Extension of the Visualization Capabilities within the European Integrated Tokamak Modelling Platform

Leon Kos
University of Ljubljana, Mech. Eng., Aškerčeva 6
SI-1000 Ljubljana, Slovenia
leon.kos@lecad.fs.uni-lj.si

Hans-Joachim Klingshirn[2], Pablo Luis García Müller[3], Frederic Imbeaux[4] and EFDA ITM-TF contributors
[2]Max-Planck-Institut für Plasmaphysik, EURATOM Association, Garching, Germany, hmk@ipp.mpg.de
[3]Centro de Investigaciones Energéticas Medioambientales y Tecnológicas, Madrid, Spain, PabloLuis.Garcia@ciemat.es
[4]CEA, IRFM, F-13108 Saint-Paul lez Durance, France, frederic.imbeaux@cea.fr

ABSTRACT

In an effort to provide visualizations for integrated fusion simulations for the European Integrated Tokamak Modeling Task Force (ITM-TF), up to now several approaches have been used. The visualization capabilities of the ITM-TF platform provide tools to developers for (automated) creation of specialized plots for Kepler workflows and sophisticated interactive data analysis tools for end users. Custom visualizations for specific physics problems/workflows are made available through the ITMVis Python visualization library. This approach structures access to Consistent Physical Objects (CPOs), data processing and plot creation. A structured repository for code fragments that perform the processing simplifies reuse of common and specialized data processing algorithms for various target visualization tools. We demonstrate this for use with native Matplotlib plots as well as embedded within VisIt and Kepler, where specialized plugins/actors have been developed for accessing CPOs in the ITM-TF database.

1 INTRODUCTION

Interfacing fusion codes within the European Integrated Tokamak Modeling (ITM-TF) project is based on Consistent Physical Objects [1] (CPOs). CPOs are standardized data structures that describe various physical aspects of fusion experiments and are designed to be suitable for use with simulation codes and experimental data. Integration with CPOs thus brings a common data model to simulations that allows to directly compare results with experiment, use experimental data as input or mixed approaches. Contents of the CPOs have been carefully crafted and collected from previous best practices in a way that the description is general enough to cover a broad range of tokamak modeling and is easy to use by simulation codes. It was anticipated that the CPO structures would improve/change during the development of the ITM-TF. To facilitate such changes and to support different programming languages, the data structure is described by a XML schema definition (XSD). Rigorous validations, data bindings and translations are possible based on this XSD definition. A basic usage of the data structure description is persistent storage in a database. As typically done for experimental fusion data stored in scientific databases (using formats like MDSplus (MDS+) or HDF5 [2, 3]), ITM-TF databases are organized using selectors like *shot* and *run* numbers. The underlying MDS+ and HDF format is then prescribed by the CPO structure definition. Access to these ITM-TF databases is performed using a library called *Universal Access Layer (UAL)* [4], which provides a set of routines for "middle-ware" database access tailored especially for CPO transfer. The UAL is "universal" in a way that different languages use the same acccess routines and that data transfer is simplified without compromising efficiency when using coupled codes in a "scientific workflow" that couples different codes in a compatible way and sequence. To summarize, the CPO data structures combined with the UAL provides a common backbone for various tasks within the ITM modeling effort.

Graphical inspection of data in variety of ways becomes more difficult with increasing data complexity. As time is a key physical quantity of tokamak modeling, this quantity is included in nearly every CPO and can be assumed as an additional dimension when time-varying properties are inspected. Usually, time-dependent visualization is done by specifying a time point or selecting a cycle (frame or time-slice) from the UAL database. From that point data can be retrieved and visualized with different tools in a variety of ways. Visualization tools capable of accessing UAL data directly (without exporting or converting) were developed to provide required visualizations. Three approaches are used by the ITM-TF for producing visualization:

Table 1: ITM-TF visualization tools capable of UAL direct access

ITM-TF Tool	Interactive use (user level)	Non-interactive processing	Input data pre-processing	Publication ready figures	Kepler [5] workflow integration
Python	Yes (expert)	yes	yes	yes - matplotlib	yes - Python actor
VisIt	Yes (basic)	yes	limited	yes - raster gr.	yes - VKA actor
Matlab	Yes (expert)	yes	yes	yes - builtin	no
ISE	Yes (basic)	no	no	no	yes - control only

(i) programming languