCOS number 11 was chosen in the ITER, where 11 denotes the counter-clockwise (CCW) direction is positive (from the top-view) both for the plasma current and the toroidal magnetic field while using the Wb units of the poloidal magnetic flux. Sticking to a consistent coordinate notation is essential, as in integrated modeling, various codes are coupled together and it is highly possible to make a mistake when considering actuators whose toroidal injection angle is important such as NBI or EC or when modeling the plasma rotation in the simulation. Therefore, the test simulations were conducted by flipping the signs of the plasma current and toroidal magnetic field inputs to check whether the COCOS 11 is fully kept in TRIASSIC.

Figure 20 shows the results of the test simulations. On the upper panels, the standard case - CCW plasma current with CCW toroidal magnetic field - is shown. It shows the positive toroidal field function f, safety factor q, and parallel current density. Due to the radially increasing poloidal magnetic flux ψ, the pressure gradient is negative. The signs of the equilibrium variables under positive plasma current and magnetic field are correct based on the COCOS paper. The comparison of the standard case with the CCW plasma current with the CW toroidal magnetic field case shows that only the toroidal magnetic field function f and the safety factor q differs. The results coincide with the convention, as only the f and q

depends on the sign of the toroidal magnetic field.

On the other hand, the comparison of the standard case with the CW plasma current and CCW toroidal magnetic field case shows that every equilibrium variable except for f and pressure flips sign, due to the radially decreasing poloidal magnetic flux. The safety factor also flips the sign as the poloidal direction of the magnetic pitch angle is reversed. Although it is not shown, the coordinate convention and the sanity of the equilibrium variables were verified not only when the plasma current and the toroidal magnetic field are inputs, but also when the p' (pressure gradient) and the ff' are inputs for the equilibrium calculation.



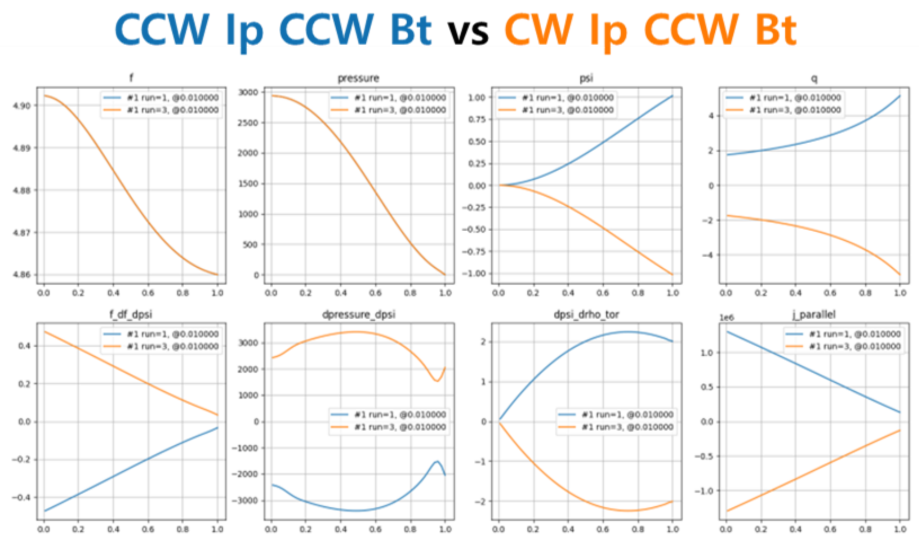


Figure 20. The comparisons of plasma equilibrium profiles (toroidal field function f, equilibrium pressure, poloidal flux psi, safety fac−tor, ff', p', $\partial\psi/\partial\rho$, and parallel current from upper−left to lower−right). The upper figure compares the standard CCW I_P, CCW B_T case with CCW I_P, CW B_T case, while the lower figure compares the standard case with CW I_P, CCW B_T case.

2.3.2. Coupling of Equilibrium−Transport

It is essential to check whether the coupling of the codes in the integrated suite is valid or not. Especially, the coupling of equilib-rium and transport is one of the most important code couplings in the integrated simulation. Here, the TRIASSIC current diffusion simulation with ASTRA and CHEASE components was compared with the standalone ASTRA-SPIDER [74] simulation. Although it is not shown in the current section, it was also checked that the particle and heat transport work well in TRIASSIC. However, here the focus is on solving the current diffusion equation (CDE) be-cause the form of the equation is much more complicated than that of the particle or heat transport equation. Moreover, the additional terms related to the volume expansion, or the magnetic field vari-ation can be significant in CDE rather than in the other transport equations.

Figure 21 shows the test simulation of current diffusion from both ASTRA-SPIDER and TRIASSIC. It was checked whether the toroidal electric field, which initially increases radially towards the edge, eventually gets flattened or not due to the current diffusion. The result shows that the toroidal electric field gets flattened in a current diffusion time-scale until the difference between the mag-netic axis and the edge gets numerically negligible. The simulation proves that the transport equation and the coupling are correctly implemented in the TRIASSIC suite of codes.

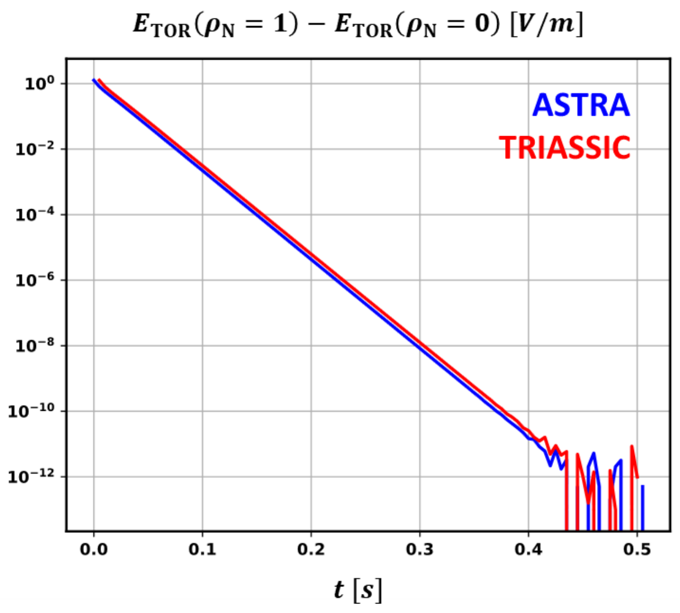


Figure 21. The flattening of the toroidal electric field of current diffusion simulations. ASTRA and TRIASSIC both prove that the current diffusion simulation is valid as they show that the electric field can be flattened until it becomes sufficiently small.

In the following test simulation, the plasma current, toroidal magnetic field, and minor radius were simultaneously varied in time to verify the suite under such a highly complicated situation. The initial values of the KSTAR range are $600\,kA$, $2.7\,T$, and $0.5\,m$, re- spectively, and all of them were varied by a factor of 1.5, as shown in Figure 22 (a), (b), and (c). The initial parallel current density profile was set to a constant value, and the parallel electric con- ductivity was set to a parabolic shape with its center value of $1\times10^{6}\,Ohm^{-1}m^{-1}$. Non-inductive current drives such as bootstrap current or externally driven currents were neglected. The time step of both simulations was intentionally set to a considerable value of $2\,ms$ to exaggerate the discrepancy between them, which would appear if the coupling were not adequately done.

The comparison of the resulting current density, safety factor, and toroidal electric field from both simulations are shown in Figure 22 (d), (e), and (f), at three normalized square roots of the toroidal flux coordinate positions, $\rho = 0.0, 0.5, 0.95$. The current density and safety factor from TRIASSIC perfectly follows its counterpart calculated from the standalone ASTRA. However, though it is not shown, the comparison of the safety factor at the near boundary ($\rho > 0.95$) showed a slight discrepancy, with its value from TRIASSIC a bit smaller. This discrepancy originated from the inherent difference between the two equilibrium codes while dealing with the magnetic equilibrium at the near-boundary region and will not be further discussed in this verification. The toroidal electric field, which shows a very transient time evolution and many peaks due to sudden rise/drop of the plasma current or the plasma size, also showed a good agreement between the two codes.

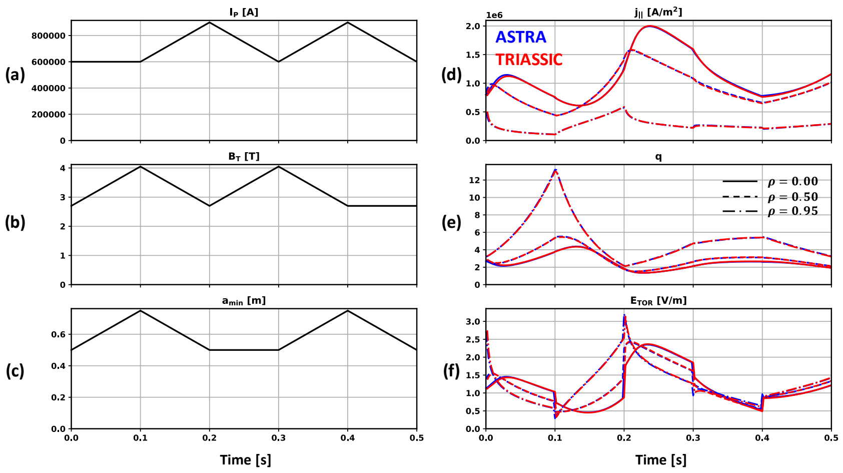


Figure 22. The comparison of current diffusion simulation between the ASTRA transport code and TRIASSIC. The graphs on the left show the prescribed evolution of the plasma current (a), toroidal magnetic field (b), and plasma minor radius (c). The graphs on the right show the resulting time evolution of the current density (d), safety factor (e), and toroidal electric field (f) at three positions from both ASTRA (marked as blue color) and TRIASSIC (marked

as red color). Each solid, dashed, and dash-dotted line represents $\rho = 0.0, 0.5, 0.95$, respectively, where ρ is the normalized square root of the toroidal flux.

2.3.3. Neoclassical Transport and Bootstrap Current

Verifying the coupling of neoclassical transport is crucial for accurate predictive simulation and interpretation. Moreover, the bootstrap current which results from the neoclassical theory is a key parameter for plasma performance. Here, the NCLASS code in TRIASSIC was verified against the NCLASS implementation from the ASTRA framework.

The comparison of TRIASSIC-NCLASS and ASTRA-NCLASS in terms of the neoclassical transport coefficients, conductivity, and the bootstrap current is shown in Figure 23. The diffusion coefficients, convection velocities, electric conductivity, and bootstrap currents showed a good agreement between the two code implementations.

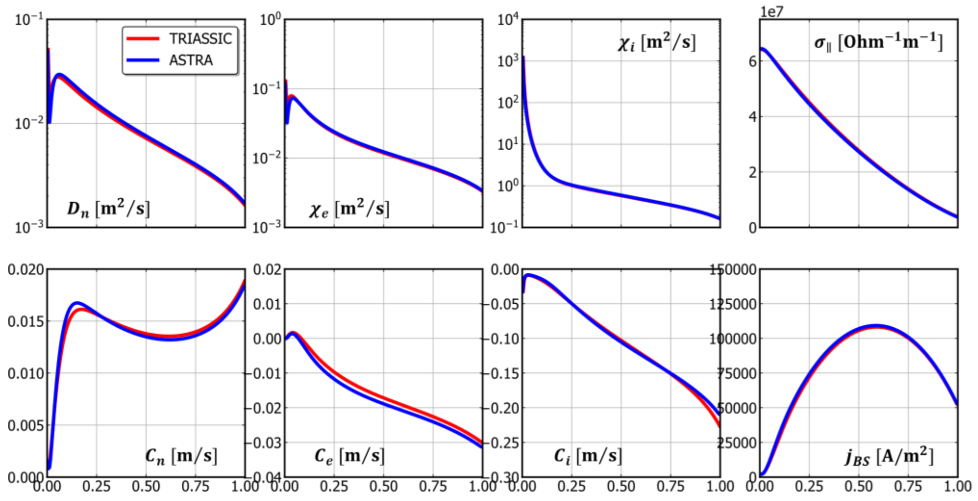


Figure 23. The comparisons (red for TRIASSIC and blue for AS-TRA) of neoclassical transport coefficients - particle, electron energy, and ion energy diffusion coefficients (upper) and convection velocities (lower). The comparisons of the electric conductivity (upper right) and the bootstrap current (lower right) are also shown.

Figure 24 shows the comparison of the bootstrap currents calculated by four different models. The bootstrap currents were calculated under the KSTAR-like simulation condition similar to the condition used in the previous section. Though it is not shown, the bootstrap currents calculated by other models showed negligible differences under the standard KSTAR-like case.

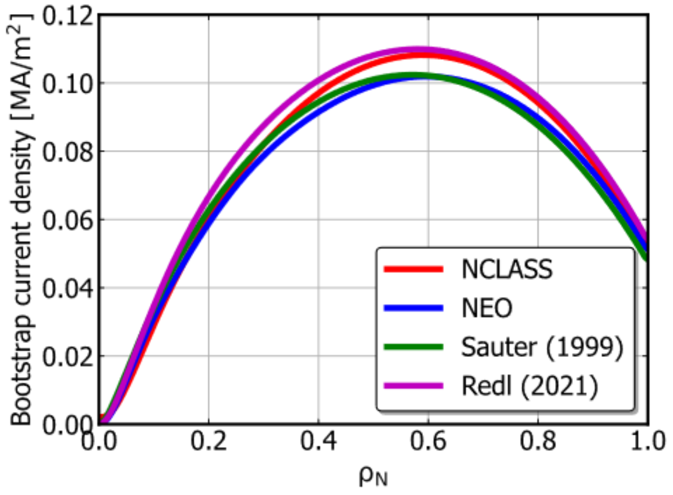


Figure 24. Comparison of four different bootstrap current models interfaced in TRIASSIC.

2.3.4. Heating and Current Drive

The verification of the NUBEAM component in TRIASSIC was conducted by comparing the H&CD calculation results with different NUBEAM implementations. The target discharge was KSTAR discharge #18602, where a total of $3.8\,MW$ of beam power with an average energy of $80\,keV$ from three beam sources were injected. In Figure 25, the power deposition, and the current drive from three different NUBEAM implementations are compared. The NUBEAM standalone stands for the standalone implementation using the KSTAR EFIT [75] equilibrium with kinetic profiles, and TRANSP-NUBEAM stands for the TRANSP implementation of the NUBEAM module. The H&CD profiles shown in Figure 25 (a) and (b) showed a good agreement between three different implementations, while there was a slight discrepancy inside $\rho = 0.2$ due to statistical noise. The integrated H&CD profiles shown in Figure 25 (c) and (d) showed $\cong 2.9\,MW$ of the total deposited heating power and $\cong$

0.14 MA of the total current drive for three cases, though the mag− netic equilibrium in TRIASSIC was calculated by the CHEASE component while the EFIT magnetic equilibrium was used for the others. The discrepancies between the total deposited power and the total driven current between TRIASSIC and two other cases were within 4%.

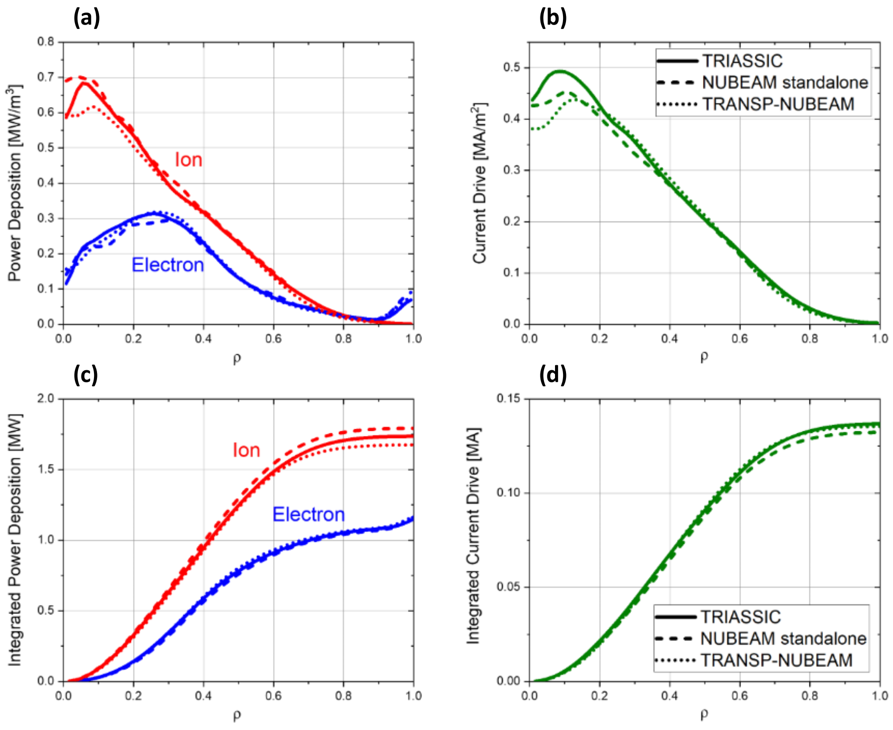


Figure 25. The comparison of neutral beam power deposition profile (a), the beam current drive profile (b), integrated power deposition (c), and integrated current drive (d) from TRIASSIC (solid line), NUBEAM standalone (dashed line), and TRANSP−NUBEAM (dot− ted line).

The coupling of the TORAY component in TRIASSIC was also compared with the standalone TORAY implementation in KSTAR. The calculation was done for KSTAR discharge #26381, where

$550\,kW$ of EC heating was injected into a normal direction. The toroidal injection angle was scanned to verify a wide range of calculations from counter–injection to co–injection relative to the plasma current and toroidal magnetic field direction, though a too–large toroidal steering angle is not viable in KSTAR. In Figure 26, the toroidal injection angle scan and the driven current are shown. The driven current calculated by the TORAY coupled with TRIASSIC showed good agreement with that of the standalone implementation.

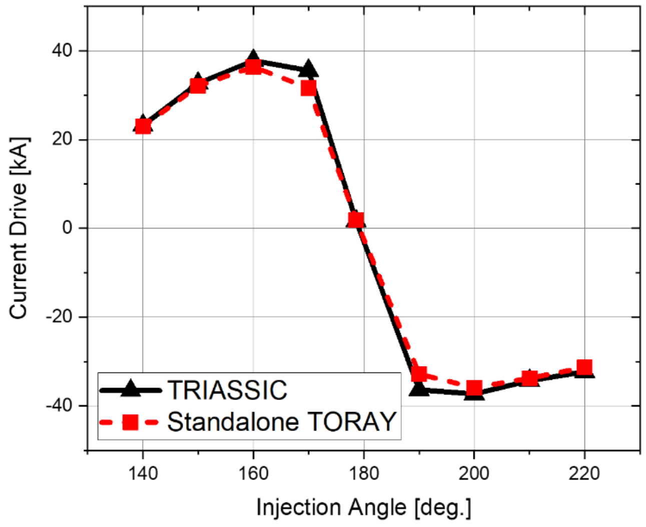


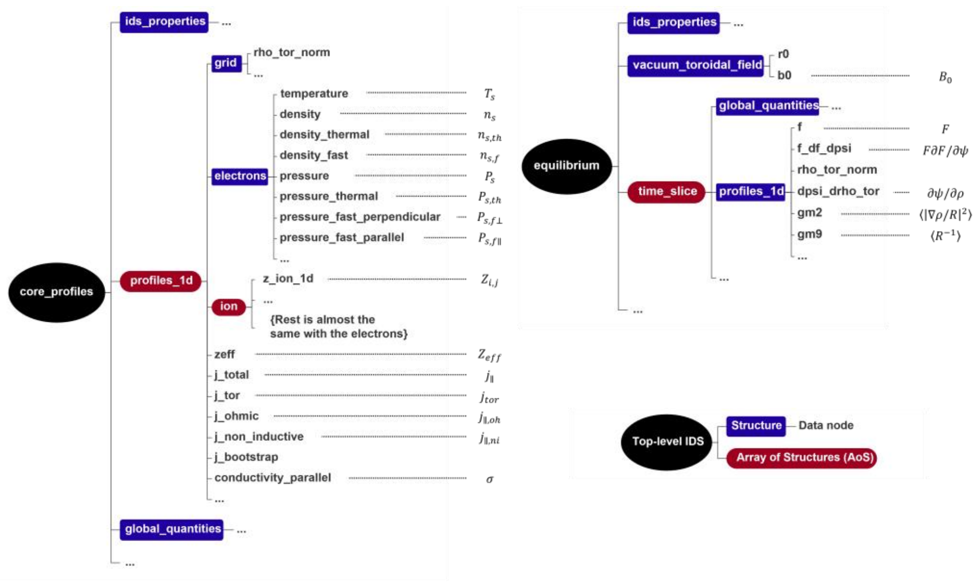
Figure 26. The comparison of the amount of EC waves current drive between TRIASSIC (black solid line) and standalone TORAY (red dashed line) with different injection angles.

Chapter 3. Improvements in Keeping the Internal Data Consistency

3.1. Background

The essence of the generic data structure of IDS is the physics data model. The data model defines the definitions of the data nodes and the expressions to link the data nodes. These definitions and expressions are provided as guidelines; hence it is hard to guarantee that they are satisfied. In addition, there exist some constraints such as quasi-neutrality that modelers would like to constrain. Under the circumstances, when a code component modifies only a part of IDS, which is the case in TRIASSIC, inconsistency of definitions/constraints in the internal data is expected. The inconsistency in data could lead to an unexpected simulation result. Thus, keeping the data consistency is critical.

As TRIASSIC takes the tier 3 approach to interface a physics code, the code interfaces were designed to update only the unique output data calculated by the code. Frequently, it is natural for a component to update only a few data nodes rather than updating a whole data node in an IDS, which latter is the case for the tiers 1 and 2 approaches. For instance, when the bootstrap current model is used, the model is expected to calculate the bootstrap current density profile by taking the plasma density and temperature profiles. As there are no useful outputs other than the bootstrap current density data that the model can calculate, it is natural that the code only updates the $j\_bootstrap$ node in the $core\_profiles$ IDS shown in Figure 27. Another example can be the situation when a thermal electron density at $t+\Delta t$ is predicted by the transport solver component, by using the thermal electron density data at a current time step, t. The useful outputs from the transport solver component will be the thermal electron density at $t + \Delta t$ and the transport-related values such as the electron flux and effective particle diffusivity. In this case, there is no need for a component to modify data other than the *density_thermal* from the electrons in *core_profiles* IDS.



core_profiles
ids_properties
...
grid
rho_tor_norm
...
electrons
temperature T_s
density n_s
density_thermal $n_{s,th}$
density_fast $n_{s,f}$
pressure P_s
pressure_thermal $P_{s,th}$
pressure_fast_perpendicular $P_{s,f\perp}$
pressure_fast_parallel $P_{s,f\parallel}$
...
profiles_1d
ion
z_ion_1d $Z_{i,j}$
...
{Rest is almost the same with the electrons}
zeff Z_{eff}
j_total $j_\parallel$
j_tor j_{tor}
j_ohmic $j_{\parallel,oh}$
j_non_inductive $j_{\parallel,ni}$
j_bootstrap
conductivity_parallel σ
...
global_quantities
...
equilibrium
ids_properties
...
vacuum_toroidal_field
r0
b0 B_0
time_slice
global_quantities
...
profiles_1d
f F
f_df_dpsi $F\partial F/\partial\psi$
rho_tor_norm
dpsi_drho_tor $\partial\psi/\partial\rho$
gm2 $\langle|\nabla\rho/R|^2\rangle$
gm9 $\langle R^{-1}\rangle$
...
Top-level IDS
Structure
Data node
Array of Structures (AoS)

Figure 27. A part of the data nodes in the *core_profiles* and *equilibrium* IDS. The *core_profiles* IDS contains the one-dimensional information of the core plasmas, and the *equilibrium* IDS has the one-dimensional flux averaged metrics from two-dimensional magnetic equilibrium as a part. The top-level IDS, a structure, and an AoS are shown as a black circle, a blue square, and a rounded red square, respectively, and the data node is shown as black plain text. The *ion* structure has almost the same as that of the *electrons*, except for the information of atomic number and charge.

Although there are cases where updating an entire data node in IDS is more appropriate, frequently, a component does not need to update all the items in an IDS. However, internal data consistency is expected to be lost in this case. For the second example of the above paragraph, when the value of *density_thermal* is updated by the transport solver, the relation between *density_thermal* , *density_fast*, and *density* in electrons can be no longer satisfied. One of *density* or *density_fast* needs a further update to match the definitions for the internal data consistency. Additionally, when the former is updated, by following the quasi–neutrality constraint of the plasma, an appropriate modification of ion densities should be made while *zeff* is assumed to be constant. The change of ion densities should eventually lead to a modification of ion pressures. The overall procedure is depicted in Figure 28. It casts a need for subsequent updates when a code component updates a data node.

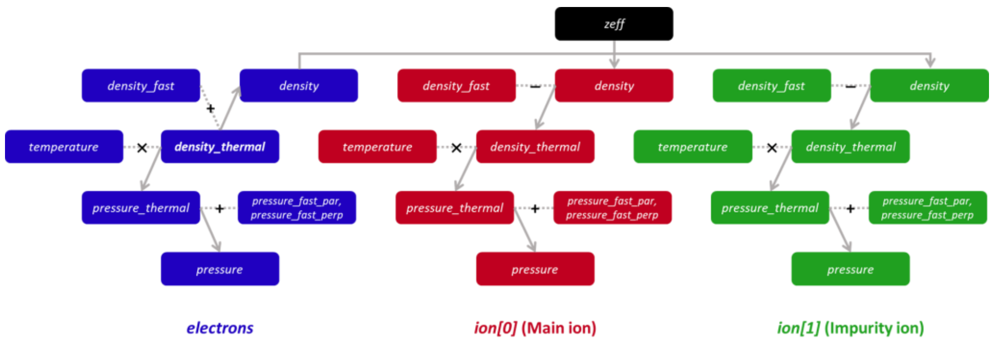


Figure 28. An example of the update procedure of the *core_profiles* IDS when the *density_thermal* node in the *electrons* structure was updated. The plasma was assumed to be composed of electrons and two ion species, where the ions have a constant charge. The elec– trons, main ion, and impurity ions are marked as blue, red, and green colors, respectively. The dotted line indicates that two linked nodes have a simple mathematical relationship to calculate the other node connected by an arrow.

It is noteworthy that this consistency problem can occur when a generic data structure is composed of data nodes that can be de– scribed by other nodes. Namely, when the data nodes are mutually related and can form a reciprocal relation by the definitions or the constraints. For example, *density* can be expressed as the sum of *density_thermal* and *density_fast*. Nonetheless, three of them are all useful quantities that deserve allotting a node. Moreover, the problem occurs when the generic data structure communicates with the codes that are originally incompatible with the data structure. In other words, the problem occurs in the tier 3 approach to code interfacing, where the data structure in a framework is more generic than the data covered by the physics code.

3.2. Possible Implementations of a Component

Three possible implementations of a component in a framework are possible from the perspectives of internal data consistency. First, the component manages to keep the internal data consistent. In this case, the data consistency can be kept when a component manages the consistency of the given IDS. This approach does not rely on the framework for keeping the data consistent. It is essentially the same as the usage proposed by the original paper [12, 19] of the generic data structure and the tiers 1 and 2 approaches. In this case, there is no problem related to internal data consistency in a framework, as the duty of consistency is burdened on a code component. However, as the method expects a component to do every procedure to keep the data consistency, the functionality of a physics code becomes too broad and ambiguous. Also, the process that keeps the data consistent would be the same for different codes with the same kind of output. In other words, the duplication of code is expected.

The second corresponds to the case where the framework provides special component(s) which explicitly conduct additional calculations to keep the internal data consistency. For this second case, the framework provides special component(s) which explicitly conduct additional calculations to keep the internal data consistency. In this case, distinct from the previous approach, the subject in charge of data consistency is not a component but the framework. To make it work, several explicit components that can be applied after a physics component execution should exist within the framework. For instance, a special component handles all the required calculations for data consistency when the electron density is renewed. Again, the code duplication problem between the special components would emerge unless the component is divided into a set of indivisible components that do the same calculation as above when in combination. If so, however, when designing a workflow, the framework demands a user to manage all the cumbersome settings by placing the required post-run components. There exists a tradeoff between code duplication avoidance and user-friendliness. Nevertheless, the approach can be an appropriate alternative to the previous one, as the functionality of the code is maintained to be specific.

The EU-IM workflows such as ETS adopt both first and second approaches. While the actors in ETS keep the consistency within an IDS, the non-physics actors [76] keep the consistency between the IDSs or even between heating/current drive sources and transport models. The use of special components such as non-physics actors, however, made it difficult on composing a new workflow from the scratch. Hence, the EU-IM workflows are already well-defined for a specific analysis target and mostly the simulations are conducted by configuring input parameters but not by varying physics components and their execution by plug-and-play of the models. The details will be further discussed in this chapter.

Lastly, the framework can conduct automated implicit subsequent updates after the component execution. The approach is al-most the same as the previous one, but the main difference comes from the transparency of the procedure that brings the data con-sistency. When a component updates the data, the framework will do the rest of the calculations beneath the surface, satisfying the data consistency. There is no need for a user to take care of the post-run components in designing a workflow in this approach. In contrast to the second approach, however, it is hard to notice what constraint/definition the framework imposes due to implicitness.

Further in this approach, the drawbacks could be alleviated by adopting the declarative programming concept in the framework, compared with the typical imperative programming style. Declaring the relations that should be satisfied could make it easy to check which definitions or constraints are applied in the simulation and thus how the data node can be calculated in the framework. Also, declarative programming was far more flexible for various use cases than the imperative programming style, which requires a case-by-case study on various use cases. A detailed explanation of the flexibility of this approach will be addressed in the following sections.

3.3. A Method Adopted in the Framework

3.3.1. Prerequisites and Relation Definitions

For being a flexible and user-friendly framework, the last method introduced in the previous section was adopted in the TRI-ASSIC framework. First, imperative programming was adopted to achieve consistency. However, the maintenance was difficult as adding many physics components and using the components for various purposes. To overcome the maintenance issue, TRIASSIC was upgraded to use declarative programming to achieve internal data consistency. A detailed comparison of previous and current methods will be addressed in the following section.

Table 1 shows a part of the relations (definitions and constraints) used in the framework. The input and output variables ordered by the constancy for each relation are summarized in Table 1. One can think of a situation where the electron density was modified by a physics component, while there exist two ion species - the main and the impurity ion - in the plasma. Focusing on the relation 1 and 2, which are the quasi−neutrality constraint and the definition of effective charge Z_{eff}, respectively, there can be a case where the framework should determine two ion densities while Z_{eff} is fixed. On the other hand, when the relation 3 came in to constrain the impurity density as a certain fraction of the electron density, the framework should determine both the main ion density and Z_{eff}. The use case can become even more diverse when more than two ions are considered in the simulation. Given the situation, satisfying the relations via imperative programming would require examining all the use cases and dealing with the corresponding case one by one. The method is inefficient and prone to unexpected errors when a new use case is introduced.

	Relation	Output varia−ble(s) (by order of constancy)	Input varia−ble(s)
1	$n_e = \sum_{j=1}^{k} n_{i,j} Z_{i,j}$	$n_{i,1..k}$, n_e	$Z_{i,1..k}$
2	$n_e Z_{eff} = \sum_{j=1}^{k} n_{i,j} Z_{i,j}^2$	n_e, Z_{eff}, $n_{i,1..k}$	$Z_{i,1..k}$
3	$n_q = f_{pq} n_p$	n_p, n_q	
4	$n_s = n_{s,th} + n_{s,f}$	$n_{s,f}$, n_s, $n_{s,th}$	

5	$P_{s,th} = n_{s,th} T_s$	$P_{s,th}$	$n_{s,th}$, T_s
6	$P_s = P_{s,th} + f_{\parallel} P_{s,f\parallel} + (1 - f_{\parallel}) P_{s,f\perp}$	$P_{s,th}$, P_s	$P_{s,f\parallel}$, $P_{s,f\perp}$
7	$j_{\parallel} = j_{\parallel,oh} + j_{\parallel,ni}$	$j_{\parallel,ni}$, $j_{\parallel}$, $j_{\parallel,oh}$	
8	$j_{\parallel,oh} = \sigma E_{\parallel}$	$j_{\parallel,oh}$, $E_{\parallel}$	σ
9	$j_{tor} = \frac{B_0}{F\langle R^{-1}\rangle} j_{\parallel} + \frac{1}{2\pi\mu_0 \langle R^{-1}\rangle F^2} \left\langle \left\| \frac{\nabla\rho}{R} \right\|^2 \right\rangle \left\| \frac{\partial\psi}{\partial\rho} \right\|^2 F \frac{\partial F}{\partial\psi}$	j_{tor}	$j_{\parallel}$, B_0, F, $\langle R^{-1}\rangle$, $\left\langle \left\| \frac{\nabla\rho}{R} \right\|^2 \right\rangle$, $\frac{\partial\psi}{\partial\rho}$, $F\frac{\partial F}{\partial\psi}$

Table 1. Examples of definitions and constraints that are being used in the framework. The corresponding IDS data node of the symbols is shown in Figure 27. The indices e, i, and s indicate the electron, ion, and generic species. The k is the total number of ion species in the plasma. The bracket notation $\langle a\rangle$ indicates a flux-surface average of quantity, a. The p, q, f_{pq}, and $f_{\parallel}$ are the user inputs that are provided by the simulation user.

For the sake of flexibility and the explicitness of reciprocal relations, the declarative programming approach, opposite to the imperative programming approach, was adopted. In this approach, setting aside the method to keep consistency, only the definition (declaration) of a set of relations is required. In defining the relations, clarifying extra information about the node relations is essential for a computer code to conduct the updates automatically.

The first thing to clarify is whether the variables composing the reciprocal relation are input or output. Strictly speaking, the variables that are rarely considered as an outcome of a given relation are classified as input variables, and the rest becomes output. For example, in relation 4 in Table 1, all three variables (fast, total, and thermal density) are the output variables. On the other hand, in relation 5 (relation between the thermal density, temperature, and thermal pressure), the thermal pressure is the only one that can be calculated as an output, whereas the rest are categorized as input. In this case, the rest can only act as input because it is rare to use thermal pressure data to calculate thermal density or temperature. The same can also be applied to relations 1 and 2; the ion charges are usually not considered as outputs calculated from those rela-tions.

The reason for categorizing the input and output of a relation is to measure the priority of the relations. The notion of priority comes in to establish the calculation order of the relations. For ex-ample, suppose an NBI simulation is conducted. As the NBI simu-lation calculates the fast-ion density and its pressure (fast-ion pressure), the component should have updated $density\_fast$, $pressure\_fast\_parallel$, and $pressure\_fast\_perpendicular$ for the main ion in Figure 28. Then, it is natural to calculate $density\_thermal$, $pressure\_thermal$, and $pressure$ in order by rela-tion 4, 5, and 6, respectively. On the other hand, satisfying the re-lations in a reversed order, i.e., in order by relation 6, 5, and 4, would give an unexpected result. Therefore, the priority of the re-lations should be explicitly defined. For letting the framework know which relation has higher priority, it is measured how many other relations can be affected by the outputs from a given relation. Higher the number of affected relations, the higher priority the re-lation has. Although the method used to measure the priority is not strictly validated from a point of view of the graph theory and can-not guarantee that it would always work as intended, it has worked quite well under various kinds of simulations and workflows. The method used to measure the priority of the relations will be theo-retically validated and amended through further developments.

It is noteworthy that it is usual to assume some of the variables as fixed when satisfying a relation between three or more nodes and thus only a part of the variables from the relation are updated. Hence, it should be defined from which order the variables will be fixed. Specifically, it is necessary to arrange the output variables by order of constancy. Again, in relation 4 in Table 1, for instance, when a transport solver renews the thermal ion density while the change of the total and fast-ion density is unspecified, it is natural to change the total ion density while letting the fast-ion density be fixed. Also, it is natural to have fast-ion density fixed when conducting an interpretive analysis on the ion density evolution, as the fast-ion density would have been calculated from another component or given manually. When an NBI simulation updates the fast-ion density, it is natural to have total ion density unchanged unless otherwise specified, as most of the fast-ions would have originated from the charge-exchange reaction of beam neutrals with thermal ions. Overall, the fast density is the most invariant, while the thermal density is the least invariant. The same procedure can be applied to other relations.

Note that the priority only affects the default move of the data and hence if the data is determined by the component, it does not affect the update procedure. In the same example of the above paragraph, if the NBI code also considers the total deposition of the thermal ions due to the slowing-down of the fast ions during the calculation interval and ultimately gives both the thermal and fast ion densities, both densities are primary outputs from the component and the priority does not affect the calculation procedure of relation 4 in Table 1.

The relations should be defined in a form that accommodates the user input. In Table 1, some relations that require user inputs are listed. Relation 3 requires p and q to specify two species which are related, and f_{pq} to specify the density ratio between the two species. Relation 6 relates the total, the thermal, and the parallel/perpendicular fast pressure. The free parameter $f_{\parallel}$ determines the relative contribution of the fast particle parallel pressure to the isotropic total pressure, where it is usually taken to be a half

[77]. Also, it can be found that some of the relations can be applied to a generic specie and so the number of occurrences is greater than 1. It should be able to apply those relations for a different species in the same way. Hence, the relation definition should be general enough for reusability.

Finally, the relations are not only limited to within an IDS but also between the IDSs. Usually, the same definition is used but the grids where the data is defined are different between the IDSs. It can be found in relation 8, which is the relation between the Ohmic current, parallel electric field, and electric conductivity. Although not shown, the *core_transport* IDS, which contains the 1D transport quantities of the core plasmas, also has a dedicated node for the electric conductivity. The electric conductivity stored in the *core_profiles* IDS should be consistent with the data stored in the *core_transport* IDS. As the different radial grid is used for each IDS in general, interpolation of 1D profiles between the different IDSs is required and the data should be consistent for a valid simulation. The situation is much clear for relation 9, which is the definition of toroidal current density. Here, a couple of equilibrium metrics (flux-surface averaged geometric values) are required. However, as the radial coordinate used in the *equilibrium* IDS is different from the *core_profiles* IDS in general, interpolation is necessary to exchange data between the different IDSs.

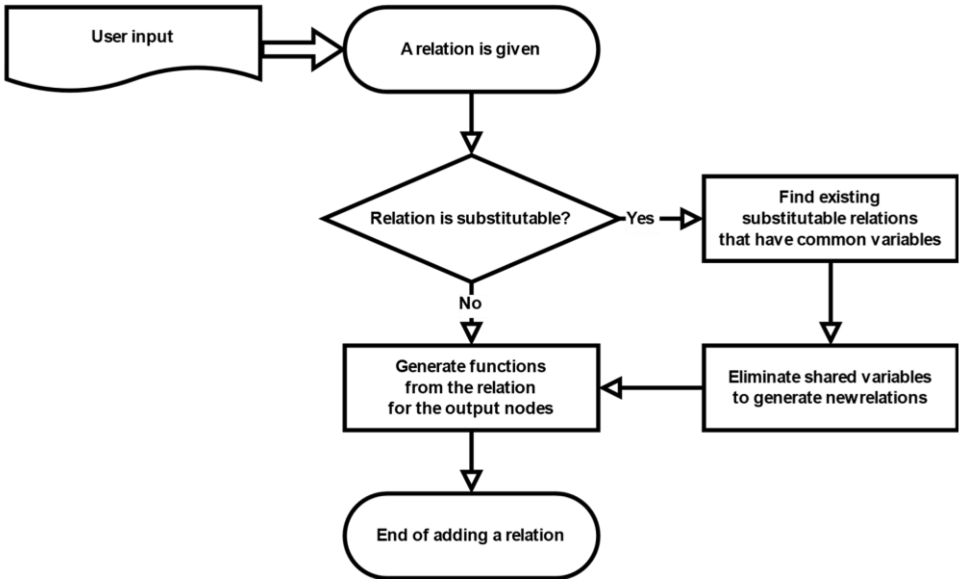
In total, 41 relations are defined and will further be added to cover a wider range of data consistency. Most of the relations are limited to a few IDSs, mainly on *core_profiles*, *equilibrium*, *core_transport*, and *core_sources*. Two relation definitions are shown as a code snippet in Appendix A.

3.3.2. Adding Relations in the Framework

After several prerequisites are clarified, the relations can be added to the framework to determine the update process automatically. Calculating the output node value using the other nodes' values requires a function, not a relation. Therefore, converting the relation into a set of functions is needed. In expression, the relation 4 in Table 1, $n_s = n_{s,th} + n_{s,f}$ should be converted into three functions, $n_s = f(n_{s,th}, n_{s,f}) = n_{s,th} + n_{s,f}$, $n_{s,th} = f(n_s, n_{s,f}) = n_s - n_{s,f}$, and $n_{s,f} = f(n_s, n_{s,th}) = n_s - n_{s,th}$, as there are three output variables available in the relation. For converting a relation into a set of functions, the symbol corresponding to a variable is allotted for every node while adding a relation in the framework. The symbolic manipulation to generate a set of functions from the relations was done by the Python library, SymPy [78], and further accelerated by SymEngine [79]. The functions are generated only for the nodes that are the output for a relation. The categorization of input and output nodes helped increase the performance in this stage.

It can be noticed that there exists a case where more than two nodes should be determined at a time. In other words, relations can form a simultaneous equation, and thus the variables should be substituted to solve the equation for a given variable. The case was already seen, where ion densities should be determined through relations 1 and 2 in TABLE 1, while the electron density and Z_{eff} was fixed. In this case, one of the ion densities should be eliminated by using two equations so to determine the other. The elimination of one of the shared variables could be done by symbolic manipulation in SymPy. However, doing such symbolic manipulation is time-consuming. Thus, in the framework, it was chosen whether the variables in the relations can be substituted (eliminated) or not. From the approach, the framework could satisfy the relations under the circumstances where the simultaneous equations exist, with a reasonable performance.

The flowchart for adding a relation during the simulation initialization is shown in Figure 29. Depending on the user input, the framework prepares a set of relations to add. When a relation is given, the framework first judges whether the relation is substitutable or not. If it does not, the relation is used to generate functions that can be utilized to calculate output nodes, and then adding a relation is finished. If the relation is substitutable, the framework finds existing substitutable relations that can compose a simultaneous equation. The elimination of shared variables in a simultaneous equation gives new relations. The new relations are then used to generate functions, as before.


User input
A relation is given
Relation is substitutable?
Yes
No
Find existing substitutable relations that have common variables
Generate functions from the relation for the output nodes
Eliminate shared variables to generate newrelations
End of adding a relation
Figure 29. The flowchart for adding relations during the simulation initialization.

3.3.3. Applying Relations

Once the relations are added, the simulation with the physics component can be started. Figure 30 shows the flowchart of the consistency loop, which keeps the internal data consistency after a component call. Following the procedure, after a component is called, the framework detects the nodes updated by the component. Detecting the updated nodes was implemented by adopting the data descriptor in Python. Briefly, the framework dynamically converts the data nodes into the data descriptors by defining the *__set__* method for the attribute. As a result, the *__set__* method is invoked on setting the attribute value in a code interface; the method then lets the framework know that the attribute value was updated. Checking the updates is technical and somewhat out of the scope of this thesis, so it will not be further discussed in this thesis. The main point is that the component does not have to let the framework know about the list of updated nodes, as other approaches do.

Given the updated nodes, the framework checks whether the update scenario exists or not. The update scenario is the sequence of the function applications done during the overall update process. For the first execution of the component, the update scenario would not exist. Then, as a next step, the framework checks whether an unsatisfied relation with only one undetermined node exists. If there is no such relation, the framework fixes the most invariant node in the relation with the highest priority until such relation is found. Fixing the most invariant node in the relation with the highest priority would eventually lead to the emergence of the relation that has only one undetermined node. Then, the undetermined node can be calculated by the relation. After iterating the procedure until all relations are satisfied while recording all the function calls, the update scenario is generated and saved.

There will be a limited number of components in a simulation or a workflow, and a component will update the same nodes at every call. So, from the second call of the component, the saved update scenario can be utilized to update the data nodes for consistency. Saving the update scenario could reduce the time spent on checking whether a relation is satisfied in a nested for−loops.

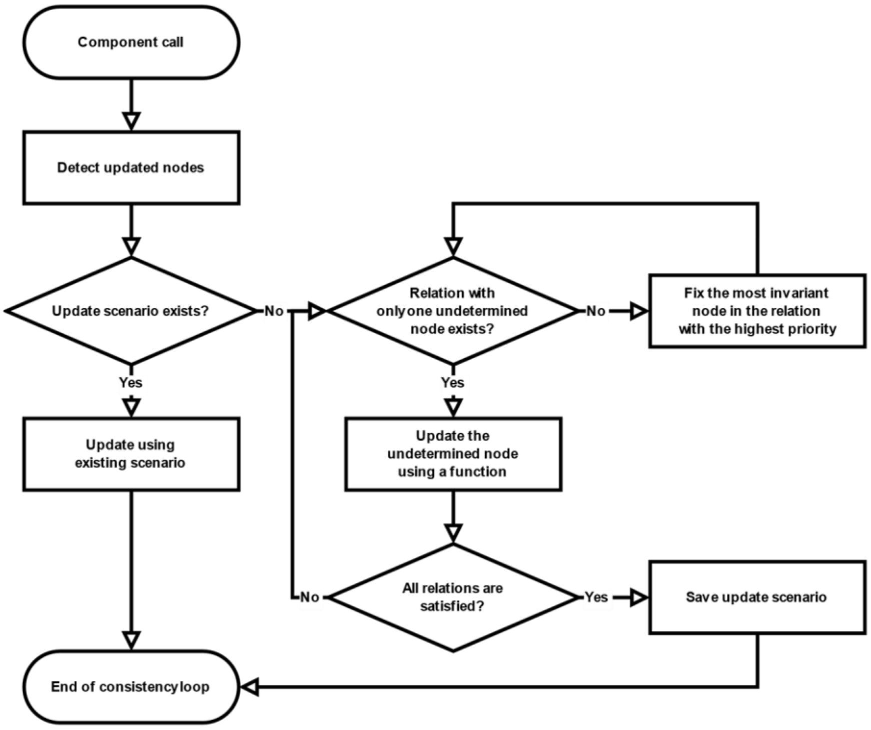

Component call
Detect updated nodes
Update scenario exists?
No
Yes
Update using existing scenario
Relation with only one undetermined node exists?
No
Fix the most invariant node in the relation with the highest priority
Yes
Update the undetermined node using a function
All relations are satisfied?
No
Yes
Save update scenario
End of consistency loop

Figure 30. The flowchart for applying relations after a component call.

Although not shown in Figure 30 for simplicity, if not all rela− tions are satisfied while all the nodes are fixed, it raises an error. In other words, the data consistency is checked even if the data nodes in an IDS are fully updated by a component. The framework can be used to check whether a component keeps the internal data consistency of the nodes, as there can exist human mistakes in calculating data nodes in a component.

3.4. Performance and Flexibility of the Framework

3.4.1. Performance Enhancement

The performance of the simulation was greatly improved from the previous imperative approach. Figure 31 shows the elapsed time for the consistency loop on the calculation step of various simulations. The calculation speed of the previous imperative ap- proach was about 100 steps/s. While the approach does not con- sume time during initialization and the speed is maintained for the entire range of the calculation steps, it takes about 13 s to initialize (add relations) the simulations for the improved approach. Note that the time required for the initialization varies by the complexity of the simulation such as the number of ion species, number of heating/current drive sources, and number of transport models.

The calculation speed of the full (an unrealistic component that intentionally updates various kinds of data nodes) component is 80 steps/s, and it is even slower than the previous approach. This is because TRIASSIC was upgraded to cover more data than those covered in the previous version. Anyhow, it is reasonable that the improved approach shows a similar performance to the previous approach when all the calculations defined by the node relations were intentionally turned on. On the other hand, the calculation speed was significantly accelerated to 2000 steps/s for the null (an unrealistic component that does nothing to suppress the update procedure after the code execution) component. For the normal simulation components shown as a green shaded region in Figure 31, the calculation speed was faster than 350 steps/s which is ac- celerated at least 3.5 times from the previous approach. The sim- ulation was indeed accelerated by removing unnecessary calcula- tions for consistency.

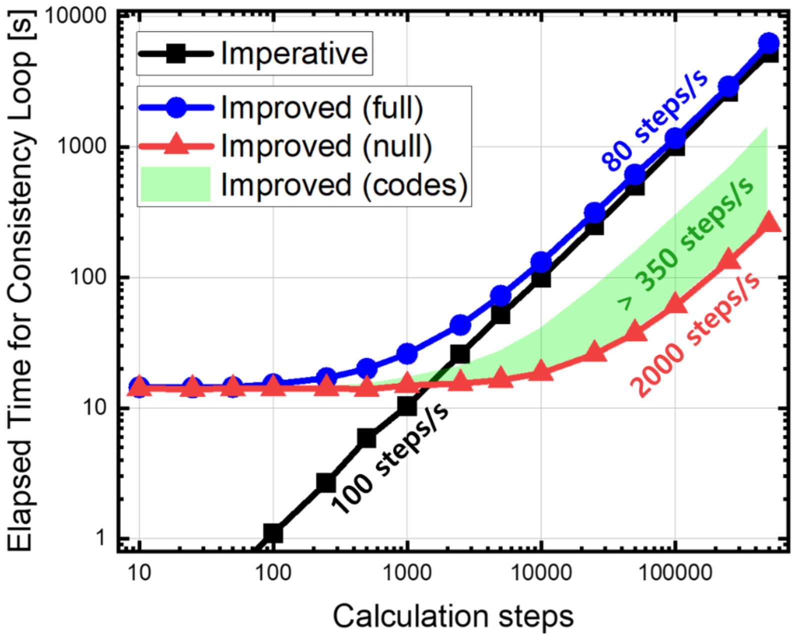


Figure 31. The comparison of performances of imperative method (black) and improved methods. The *full* indicates the code component intentionally updates various data nodes in various IDSs, and the *null* indicates the code component updates nothing. Performances of normal code components are shown as a green shaded region.

It is noteworthy that the consistency loop can consume considerable time relative to the total elapsed time for the simulation. Figure 32 shows the proportion of the elapsed time for the consistency loop relative to the total elapsed time. It can be found that as the complexity of the model and the calculation step decreases, the consistency loop can take significant time consumption. An example of time-dependent predictive simulation in the improved TRIASSIC showed that 3% of the total elapsed time was consumed for the consistency loop. Considering the improved performance, the consistency loop would have taken up at least 10% of the total elapsed time or even higher before the improvement.

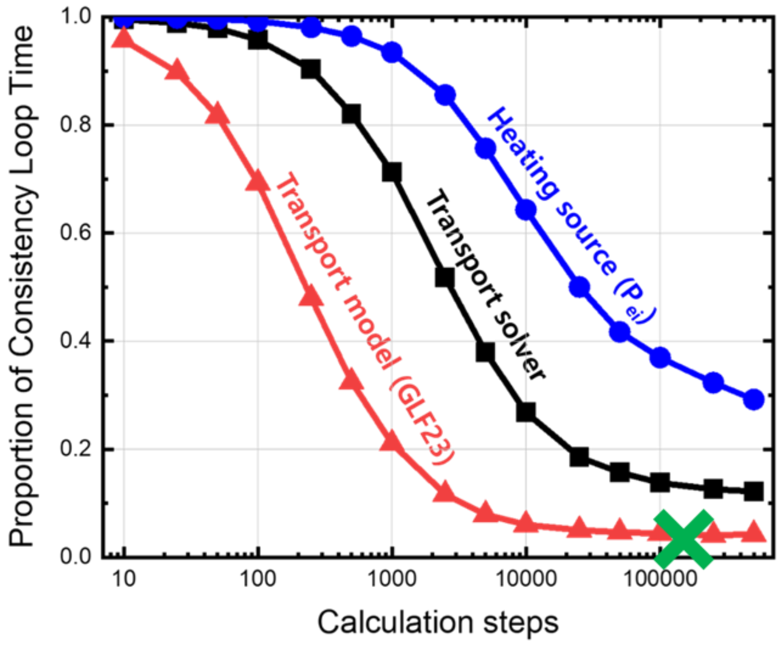


Figure 32. The proportion of the elapsed time for the consistency loop for three different physics codes (heating source, transport solver, and transport model) and various calculation steps. The time spent on the consistency loop becomes significant as the calculation step decreases and the model calculation gets faster. An example time−dependent predictive simulation case is marked as a green marker.

3.4.2. Flexibility and Maintenance of the Framework

So far, in the various integrated transport modeling codes, the segregation between the electron solver and the ion solver in a simulation was somewhat strict [16]. This is because the situation depicted in section 3.1 and Figure 23 is complex and it requires cumbersome use-case studies to flexibly enforce the quasi-neutrality constraint while solving the particle transport. In the electron solver, the impurity ions were usually assumed to be proportional to the electron density while letting the quasi-neutrality condition determine the main ion density. On the other hand, the electron density was set to satisfy the quasi-neutrality constraint in the ion solver by summing the ion charge densities. Due to the quasi-neutrality constraints, it was difficult to conduct a transport modeling by selecting some species to be predicted among the electron and the ions. The declarative method implemented in TRIASSIC could generalize the quasi-neutrality constraint and enable the mixing of electron and ion species to be solved.

The strength of this work can be emphasized in predicting the burning plasmas of a future reactor such as ITER or DEMO. In such burning plasmas, it is expected that there exists a lot of non-negligible ion compositions, other than the deuterium (D) and tritium (T) fuel. For example, hydrogen and helium (both helium-3 and helium-4) can be generated by the D-D and D-T fusion reactions. Moreover, in such large devices, seeding impurities such as nitrogen or argon would be essential to reduce the heat flux on the divertor target [80]. There can also exist sputtered wall/divertor materials such as tungsten or beryllium. By considering the whole ion in the framework, for example, when the research focuses on the tungsten and helium-ash accumulation in the core plasmas, one can solve the transport of the electron, helium, and tungsten while assuming the seed impurity density proportional to the electron density and letting the amount of D and T to cover the remaining charge for the quasi-neutrality condition.

Figure 33 shows the calculation of plasma flows and the radial electric field in the framework. When the plasma densities and the temperature is provided by the experimental measurements, the diamagnetic flows of the main ion and the impurity ion can be calculated. Also, the poloidal flow of the ion species can be calculated by the neoclassical model. Frequently, the toroidal flow velocity of the impurity ion specie is provided by the charge exchange spectroscopy (CES) measurements. In such a case, by gathering the diamagnetic, poloidal, and toroidal flow information of the impurity ion specie, the radial electric field can be calculated by using the radial ion force balance equation. The calculated radial electric field is then used to calculate electrostatic potential and the $E \times B$ shearing rate. Assessing the $E \times B$ shearing rate is important as it is well-known that it significantly affects turbulent transport. The radial electric field can eventually be used to calculate the toroidal flow velocity of the main ion again by the radial ion force balance equation.

Note that the ion force balance equation is used twice; while the toroidal rotation was used to calculate the radial electric field for the impurity ion specie, the toroidal rotation was the resultant for the main ion species. As seen in the example, the plasma flows of multiple ion species and the radial electric field can form a complex relationship, and depending on which specie' s toroidal flow is provided, the calculation order of the data node can vary. The situation gets even more complex when the second impurity specie such as oxygen is considered and the toroidal rotation of the given specie is measured. The declarative programming approach of TRIASSIC flexibly copes with complex situations without cumbersome checks of which data of which specie is provided.

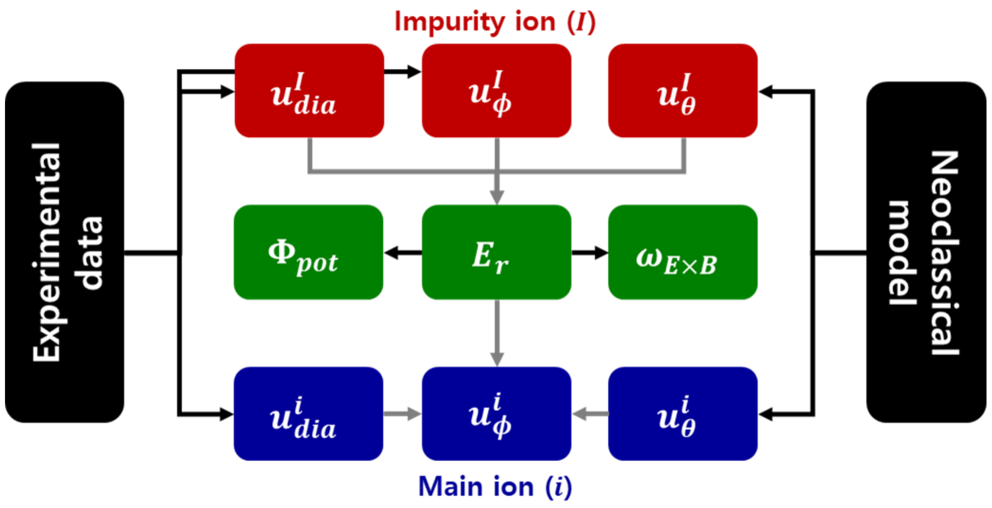


Figure 33. The calculation procedure of the flows of impurity and main ion, and the radial electric field.

As was discussed in section 2.2.2, the unique code interfacing requires the framework to do the rest of the calculations for internal data consistency. While in EU-IM workflows the IDS should be added to the I/O routine of each physics code and the consistency is kept by the physics code itself, the physics code in TRIASSIC does not contain the IDS in them and so the framework should do the rest of the calculations for data consistency. The comparison is depicted in Figure 34. The last approach addressed in section 3.2 enabled unique code interfacing in TRIASSIC. Moreover, the choice of declarative programming concept in the relation definition helped the framework cope with any kind of situation and made it easy to interface a new code with the framework, as the consistency is automatically satisfied after the code execution. In other words, the flexibility gained from the declarative programming approach simplified the code interfacing task and so the extensibility was enhanced, and the code maintenance became easier.

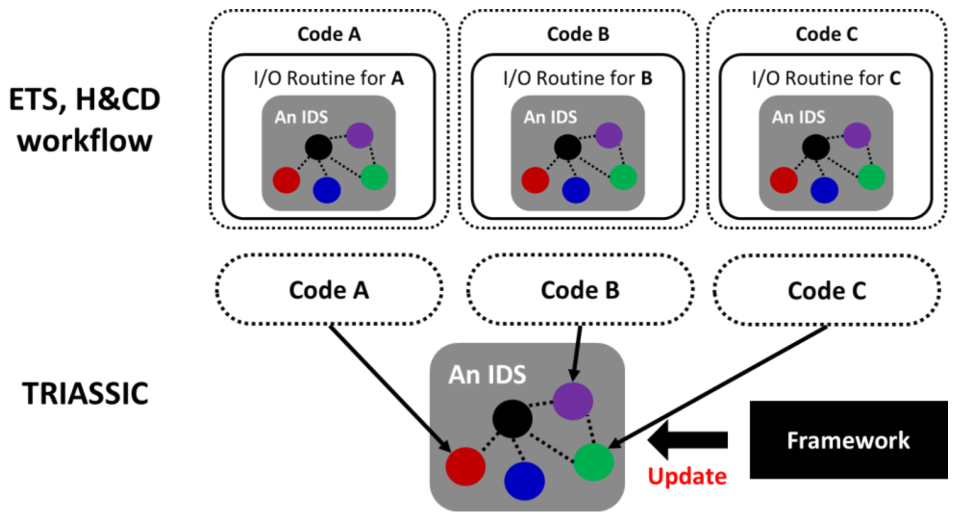


Figure 34. The comparison of the EU−IM workflows and TRIAS−SIC. The components in EU−IM workflows should be significantly modified to contain IDS in their I/O routine and to keep consistency within the IDS. On the other hand, the components in TRIASSIC do not consider consistency as the framework manages to keep the consistency between the data nodes.

Figure 35 shows the comparison of legacy frameworks, tiers 1, 2, and 3 frameworks, and TRIASSIC in terms of the level of IDS utilization and the simplicity of the legacy code interfacing. Com−pared with the legacy frameworks and the tier 3 approaches such as OMFIT and IPS, TRIASSIC uses the IDS as native data for a framework. Although the data structure adopted in the tier 3 frameworks are also well−defined and generic, using the standard and generic data, TRIASSIC is more extensible in terms of the new models. For the frameworks that fully adopt the IDS, as aforemen−tioned, TRIASSIC was improved from the tiers 1 and 2 approaches by simplifying the interfacing procedure. It takes less effort for interfacing with a physics code that already exists.

Lastly, it is noteworthy that the improvement manages to keep consistency not only within the IDS but also between the IDSs. Unlike other workflows that require additional non−physics actors to transfer data between the IDSs, the data nodes between the IDSs are automatically kept consistent in TRIASSIC. Compared to the other workflows which demand significant tasks as the physics components and non−physics components should be properly placed therein, a workflow can be easily composed in TRIASSIC only by arranging the code components in an appropriate order. While the former workflow−oriented approach is proper when the analysis target is limited to some cases, it is nearly impossible for a user to compose a new workflow in a short term for a new type of analysis. TRIASSIC gains huge merit in terms of the flexibility of the simulation type. Various workflows could be made by the improvements; the applications will be shown in the following chapter.

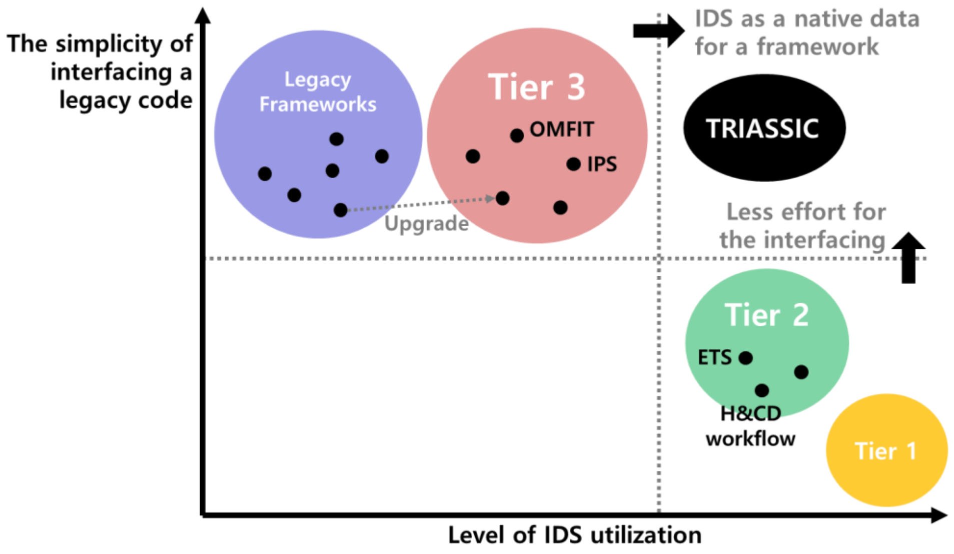

The simplicity of interfacing a legacy code
Legacy Frameworks
Tier 3
OMFIT
IPS
Upgrade
IDS as a native data for a framework
TRIASSIC
Less effort for the interfacing
Tier 2
ETS
H&CD workflow
Tier 1
Level of IDS utilization

Figure 35. The comparison of TRIASSIC and other frameworks in terms of the level of IDS utilization and the simplicity of interfacing with a legacy code. TRIASSIC uses IDS as native data for a frame−work, while less effort for the interfacing is required when com−pared with the tiers 1 and 2 approaches.

Chapter 4. Applications to Various Devices

4.1. Applications to KSTAR

4.1.1. Kinetic equilibrium workflow and its validation

The schematic view of the kinetic equilibrium workflow is shown in Figure 36. The workflow is composed of the equilibrium solver, neutral beam injection code, neoclassical model, and the current diffusion solver. In the workflow, the current diffusion is solved while updating the plasma equilibrium, neoclassical transport (bootstrap current), and beam absorption (NB current drive), while the plasma density and temperatures are provided by the measurements. The iteration of four components gives the safety factor profile and total stored energy at a stationary state. Here, the validation is focused on the total stored energy, and hence mostly the validation is about the test of the neutral beam model as the fast ion energy assessed by the model significantly affects the total stored energy calculation.

In the simulations, the total stored energy ($W_{Interpret}$) calculated by summing all the thermal plasma energy and the fast-ion energy was compared with the energy calculated by the EFIT code (W_{MHD}). The EFIT code assumes the total pressure as a simple formula with some adjustable coefficients and finds an optimal solution that min-imizes an error based on magnetic measurements such as magnetic pick-up coils and flux loops (magnetic-EFIT). The EFIT code is a standard way of calculating the internal structure and some global quantities of the plasma, and it has long been used as a reference equilibrium reconstruction code for obtaining plasma energy, β, and plasma shaping in KSTAR [81]. In this analysis, it was assumed that the total stored energy calculated from the EFIT code based on the magnetic measurements well represents the total stored energy so that the additional information from Motional Stark Effect measurements (MSE-EFIT) is not needed.

The thermal part of the total energy was calculated using the measured electron density and temperature from the Thomson scattering (TS) diagnostic and the ion temperature from the charge exchange spectroscopy (CES) diagnostic. It was assumed in the workflow that all the ions have the same temperature due to frequent collisions between them, and the effective charge Z_{eff} is equal to 1.9 based on previous research [82, 83]. It is noteworthy that the validation of the neutral beam model is essential, particularly for predictive simulations, as the time evolution of plasma largely depends on the particle and H&CD sources. The validation of kinetic equilibrium workflow can be the first step of the validation of whether the fast ion orbiting and their slowing down are properly addressed or not.

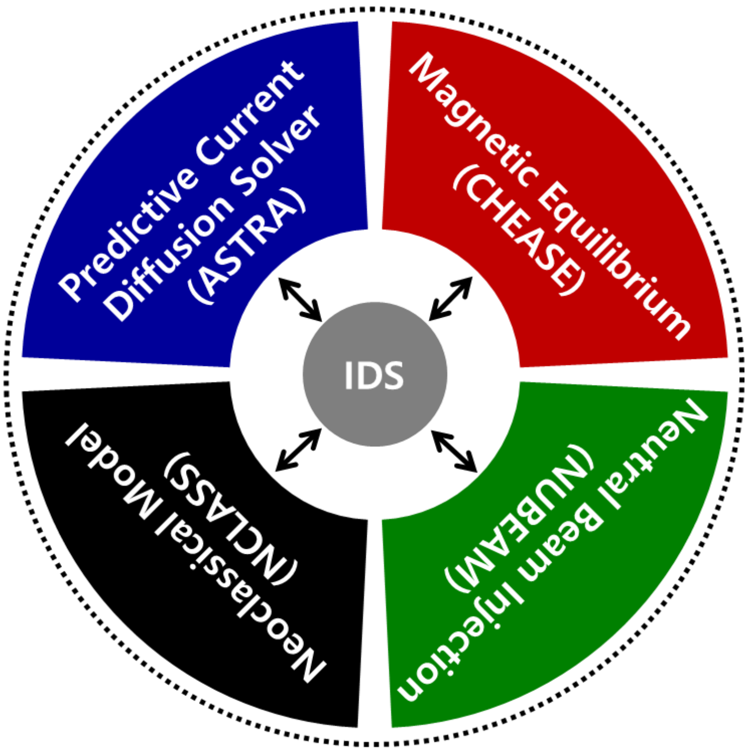

Predictive Current Diffusion Solver (ASTRA)
Magnetic Equilibrium (CHEASE)
IDS
Neoclassical Model (NCLASS)
Neutral Beam Injection (NUBEAM)

Figure 36. The kinetic equilibrium workflow in TRIASSIC. The workflow is composed of the magnetic equilibrium code (CHEASE), neutral beam injection code (NUBEAM), neoclassical model (NCLASS), and current diffusion solver (ASTRA).

The comparison of W_{MHD} and $W_{Interpret}$ is shown in Figure 37. Figure 37 (a) shows that the interpretively calculated energy $W_{Interpret}$ well lies on the $y = x$ line versus W_{MHD} on the x−axis. Figure 37 (b) indicates that the value of $W_{Interpret}/W_{MHD}$ does not deviate much from 1.0, as its average and standard deviation are 1.06 and 0.12, respectively. On the high tail of the histogram, there were high β_P discharges whose $W_{Interpret}$ was somewhat overes− timated from its W_{MHD} values. The β_P values in those discharges were the highest (> 2.2) among the database, while the average value of β_P was 1.5 in the database. Those high β_P discharges showed the TAE activity due to an absence of EC wave H&CD [84].

The overestimation of $W_{Interpret}$ in the simulation might have occurred from neglecting anomalous fast-ion transport, which has risen from the TAE activities. The deviation of $W_{Interpret}$ was not significant when the TAEs were successfully stabilized by the EC heating/current drive to suppress them. A preliminary application of the linear TAE model [70] with an arbitrary saturation rule on these discharges showed that this overestimation could be relaxed to some extent.

The remaining high tail part on $W_{Interpret}/W_{MHD} > 1.2$ was from the Argon impurity injection experiment, shown in a black circle in Figure 37 (a). In this experiment, the injected impurity would have diluted the hydrogenic ions and raised Z_{eff} under the same electron density level. However, as Z_{eff} was fixed to be 1.9 in this modeling, this low Z_{eff} assumption might have caused an overestimation of the total ion population and the resulting thermal energy. The overestimation of the stored energy was not observed before the impurity injection in the same discharge. An additional simulation with an increased Z_{eff} setting to 3.1 showed that the overestimation could be relaxed to a normal range, which casts the need for further validations with impurity measurements as future work. Note that robust measurements of Z_{eff} are not available yet in KSTAR.

On the other hand, there were no cases with a large discrepancy on the low tail of the histogram. The previous study [85] reported that resonant magnetic perturbation (RMP) could induce fast-ion loss in KSTAR, but no significant energy deficit due to the RMP is found in this comparison. We note that the database used in this study does not include the discharge with a strong core field penetration. As the fast-ion loss induced by the RMP mainly occurs in the core regime where fast-ions are abundant, this may explain why such an RMP-induced fast-ion loss effect was not observed.

Excluding the significantly deviated discharges in which the cause of the deviation is clear, the modeling could more accurately find the total energy. In such a case, the average, and the standard deviation of $W_{Interpret}/W_{MHD}$ were 1.04 and 0.09, respectively.

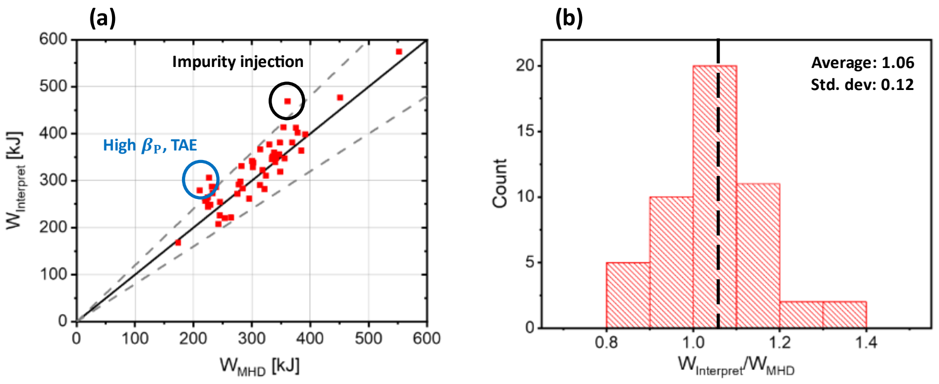


Figure 37. The comparison of the plasma energy calculated by the equilibrium reconstruction code and that calculated by the interpretive simulation. The $y = x$ line is shown as a black solid line, with two dashed gray lines representing a 20% deviation where two kinds of discharges (high β_P TAE and impurity injection) with significant deviations marked in circles. (b) The histogram of the interpretively calculated energy divided by the energy calculated by the equilibrium reconstruction, where the average and the standard deviation of the value are shown.

4.1.2. Stationary−state predictive modeling workflow

The stationary−state predictive modeling workflow contains the kinetic equilibrium workflow and more models such as heating/cooling models, a neutral model, and an anomalous transport model were added, as shown in Figure 38. In this workflow, the transport solver solves not only the poloidal field but also the plasma density and energy. The prediction of plasma quantities was solved for a sufficiently long time (stationary−state) when com− pared with the current diffusion or the energy confinement time. The other components, such as neoclassical and anomalous transport, NBI, ECH, other heating/cooling, and neutrals, were re− calculated to reflect the change of the plasma quantities so the simulation is self−consistent. The outcome of the simulation is mainly the plasma density and temperature, as the quantities are deeply related to the plasma performance. To check the validity of the simulation workflow, the predictive modeling results were compared with the experimental measurements of plasma energy, density, and temperature.

In the simulations, the ion densities were set to be proportional to the electron density with $Z_{eff} = 1.9$ as before, which is the same as assuming 3% of the carbon density relative to the electron den− sity. For all the particle and heat transport channels, the quantities were fixed at the boundary position, $\rho = 0.8$. Other quantities such as plasma current, vacuum magnetic field, and plasma shape were fixed during the predictive simulation. The original saturation rule (SAT0) [86] was used in the TGLF model, and the transport shortfall, which frequently occurs in the high β_P plasmas [87], was observed for the high β_P plasmas. We note that the high β_P plas− mas were excluded in this predictive modeling due to this transport shortfall issue. The predictive simulation was conducted for a suf− ficiently long time of $0.4\,s$, which is larger than the typical energy confinement time for H−mode plasmas in KSTAR ($\cong 0.1\,s$), to find a stationary condition for a given plasma/actuator condition. The predicted stationary plasma was then compared with the experi− ments.

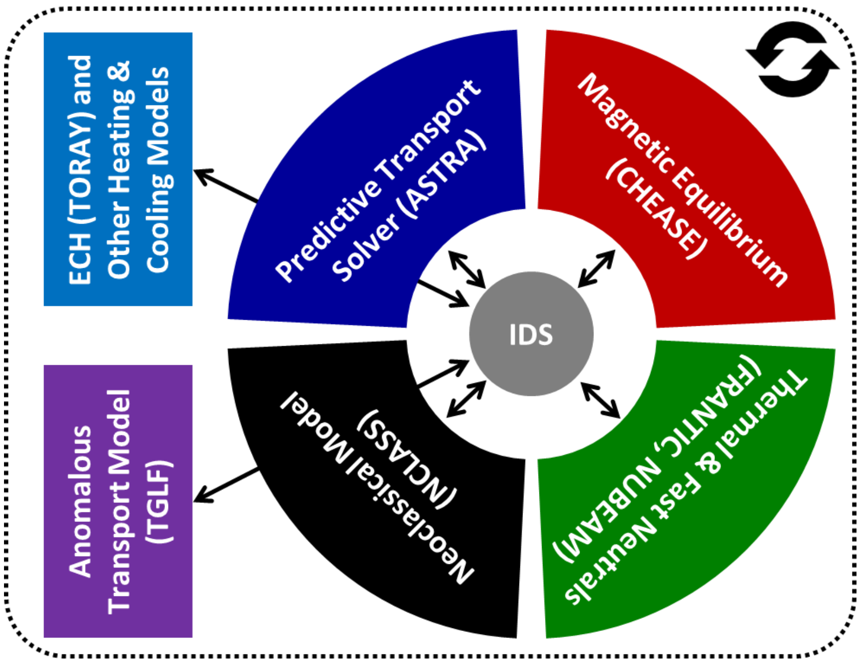

ECH (TORAY) and Other Heating & Cooling Models
Predictive Transport Solver (ASTRA)
Magnetic Equilibrium (CHEASE)
IDS
Anomalous Transport Model (TGLF)
Neoclassical Model (NCLASS)
Thermal & Fast Neutrals (FRANTIC, NUBEAM)

Figure 38. The stationary−state predictive modeling workflow. The heating/cooling models (TORAY, Ohmic heating, etc.), thermal neu− tral model (FRANTIC), and anomalous transport model (TGLF) were added from the kinetic equilibrium workflow shown in Figure 36.

The predictive simulation showed a significant underestimation of the density level when the cold neutral from the wall recycling was not considered. Such underestimation is depicted in Figure 39 (a). In this figure, the line−averaged electron density from the prediction is shown as gray circles, which are much smaller than the experimental values. The lack of particle source from the wall neutral ionization can be the reason for this discrepancy. Accord− ingly, the effect of wall recycling was considered by assuming a constant influx of cold neutrals, and the neutral influx value was scanned to find the best match with the measured density level. The neutral influx value determined by the scan was $6 \times 10^{20}\, m^{-2} s^{-1}$. The prediction of energy level was also increased when the cold neutral from the wall was considered, as shown in

Figure 39 (b).

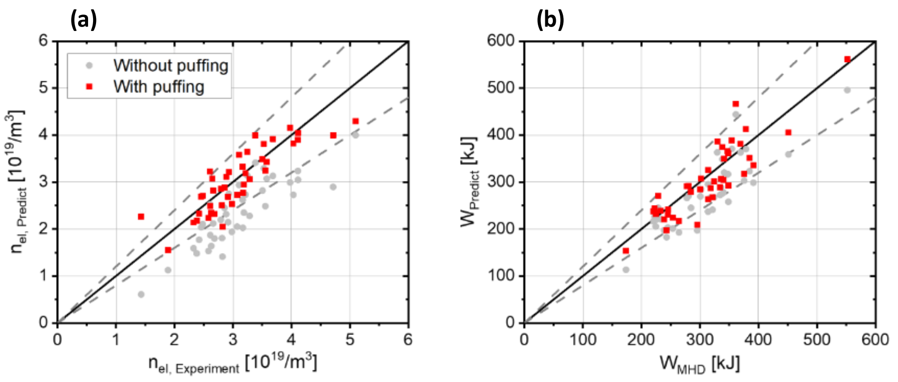


Figure 39. The comparison of the experimentally measured line-averaged electron density with the predicted one (a) and the comparison of W_{MHD} with predicted total energy ($W_{Predict}$) (b). The $y = x$ line is shown as a black solid line, with two dashed gray lines representing a 20% deviation. In both figures, the prediction result with and without the neutral gas puffing is shown as red squares and gray circles, respectively.

The validation result of the predictive modeling is evaluated in Figure 40 and Figure 41. The average value of the relative amount of predicted line-averaged electron density ($n_{el,Predict}/n_{el,Experiment}$) was 0.99 by scanning the amount of the neutral influx. Simultaneously, by increasing the neutral gas puffing rate, the average value of the relative amount of predicted energy ($W_{Predict}/W_{MHD}$) was increased to 0.97. The agreement of $W_{Predict}$ and W_{MHD} indicates that the predictive modeling under a constant neutral gas puffing is a reasonable way to predict both density and energy consistently. Also, it should be noted that the standard deviation of the density and the energy prediction was 0.14 and 0.12, respectively, which indicates that the density and the energy can be predicted with high confidence.

The simulation was also validated by comparing the predicted plasma temperatures with diagnostic measurements, as shown in Figure 41 (a). The volume-averaged electron and ion temperature from predictive modeling showed a good agreement with their counterpart from experimental measurements. The distribution of the ratio between the predicted and measured temperature for both electron and ion are shown in Figure 41 (b) and (c), respectively.

The ion temperature prediction was accurate with its average value of 1.02; however, the electron temperature prediction showed an underestimation with its average value of 0.92. It was observed that the electron temperature profile from predictive modeling tends to have a downward-convex shape, whereas the shape is upward-convex for the experimental fit, as shown in the electron temperature profiles in Figure 42 (a). This difference in electron temperature profile shape at $R_{major} \cong 2.0\,m - 2.15\,m$ was the main cause of the overall underestimation. It is hard to determine the exact shape of the electron temperature profile from TS measurements, and for further validation of electron temperature prediction, it requires a firm experimental fit with additional temperature diagnostics such as the electron cyclotron emission measurement.

A few cases showed non-negligible discrepancies against the experimental measurements. In the low-density range of Figure 39 (a), the density was highly overestimated. The reason for this overestimation might be due to a too high neutral gas puffing rate, as the experiment was conducted under the low prefill gas condition to operate with a low-density condition. Although constant neutral gas puff modeling can be a good approach to predicting thermal plasmas, indeed, it might need a shot-by-shot adjustment of puffing rates based on the neutral pressure measurements. As an alternative approach, determining the amount of wall-neutral influx by assuming a constant recycling rate [88] is also possible.

The approach will be tested in future work to dissolve the limitations of the current approach and improve the predictive modeling capability.

The high-Z impurity injection experiment, which was already discussed in the validation of the interpretive simulation above, was a case with a high energy estimation in Figure 39 (b) and Figure 40 (b). The case with significant underestimation of the ion temperature shown in Figure 41 (c) was the internal transport barrier (ITB) discharge. The formation of ITB was not seen in the predictive simulation, and it might be due to the absence of delicate modeling of the current profile, as the onset of ITB could be related to the safety factor profile in KSTAR [89]. For the remaining cases with a significant deviation of the ion temperature prediction in Figure 41 (c), the two-Gaussian fitting [90] was used to calculate the ion temperature from the CES measurement instead of using the usual beam modulation technique since the beam modulation was not performed in those discharges. A significant prediction error would have appeared due to this difference in the ion temperature calculation method. Other cases with a significant deviation were in the normal operating range. A significant error might be due to a wrong experimental fitting, especially for the electron temperature or the non-stationarity, and will not be further discussed in detail. It is noteworthy that the differences in predictability between L-mode and H-mode plasmas were not found, and the models used in the simulations have their own shortcomings anyway.

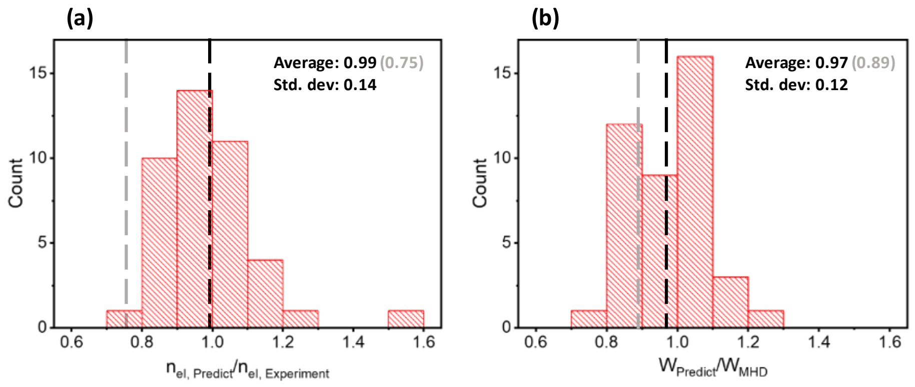


Figure 40. The histogram of the predicted electron density divided by the experimental electron density (a) and the histogram of the predicted energy divided by W_{MHD} (b) for the neutral gas puffing predictive simulations. The average values for simulations with and without the gas puffing are marked as a black and gray dashed line.

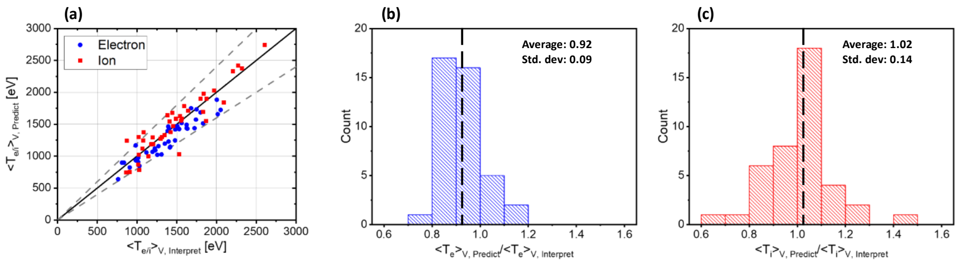


Figure 41. (a) The comparisons of interpretively calculated volume-averaged temperature ($\langle T_{e/i}\rangle_{V,interpret}$) with the predicted one ($\langle T_{e/i}\rangle_{V,predict}$), where the electron and the ion are shown as blue circles and red squares, respectively. The $y = x$ line is shown as a black solid line, with two gray dashed lines representing a 20% deviation. The ratio between the predictive and the interpretive volume-averaged temperature is shown for both electron (b) and ion (c). The average values are marked as a black dashed line.

The electron density, electron temperature, and ion temperature profiles from a well-reproduced prediction case and the worst prediction case are shown in Figure 42. In the well-reproduced case, the electron density was very flat in the core region without the cold neutrals, but the peaking was recovered if the wall-neutral ionization source was considered. On the other hand, the electron and ion temperatures were slightly decreased due to neutrals. The deviations from the experimental level of density, electron temperature, and ion temperature were +2%, −6%, and +5%, respectively. The worst prediction case corresponds to the ITB discharge. The steep gradients in both temperature profiles were not reproduced, and the underestimation of ion temperature is prominent. In this case, the deviations of density, electron temperature, and ion temperature were −16%, +19%, and −33%, respectively.

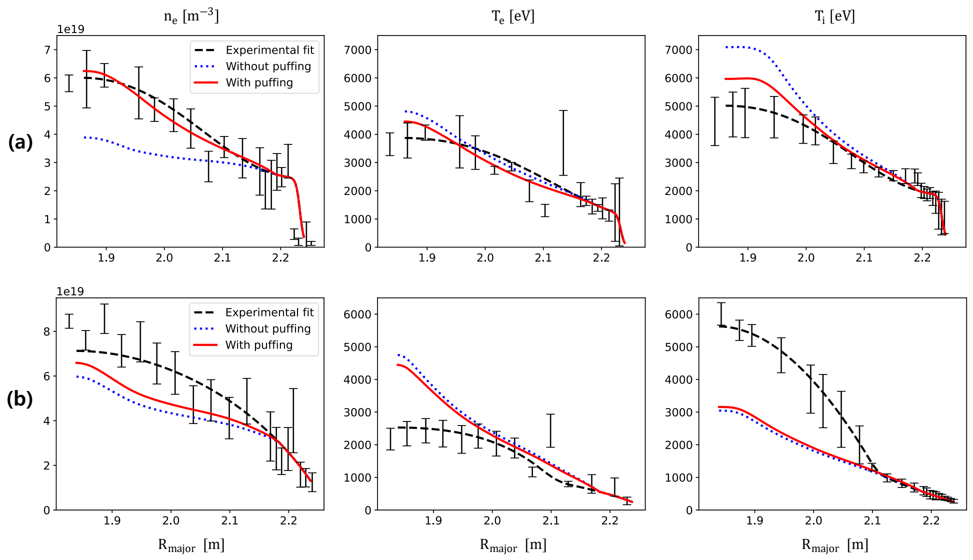


Figure 42. The profiles of the electron density (left), electron temperature (middle), and ion temperature (right) from the well-reproduced case (a) and the worst prediction case (b). The density and temperatures from experimental measurements are shown as black error bars, and its experimental fit is shown as a black dashed line. The predictive simulation results with and without neutral gas puffing are shown as red solid and blue dotted lines, respectively.

4.2. Application to VEST

4.2.1. Time−dependent predictive modeling workflow

The TRIASSIC suite of codes was applied to the VEST [91] device by composing a time−dependent predictive modeling work−flow, as shown in Figure 43. For the simulation through the work−flow, the plasma boundary information was prepared by the equi−librium analysis code [92, 93]. Also, the simulation was set to fol−low the plasma current measured by the Rogowski coil. As the workflow has the same target (predictive simulation) with the workflow introduced in the previous section, the two workflows share many common grounds. However, different from the station−ary−state workflow, as the simulation targets to address the time−dependent behavior of the plasmas, basically the time interval for the models is much shorter than the workflow adopted in the KSTAR modeling.

For the validity of the predictive simulation in the ST device, models were deliberately selected and adopted in the workflow. The TGLF model is known to be valid in the ST device due to the improvements from its predecessor, the GLF23 model, in terms of the geometry and the treatment of trapped particles. Also, as the low−temperature plasma is expected at least at the near−edge re−gion of the plasmas, there are high chances of finding impurities that are not fully stripped. Hence, the radiative energy loss from the line emission should be accurately considered. For the calcu−lation of steady−state line emission, ionization, and ionization cooling, the reaction rates from the ADAS collisional−radiative model were adopted. Lastly, to reproduce the rapid increase in the electron density observed by the Thomson scattering diagnostics, the FRANTIC module was used to simulate the pre−fill gas in ex−periments. The data provided by the experiment and the workflow are shown in Figure 43.

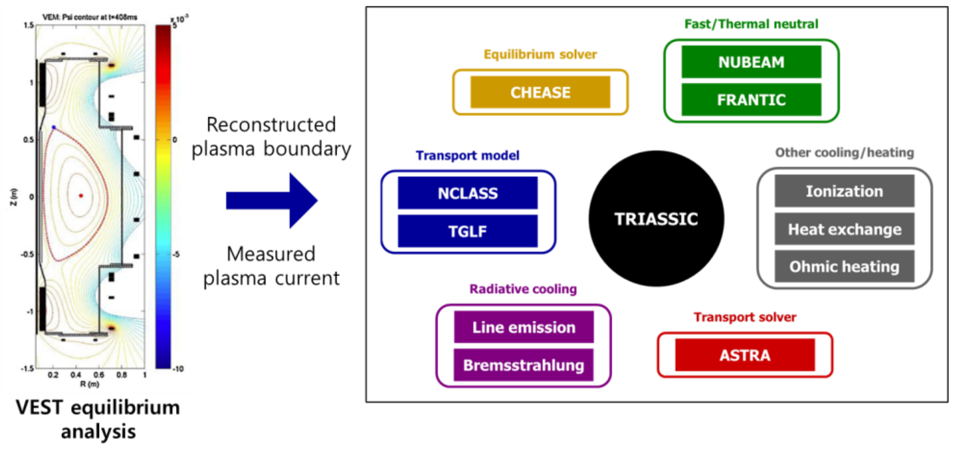


Figure 43. The time-dependent predictive transport simulation workflow. The reconstructed plasma boundary and the measured plasma current are used as input for the predictive simulation.

The predictive simulation was set to follow the experimental values that are provided while solving the particle transport, heat transport, and current diffusion equations. For the boundary condition of the particle and heat transport, it was assumed that the particle and heat fluxes are proportional to the boundary density and energy density. The coefficients were adjusted until reasonable values of density and temperature are found, based on the triple Langmuir probe measurements outside the plasma boundary. The overall simulation setting was quite similar to the previous predictive modeling approach on VEST [94].

The predictive simulation was conducted several times by varying the amount of the gas influx. Figure 44 shows the predicted electron density and temperatures by the simulation, for three different gas influx conditions. The electron density and the temperatures increase in time even during the ramp-down phase after the current max timing at $0.314\,s$. The tendency is supported by the TS measurements. It can be noticed that the pulse duration of VEST is not long enough for plasma to reach the stationary state and indeed composing a workflow with sufficiently short time intervals for the time-dependent prediction is essential for the comparison with measurements.

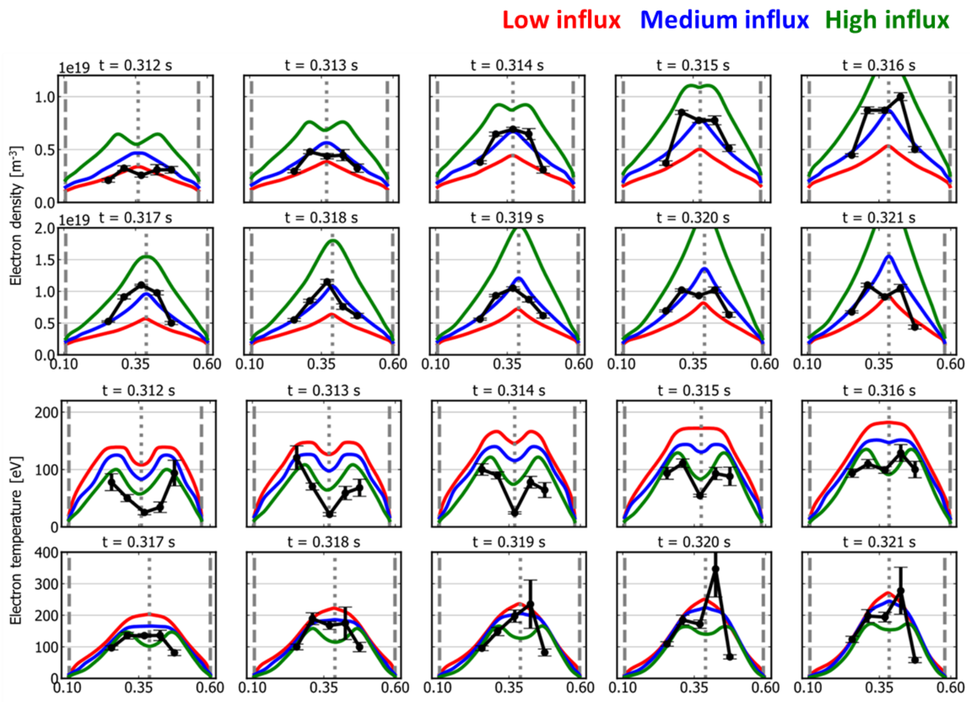


Figure 44. The electron density (upper two rows) and temperature (lower two rows) profiles predicted by three simulations of low (red), medium (blue), and high (green) gas influx. The Thomson scattering measurements are shown as black markers and lines.

As was intended, by increasing the amount of pre-fill gas, higher electron density and lower electron temperature was pre-dicted. The predictive modeling could reproduce the overall elec-tron density evolution during the whole simulation time, especially for the medium gas influx case. Also, the electron temperature evolution after the current-max timing (0.314 s) showed good agreement with the experimental measurements, and the hollowness of the electron temperature profile was found both in the simulations and experiment. However, although the evolution of the electron density is well captured in the medium gas influx simulation, the simulation workflow could not reproduce a super-low electron temperature at the early phase (0.312 s - 0.314 s), even for the highest gas influx simulation. The poor prediction of the electron temperature at the early phase slowly recovers to the tolerable range after the current-max timing.

The predictive modeling workflow should be further validated by dedicated experiments with more measurements. Especially, the ion temperature predicted by the simulation should be compared with the ion temperature measured by ion Doppler spectroscopy [95, 96]. Also, the electron density measured by interferometry should further be adopted to cross-check with the Thomson scattering measurement or to be compared with the prediction results. On the other hand, the validity of the modeling workflow should be assessed in terms of the steady-state neutral redistribution and steady-state collisional radiative model assumption. The modeling workflow will be improved in the future and will be validated with dedicated experiments for valid predictive simulations.

4.3. Application to KDEMO

4.3.1. Predictive simulation workflow for optimization

The optimization of design parameters is essential for the construction of a new reactor. The performance of the plasma that may appear in the designed reactor should be predicted and further optimized to reduce the risk of the research. For the optimization of design parameters, 9 design parameters, R_0, a_0, δ, κ, P_{NB}, R_{tan}/R_0, E_{NB}, I_P, and B_T were selected and two modeling parameters, $f_{GW,ped}$ and $n_{e0}/n_{e,ped}$ were selected as shown in Table 2. The modeling parameters were required because determining the Greenwald density fraction and the density peaking is difficult as theoretical understandings of them are not complete and the predicted values of them are controversial in the future reactor. Hence, the parameters were also considered as design parameters to assess the effects of the parameters by the scan. The minimum and maximum values of the design and modeling parameters were chosen by referring to the previous research [97] and KDEMO conceptual study report.

Parameter	Description	Min.	Ref.	Max.
R_0	Major radius	6.0 m	6.8 m	8.0 m
a_0	Minor radius	1.8 m	2.1 m	3.5 m
δ	Triangularity	0.20	0.60	0.65
κ	Elongation	1.7	2.0	2.1
P_{NB}	NBI heating power	90 MW	105 MW	140 MW
R_{tan}/R_0	NBI tangency radius relative to the major radius	0.6	1.0	1.2
E_{NB}	NBI injection energy	0.5 MeV	0.5 MeV	1.5 MeV
I_P	Plasma current	11.0 MA	15.5 MA	18.0 MA
B_T	Vacuum toroidal magnetic field at R_0	5.0 T	7.4 T	14.0 T
$f_{GW,ped}$	Greenwald fraction of pedestal electron density	0.6	0.9	1.1
$n_{e0}/n_{e,ped}$	Electron density peaking relative to the pedestal density	1.0	1.1	2.5

Table 2. Selected design and modeling parameters for the optimization and their minimum, maximum, and reference [97] values for the simulations.

Figure 45 shows the modeling workflow from the random selection of parameter values to the storage of the simulation outputs. From the minimum and maximum range of given parameters, the simulation input values were chosen and were fed into the simulation with the proper initial condition. The simulation was conducted by the TRIASSIC predictive simulation workflow and the results were systematically stored in an IDS format.

As the modeling workflow should be able to deal with the extreme case of larger, hotter, and denser conditions compared to the previous predictive modeling studies, several models should be noted. For the modeling of the pedestal, the model from the pedestal scaling law by Sugihara was adopted. From the pedestal height calculated from the model, the pedestal temperature is determined by using the prescribed density (by $f_{GW,ped}$). The density and temperatures at the pedestal region were renewed by the (brute) pedestal transport model which slowly modifies the plasma quantities manually. The radiative losses are significant in such hot and dense conditions, and so the Bremsstrahlung and synchrotron radiations were additionally considered. Also, the thermonuclear and beam−driven fusion reactions were considered by the NUBEAM component.

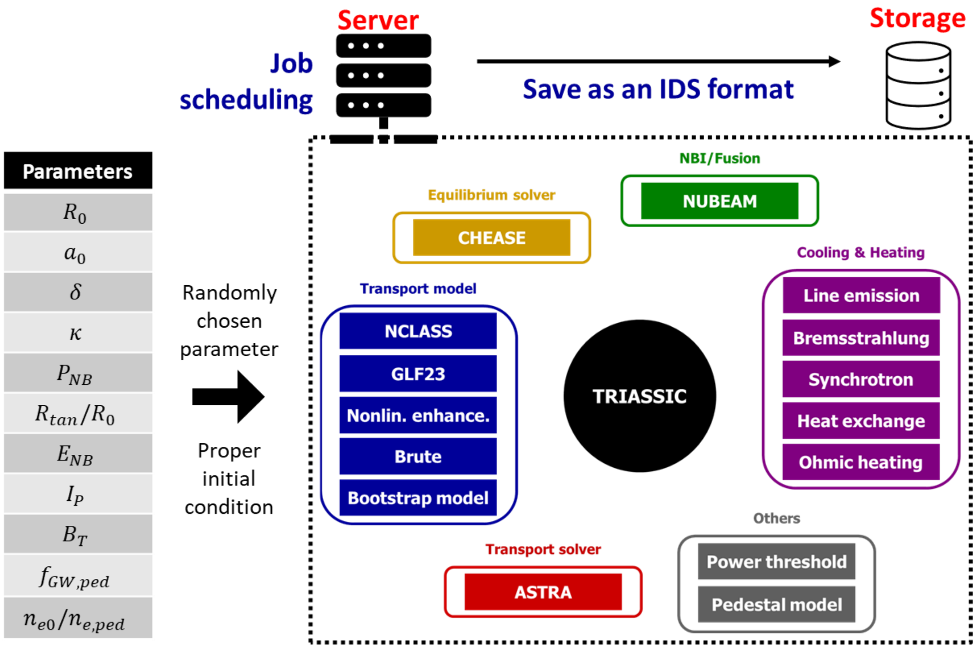


Figure 45. The predictive simulation workflow for the KDEMO design and modeling parameters optimization study. 11 parameters

were randomly chosen and fed into the workflow with proper initial conditions. The simulations were executed through the job sched-uling functionality of TRIASSIC and systematically stored in an IDS format.

By using the predictive simulation workflow, about 400 simula-tions were conducted with randomly chosen parameters to con-struct a set of statistical models. The input parameters for the sta-tistical models were 11 design and modeling parameters, and the target parameters were the performance targets such as non-in-ductive current drive fraction (f_{NI}) and fusion gain (Q_{FUS}). An ex-ample of the prediction of the statistical models and the simulation data is shown as gray dashed lines and black dots, respectively in Figure 46. The minimum and maximum values predicted by the statistical models, which represent the confidence of the statistical model prediction, are shown as the blue shaded regions. The opti-mal design and modeling parameter value and its corresponding performance values which were found by manually modifying the input parameters are shown as red circles.

The prediction of performances based on the parameters from the previous research showed lower performances than the pre-vious research. The reason was due to the lower pedestal height predicted by the workflow. However, the pedestal height predicted by the scaling law was indeed realistic and hence the approach was regarded as a conservative approach to the performance predic-tions.

To overcome the lack of fusion gain without the increase of the plasma current, it was essential to increase the density peaking factor. The density peaking factor is expected to be significant [98] due to the low collisionality of the burning plasmas. The increase of the density peaking factor hindered the neutral beam current drive, and the amount of non−inductive current drive fraction was insufficient for the steady−state operation. For the current drive fraction, the minor radius and its relative values against the major radius (aspect ratio) were the key parameters. The increase of minor radius (decrease of aspect ratio) significantly helped in−crease the bootstrap current drive, and hence it was able to achieve the target parameters at a smaller machine size and a slightly higher toroidal magnetic field. The prediction from the statistical model is shown in terms of the minor radius, a_0, in Figure 46. It can be seen that the increase of minor radius has a tradeoff as it enhances the non−inductive current drive while diminishing the fusion gain. In conclusion, the design parameters were determined by $R_0 = 6.2\,m,\ a_0 = 2.3\,m,\ B_T = 8.0\,T,\ I_P = 15.5\,MA$ but with a higher peaking factor of $n_{e0}/n_{e,ped} = 1.5$. The other parameters were only slightly modified.

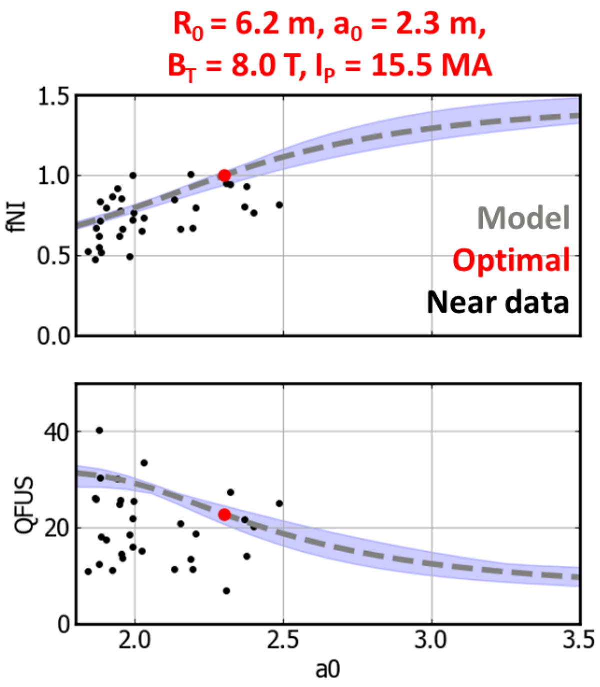


Figure 46. The optimization result of the KDEMO design and mod−eling parameters. The statistical model, optimal point, and predic−tive simulation results that are near to the optimal point are shown

for the optimization targets, f_{NI} and Q_{FUS} as function of the minor radius, a_0. The minimum and maximum values predicted by the statistical model are represented as the blue-shaded regions.

The target constraint used in this study was $f_{NI} > 1.0$ and $Q_{FUS} > 20$. The modification of input parameters was done manually to achieve both targets while decreasing the machine size and estimated cost of facilities such as reducing plasma current and toroidal magnetic field. In the future, when the cost and risk can be provided by the additional model as a function of input parameters, a more sophisticated optimization result would be drawn by the quantitative analysis. Also, when the amount of the non-inductive current drive is significant, the simulation often crashed due to low current density near the magnetic axis. This leads to the lack of highly non-inductive prediction results, as can be seen in Figure 46. It can be also checked that the prediction from the statistical model is not confident enough for the high f_{NI} region. In the future, the workflow and the equilibrium model therein should be revised to stably predict the plasma equilibrium even when the current near the magnetic axis is scarce.

Chapter 5. Summary and Conclusion

5.1. Summary and Conclusion

The integrated modeling approach was required for a compre− hensive understanding of the experiments and to predict upcoming reactors for efficient research & development. For constructing an integrated modeling framework, the modular approach was supe− rior to the non−modular approach due to its interchangeability. In adopting the modular approach to the framework, generic data was required to communicate with the various physics models. The IDS, which is generic, hierarchical, and standard, was a good candidate for the central data storage of the framework. Due to the generic structure, the data nodes in the IDS could be represented by other nodes through definitions and constraints. Also, it was introduced that there are three tiers of code integration with the IDS; the physics code kept data consistency for the tiers 1 and 2 approaches, while it was not for the tier 3 approach.

In chapter 2, the development of the TRIASSIC framework, which is the first Python−based tokamak simulation framework that fully embraces the IDS, was introduced. TRIASSIC was com− pared with other pre−developed frameworks. In the comparison with the IPS framework, TRIASSIC was distinguishable that the IDS is adopted as a native data structure of the framework. On the other hand, when compared with the EU−IM workflows which physic actors therein contain the IDS in their I/O routine, TRIASSIC took the approach of tier 3 code integration. By the approach, the original codes were only slightly modified to be interfaced with the framework. The codes were compiled together with the driver routine and the wrapper generated by F2PY or SWIG, to produce a library that can be utilized in the high-level language framework. This unique code interfacing method adopted in TRIASSIC made it easy on utilizing the legacy physics codes in the framework.

Owing to the simplification of the code interfacing tasks, many physics codes and models could be adopted in the framework in a short period of development. The physics codes and models that are available in the framework were introduced. Also, the graphical user interface which enhanced usability by facilitating input file generation and the parallel computation of message passing interface through the job scheduling system which improved the performance of the simulations were addressed. Lastly, the verifications of TRIASSIC code interfaces and their couplings were shown.

The problems related to the internal data consistency in TRIASSIC and the solution were introduced in chapter 3. In terms of data consistency, three possible implementations of a component and the role of the framework were addressed. EU-IM workflows took both first and second approaches, where the physics actors keep the consistency within an IDS by themselves (owing to tiers 1 and 2 integrations) and also by placing the non-physics actors that keep the consistency between the IDSs in the workflows. The third approach, in which the framework conducts the implicit subsequent updates after the component execution, was adopted in TRIASSIC. The drawback of the approach, the implicitness of the procedure that makes it difficult to notice which constraint/definition the framework imposes, was overcome by the declarative programming approach. Moreover, compared to the typical imperative approach, the framework could cope with various situations encountered by adopting various kinds of models without cumbersome use-case studies.

The input and output nodes, constancy of the nodes, and user inputs were defined for every relation found in some IDSs, to enable a computer code to automatically find a way to update data nodes consistently. Then, the procedure was divided into two parts - adding and applying relations. Relations were added (registered) to the framework during the simulation initialization, and the actual node values were calculated during the simulation by applying the relations. Algorithms for both situations were addressed. With the improvement of the consistency loop, simulations were accelerated from the previous imperative approach. Also, the simulation could flexibly cope with unusual modeling settings, especially in terms of particle transport and plasma flow modeling, that are traditionally impossible for other frameworks.

By keeping the consistency automatically by the framework, unique code interfacing was possible. Moreover, the maintenance was enhanced as the integration of a new code does not demand a physics code developer to deal with every data node in IDSs. In terms of IDS utilization and the simplicity of the code integration, TRIASSIC was superior to the IPS and OMFIT frameworks or EU-IM workflows. Furthermore, the consistency between the IDSs enabled modelers easily compose a workflow only by arranging the code components.

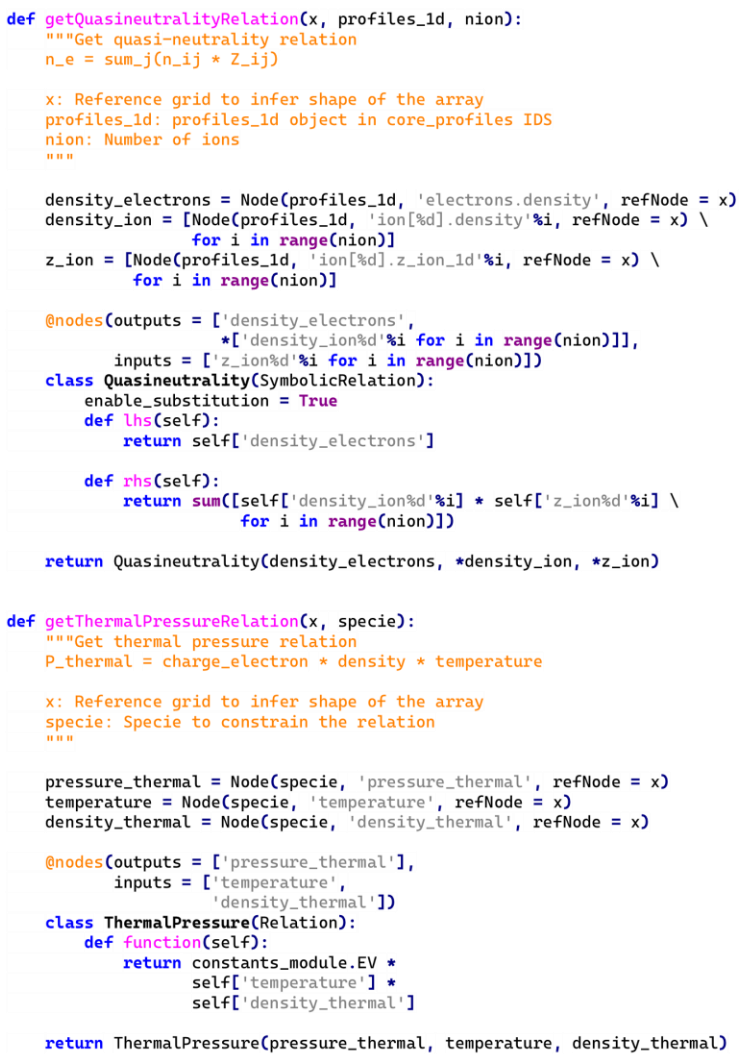
In chapter 4, TRIASSIC was utilized to compose various workflows for applications to various devices. The kinetic equilibrium workflow was composed and validated by comparing the calculated and measured plasma energies in KSTAR. By the stationary-state predictive modeling workflow, the density and temperatures of the KSTAR plasmas at a stationary state could be predicted. The workflow was validated in terms of plasma energy, electron density, and temperature predictions. For the application to VEST, a time-dependent predictive modeling workflow was composed, for valid simulations under a short pulse duration. The comparison of pre-dictions with the experiment, however, showed non-negligible discrepancies and cast the need for future work. The validity of the models introduced in the workflow was discussed, and future work was planned. The application to KDEMO was done by multiple sim-ulations of randomly chosen 11 machine designs and modeling pa-rameters. Owing to the job scheduling and systematic data storage, a predictive simulation database could be made and the data was used to train a statistical model for the design optimization.

Appendix

A. Code Snippet of the Relation Definition

Figure 47 shows two examples of relation definitions as a code snippet. Both relations require a reference array as the first argu− ment to initialize the input/output arrays to the same shape as the grid size. Getting a quasi−neutrality constraint requires the number of ions modeled in the simulation and the *profiles_1d* object to lo− cate the data nodes corresponding to the electron and ion densities with ion charges. Each term in the relation is then abstracted to a class instance named Node. The class *Quasineutrality* inherits *SymbolicRelation*, which requires the definition of the functions *lhs* (left−hand side) and *rhs* (right−hand side) to find the satisfaction condition of the relation and to generate functions from the relation. Inside the *Quasineutrality* relation, the *enable_substitution* flag in− dicates that the relation is substitutable, while it does not in default. The Python decorator, *nodes* designates the input nodes and the output nodes in the order of constancy. The decorator was required for the reusability under a different number of ions.

The *ThermalPressure* relation, on the other hand, inherits *Relation*, which only requires a definition of the function to calculate the only output value, *pressure_thermal*. It is noteworthy that the *ThermalPressure* relation is generally defined and can be applied to any generic species.



```python
def getQuasineutralityRelation(x, profiles_1d, nion):
    """Get quasi-neutrality relation
    n_e = sum_j(n_ij * Z_ij)

    x: Reference grid to infer shape of the array
    profiles_1d: profiles_1d object in core_profiles IDS
    nion: Number of ions
    """

    density_electrons = Node(profiles_1d, 'electrons.density', refNode = x)
    density_ion = [Node(profiles_1d, 'ion[%d].density'%i, refNode = x) \
                   for i in range(nion)]
    z_ion = [Node(profiles_1d, 'ion[%d].z_ion_1d'%i, refNode = x) \
             for i in range(nion)]

    @nodes(outputs = ['density_electrons',
                      *['density_ion%d'%i for i in range(nion)]],
           inputs = ['z_ion%d'%i for i in range(nion)])
    class Quasineutrality(SymbolicRelation):
        enable_substitution = True
        def lhs(self):
            return self['density_electrons']

        def rhs(self):
            return sum([self['density_ion%d'%i] * self['z_ion%d'%i] \
                        for i in range(nion)])

    return Quasineutrality(density_electrons, *density_ion, *z_ion)


def getThermalPressureRelation(x, specie):
    """Get thermal pressure relation
    P_thermal = charge_electron * density * temperature

    x: Reference grid to infer shape of the array
    specie: Specie to constrain the relation
    """

    pressure_thermal = Node(specie, 'pressure_thermal', refNode = x)
    temperature = Node(specie, 'temperature', refNode = x)
    density_thermal = Node(specie, 'density_thermal', refNode = x)

    @nodes(outputs = ['pressure_thermal'],
           inputs = ['temperature',
                     'density_thermal'])
    class ThermalPressure(Relation):
        def function(self):
            return constants_module.EV *
                   self['temperature'] *
                   self['density_thermal']

    return ThermalPressure(pressure_thermal, temperature, density_thermal)
```

Figure 47. A code snippet that describes two relation definitions – the quasi−neutrality constraint (relation 1 in Table 1) and the definition of thermal pressure (relation 5 in Table 1).

Bibliography

[1] WANG, F. et al., Technologies and perspectives for achieving carbon neutrality, Innov. **2** 4 (2021) 100180.

[2] WESSON, J., Tokamaks, 4th ed., Oxford University Press (2011).

[3] WANG, W.X. et al., Nonlinear flow generation by electrostatic turbulence in tokamaks, Phys. Plasmas **17** 7 (2010) 072511.

[4] GRIERSON, B.A. et al., Orchestrating TRANSP Simulations for Interpretative and Predictive Tokamak Modeling with OMFIT, Fusion Sci. Technol. **74** 1–2 (2018) 101.

[5] REN, Y. et al., Exploring the regime of validity of global gyrokinetic simulations with spherical tokamak plasmas, Nucl. Fusion **60** 2 (2020) 026005.

[6] WALTZ, R.E. et al., A gyro−Landau−fluid transport model, Phys. Plasmas **4** 7 (1997) 2482.

[7] STAEBLER, G.M., KINSEY, J.E., WALTZ, R.E., Gyro−Landau fluid equations for trapped and passing particles, Phys. Plasmas **12** 10 (2005) 102508.

[8] STAEBLER, G.M., KINSEY, J.E., WALTZ, R.E., A theory−based transport model with comprehensive physics, Phys. Plasmas **14** 5 (2007) 055909.

[9] BOURDELLE, C. et al., Core turbulent transport in tokamak plasmas: bridging theory and experiment with QuaLiKiz, Plasma Phys. Control. Fusion **58** 1 (2016) 014036.

[10] GARCIA, J. et al., First principles and integrated modelling achievements towards trustful fusion power predictions for JET and ITER, Nucl. Fusion **59** 8 (2019) 086047.

[11] KIM, S.H. et al., CORSICA modelling of ITER hybrid operation scenarios, Nucl. Fusion **56** 12 (2016) 126002.

[12] IMBEAUX, F. et al., A generic data structure for integrated modelling of tokamak physics and subsystems, Comput. Phys. Commun. **181** 6 (2010) 987.

[13] HAWRYLUK, R.J., An Empirical Approach To Tokamak Transport, Physics of Plasmas Close to Thermonuclear Conditions, Vol. 1, (1981) 19–46.

[14] CENACCHI, G., TARONI, A., JETTO: A Free Boundary Plasma Transport Code, JET−IR(88)03 (1988).

[15] PEREVERZEV, G., YUSHMANOV, P.N., ASTRA User Guide,

IPP−Report IPP 5/98 (2002).
[16] NA, Y.S. et al., On benchmarking of simulations of particle transport in ITER, Nucl. Fusion **59** 7 (2019) 076026.
[17] ARTAUD, J.F. et al., The CRONOS suite of codes for integrated tokamak modelling, Nucl. Fusion **50** 4 (2010) 043001.
[18] ROMANELLI, M. et al., JINTRAC: A System of Codes for Integrated Simulation of Tokamak Scenarios, Plasma Fusion Res. **9** (2014) 3403023.
[19] IMBEAUX, F. et al., Design and first applications of the ITER integrated modelling & analysis suite, Nucl. Fusion **55** 12 (2015) 123006.
[20] COSTER, D.P. et al., The European Transport Solver, IEEE Trans. Plasma Sci. **38** 9 (2010) 2085.
[21] STRAND, P. et al., Towards a Predictive Modelling Capacity for DT Plasmas: European Transport Simulator (ETS) Verification and Validation, Preprint: 2018 IAEA Fusion Energy Conf. (Gandhinagar, India, 22–27 October 2018), IAEA−CN−451.
[22] MENEGHINI, O. et al., Integrated modeling applications for tokamak experiments with OMFIT, Nucl. Fusion **55** 8 (2015).
[23] MENEGHINI, O. et al., Neural−network accelerated coupled core−pedestal simulations with self−consistent transport of impurities and compatible with ITER IMAS, Nucl. Fusion **61** 2 (2021) 026006.
[24] SCHNEIDER, M. et al., Simulation of heating and current drive sources for scenarios of the ITER research plan, Nucl. Fusion **61** 12 (2021) 126058.
[25] OMAS: Ordered Multidimensional Array Structures, http://gafusion.github.io/omas.
[26] LEE, C.Y. et al., Development of integrated suite of codes and its validation on KSTAR, Nucl. Fusion **61** 9 (2021) 096020.
[27] SUGIHARA, M., MUKHOVATOV, V., POLEVOI, A., SHIMADA, M., Scaling of H−mode edge pedestal pressure for a Type−I ELM regime in tokamaks, Plasma Phys. Control. Fusion **45** 9 (2003) L55.
[28] SNYDER, P.B., GROEBNER, R.J., LEONARD, A.W., OSBORNE, T.H., WILSON, H.R., Development and validation of a predictive model for the pedestal height, Phys. Plasmas **16** 5 (2009).
[29] MENEGHINI, O. et al., Self−consistent core−pedestal

transport simulations with neural network accelerated models, Nucl. Fusion **57** 8 (2017).

[30] PARAIL, V. et al., Integrated modelling of ITER reference scenarios, Nucl. Fusion **49** 7 (2009) 075030.

[31] KOECHL, F. et al., Modelling of transitions between L− and H−mode in JET high plasma current plasmas and application to ITER scenarios including tungsten behaviour, Nucl. Fusion **57** 8 (2017) 086023.

[32] ELWASIF, W.R. et al., The Design and Implementation of the SWIM Integrated Plasma Simulator, 18th Euromicro Int. Conf. on Parallel, Distributed and Network−Based Processing, (2010) 419–427.

[33] HAWRYLUK, R.J., “AN EMPIRICAL APPROACH TO TOKAMAK TRANSPORT”, Physics of Plasmas Close to Thermonuclear Conditions, Elsevier (1981) 19–46.

[34] BATCHELOR, D. et al., Advances in simulation of wave interactions with extended MHD phenomena, J. Phys. Conf. Ser. **180** (2009) 012054.

[35] LÜTJENS, H., BONDESON, A., SAUTER, O., The CHEASE code for toroidal MHD equilibria, Comput. Phys. Commun. **97** 3 (1996) 219.

[36] HINTON, F. L., HAZELTINE, R.D., Theory of plasma transport in toroidal confinement systems, Rev. Mod. Phys. **48** 2 (1976) 239.

[37] FreeGS: Free boundary Grad−Shafranov solver, https://github.com/bendudson/freegs.

[38] PARK, J.M. et al., Integrated Modeling of Core, Edge Pedestal and Scrape−Off−Layer for High β N Steady−State Scenarios on DIII−D, Preprint: 2018 IAEA Fusion Energy Conf. (Gandhinagar, India, 22−27 October 2018), TH/P7−2.

[39] LUDA, T. et al., Integrated modeling of ASDEX Upgrade plasmas combining core, pedestal and scrape−off layer physics, Nucl. Fusion **60** 3 (2020) 036023.

[40] RAFIQ, T., KRITZ, A.H., WEILAND, J., PANKIN, A.Y., LUO, L., Physics basis of Multi−Mode anomalous transport module, Phys. Plasmas **20** 3 (2013) 032506.

[41] LUO, L., RAFIQ, T., KRITZ, A.H., Improved Multi−Mode anomalous transport module for tokamak plasmas, Comput. Phys. Commun. **184** 10 (2013) 2267.

[42] KINSEY, J.E. et al., Predictions of the near edge transport shortfall in DIII−D L−mode plasmas using the trapped gyro−

Landau−fluid model, Phys. Plasmas **22** 1 (2015) 012507.
[43] FABLE, E. 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 (2013) 124028.
[44] SCOTT, B., E×B shear flows and electromagnetic gyrofluid turbulence, Phys. Plasmas **7** 5 (2000) 1845.
[45] ERBA, M., CHERUBINI, A., PARAIL, V. V., SPRINGMANN, E., TARONI, A., Development of a non−local model for tokamak heat transport in L−mode, H−mode and transient regimes, Plasma Phys. Control. Fusion **39** 2 (1997) 261.
[46] ERBA, M., ANIEL, T., BASIUK, V., BECOULET, A., LITAUDON, X., Validation of a new mixed bohm/gyro−Bohm model for electron and ion heat transport against the iter, tore supra and start database discharges, Nucl. Fusion **38** 7 (1998) 1013.
[47] CITRIN, J. et al., Tractable flux−driven temperature, density, and rotation profile evolution with the quasilinear gyrokinetic transport model QuaLiKiz, Plasma Phys. Control. Fusion **59** 12 (2017) 124005.
[48] VAN DE PLASSCHE, K.L. et al., Fast modeling of turbulent transport in fusion plasmas using neural networks, Phys. Plasmas **27** 2 (2020).
[49] HOULBERG, W.A., SHAING, K.C., HIRSHMAN, S.P., ZARNSTORFF, M.C., Bootstrap current and neoclassical transport in tokamaks of arbitrary collisionality and aspect ratio, Phys. Plasmas **4** 9 (1997) 3230.
[50] BELLI, E.A., CANDY, J., Kinetic calculation of neoclassical transport including self−consistent electron and impurity dynamics, Plasma Phys. Control. Fusion **50** 9 (2008) 095010.
[51] SAUTER, O., ANGIONI, C., LIN−LIU, Y.R., Neoclassical conductivity and bootstrap current formulas for general axisymmetric equilibria and arbitrary collisionality regime, Phys. Plasmas **6** 7 (1999) 2834.
[52] HAGER, R., CHANG, C.S., Gyrokinetic neoclassical study of the bootstrap current in the tokamak edge pedestal with fully non−linear Coulomb collisions, Phys. Plasmas **23** 4 (2016).
[53] REDL, A., ANGIONI, C., BELLI, E., SAUTER, O., A new set of analytical formulae for the computation of the bootstrap current and the neoclassical conductivity in tokamaks, Phys. Plasmas **28** 2 (2021) 022502.

[54] PANKIN, A., MCCUNE, D., ANDRE, R., BATEMAN, G., KRITZ, A., The tokamak Monte Carlo fast ion module NUBEAM in the national transport code collaboration library, Comput. Phys. Commun. **159** 3 (2004) 157.
[55] KRITZ A. H., HSUAN H., GOLDFINGER, R.C., BATCHELOR, D.B., Ray Tracing Study of Electron Cyclotron Heating in Toroidal Geometry, Proc., 3rd Int. Symp. on Heating in Toroidal Plasmas ECE, Brussels, Belgium (1982) 707–723.
[56] POST, D.E., "Alpha−Particle Heating", Applied Atomic Collision Physics, Vol. 2, (1984) 381–394.
[57] ESTRADA−MILA, C., CANDY, J., WALTZ, R.E., Turbulent transport of alpha particles in reactor plasmas, Phys. Plasmas **13** 11 (2006) 112303.
[58] POST, D.E., JENSEN, R. V., TARTER, C.B., GRASBERGER, W.H., LOKKE, W.A., Steady−state radiative cooling rates for low−density, high−temperature plasmas, At. Data Nucl. Data Tables **20** 5 (1977) 397.
[59] SUMMERS, H.P., The Atomic Data and Analysis Structure, AIP Conference Proceedings, Vol. 543, AIP (2000) 304–312.
[60] SUMMERS, H.P., O'MULLANE, M.G., Atomic Data and Modelling for Fusion: The ADAS Project, (2011) 179–187.
[61] HUBA, J.D., NRL Plasma Formulary, Naval Research Laboratory, Washington, DC (2018).
[62] ALBAJAR, F., JOHNER, J., GRANATA, G., Improved calculation of synchrotron radiation losses in realistic tokamak plasmas, Nucl. Fusion **41** 6 (2001) 665.
[63] TAMOR, S., ANTIC: A code for calculation of neutral transport in cylindrical plasmas, J. Comput. Phys. **40** 1 (1981) 104.
[64] MANDREKAS, J., GTNEUT: A code for the calculation of neutral particle transport in plasmas based on the Transmission and Escape Probability method, Comput. Phys. Commun. **161** 1–2 (2004) 36.
[65] HSL, A Collection of Fortran Codes for Large Scale Scientific Computation., http://www.hsl.rl.ac.uk.
[66] KIM, Y., YOO, M.G., KIM, S.H., NA, Y.S., Development of vector following mesh generator for analysis of two−dimensional tokamak plasma transport, Comput. Phys. Commun. **186** (2015) 31.
[67] SHEWCHUK, J.R., "Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator", (1996) 203–222.

[68] KOECHL, F. et al., Optimising the ITER 15MA DT Baseline Scenario by Exploiting Self−Consistent−Consistent Free−Boundary Core−Edge−Sol Workflow in IMAS, Preprint: 2018 IAEA Fusion Energy Conf. (Gandhinagar, India, 22−27 October 2018), (2018) P7−25.
[69] MARTIN, Y.R., TAKIZUKA, T., Power requirement for accessing the H−mode in ITER, J. Phys. Conf. Ser. **123** (2008).
[70] SEO, J. et al., Parametric study of linear stability of toroidal Alfvén eigenmode in JET and KSTAR, Nucl. Fusion **60** 6 (2020) 066008.
[71] PETERSON, P., F2PY: A tool for connecting Fortran and Python programs, Int. J. Comput. Sci. Eng. **4** 4 (2009) 296.
[72] BEAZLEY, D.M., Automated scientific software scripting with SWIG, Futur. Gener. Comput. Syst. **19** 5 SPEC (2003) 599.
[73] SAUTER, O., MEDVEDEV, S.Y., Tokamak coordinate conventions:, Comput. Phys. Commun. **184** 2 (2013) 293.
[74] IVANOV, A.A., KHAYRUTDINOV, R.R., MEDVEDEV, S.Y., POSHEKHONOV, Y.Y., New adaptive grid plasma evolution code SPIDER, 32nd EPS Conf. Plasma Phys. 2005, EPS 2005, Held with 8th Int. Work. Fast Ignition Fusion Targets − Europhys. Conf. Abstr. **3** July (2005) 2146.
[75] LAO, L.L. et al., MHD equilibrium reconstruction in the DIII−D tokamak, Fusion Sci. Technol. **48** 2 (2005) 968.
[76] EUROfusion Integrated Modelling workflow, https://wpcd−workflows.github.io/ets.html#non−physics−actors.
[77] QU, Z.S., FITZGERALD, M., HOLE, M.J., Analysing the impact of anisotropy pressure on tokamak equilibria, Plasma Phys. Control. Fusion **56** 7 (2014).
[78] MEURER, A. et al., SymPy: symbolic computing in Python, PeerJ Comput. Sci. **3** (2017) e103.
[79] THE SYMPY DEVELOPERS, SymEngine: A Fast Symbolic Manipulation Library Written in C++, (2016).
[80] KALLENBACH, A. et al., Impurity seeding for tokamak power exhaust: From present devices via ITER to DEMO, Plasma Phys. Control. Fusion **55** 12 (2013).
[81] YOON, S.W. et al., Characteristics of the first H−mode discharges in KSTAR, Nucl. Fusion **51** 11 (2011) 113009.
[82] NA, Y.S. et al., On hybrid scenarios in KSTAR, Nucl. Fusion **60** 8 (2020) 086006.
[83] SARWAR, S., NA, H.K., PARK, J.M., Effective ion charge

(Zeff) measurements and impurity behavior in KSTAR, Rev. Sci. Instrum. **89** 4 (2018) 043504.
[84] YOON, S.W. et al., The Effect of Electron Cyclotron Heating on Thermal and Fast−Ions Transport in High Beta−Poloidal Discharges at KSTAR, Preprint: 2018 IAEA Fusion Energy Conf.(Gandhinagar, India, 22−−27 October 2018), (2018) EX/P7−5.
[85] KIM, J. et al., Initial measurements of fast ion loss in KSTAR, Rev. Sci. Instrum. **83** 10 (2012) 10D305.
[86] STAEBLER, G.M., HOWARD, N.T., CANDY, J., HOLLAND, C., A model of the saturation of coupled electron and ion scale gyrokinetic turbulence, Nucl. Fusion **57** 6 (2017) 066046.
[87] PAN, C. et al., Investigation of energy transport in DIII−D High−βP EAST−demonstration discharges with the TGLF turbulent and NEO neoclassical transport models, Nucl. Fusion **57** 3 (2017) 036018.
[88] BATEMAN, G., KRITZ, A.H., KINSEY, J.E., REDD, A.J., WEILAND, J., Predicting temperature and density profiles in tokamaks, Phys. Plasmas **5** 5 PART 1 (1998) 1793.
[89] CHUNG, J. et al., Formation of the internal transport barrier in KSTAR, Nucl. Fusion **58** 1 (2018) 016019.
[90] LEE, S.G., LEE, H.H., KO, W.H., YOO, J.W., Validation of toroidal rotation and ion temperature in KSTAR plasmas, Fusion Sci. Technol. **69** 2 (2016) 555.
[91] CHUNG, K.J. et al., Design Features and Commissioning of the Versatile Experiment Spherical Torus (VEST) at Seoul National University, Plasma Sci. Technol. **15** 3 (2013) 244.
[92] KIM, S. et al., Development of diverted plasma discharge and plan for advanced divertor study in VEST, Fusion Eng. Des. **123** (2017) 584.
[93] YANG, J., KIM, Y., JEONG, W.I., HWANG, Y.S., Simple and accurate method of diamagnetic flux measurement in Versatile Experimental Spherical Torus (VEST), Rev. Sci. Instrum. **89** 10 (2018) 103508.
[94] LEE, C.−Y. et al., Investigation of the effect of pre−fill gas in VEST discharges by predictive transport simulations, J. Korean Phys. Soc. **81** 2 (2022) 126.
[95] KIM, S., JANG, J.Y., KIM, Y., HWANG, Y.S., Acceleration of ion rotation during internal reconnection events in the versatile experiment spherical torus (VEST), Nucl. Fusion **61** 12 (2021) 126011.

[96] KIM, Y. et al., Design of combined system of helium charge exchange spectroscopy and hydrogen beam emission spectroscopy in VEST, Fusion Eng. Des. **123** (2017) 975.

[97] KANG, J.S. et al., Development of a systematic, self-consistent algorithm for the K-DEMO steady-state operation scenario, Nucl. Fusion **57** 12 (2017) 126034.

[98] WEISEN, H. et al., Scaling of density peaking in JET H-modes and implications for ITER, Plasma Phys. Control. Fusion **48** 5A (2006) A457.

Abstract in Korean

본 연구에서는 TRIASSIC (tokamak reactor integrated automated suite for simulation and computation) 코드의 자세한 디자인과 실행 결과에 대해 소개합니다. 이 시뮬레이션 코드는 기존에 존재하던 플라즈마 평형, 1.5차원 및 2차원 플라즈마 수송, 신고전 및 난류 수송 모델, 전류 구동 및 가열 (냉각) 모델, 그리고 2차원 격자 생성기 등의 코드를 구성하여 만들어졌습니다. 프레임워크 내 데이터 구조로써 일반 데이터 구조를 채택함으로써 TRIASSIC의 코드 구성요소들은 완전한 모듈화 방식으로 결합될 수 있었습니다. 일반 데이터 구조에 의존하지 않는 독특한 코드 결합 방식으로 인해, 더 이상 유지보수되지 않는 레거시 코드들 또한 쉽게 결합될 수 있었습니다. 본 코드의 그래피컬 유저 인터페이스, 프레임워크와 코드 구성 요소들의 병렬 컴퓨팅에 관한 내용도 다뤄집니다. 평형, 수송, 그리고 가열 측면에서의 TRIASSIC 시뮬레이션의 검증 내용도 소개됩니다. 시뮬레이션 프레임워크 내 일반 데이터 구조의 데이터 모델과 데이터 정의를 만족시키기 위해, 데이터를 관리하는 프레임워크의 중심부에는 선언적 프로그래밍이 도입되었습니다. 선언적 프로그래밍을 통해 일반 데이터의 데이터 노드 간 관계식을 만족시킴으로써 데이터 간 내부 일관성을 확보하고, 코드의 유연성과 명시성을 추가적으로 확보할 수 있었습니다. TRIASSIC은 해석적, 예측적 모델링 측면에서 KSTAR 플라즈마를 대상으로 검증되었습니다. VEST 장치를 대상으로 한 예측 및 이에 대한 검증 내용 또한 서술됩니다. 경제적인 핵융합 실증로 건설을 목표로 KDEMO 장치에 대한 적용 및 장치 설계 최적화 연구도 소개됩니다.

주제어: 토카막, 플라즈마, 핵융합, 통합 모델링, 통합 시뮬레이션
학번: 2016-21305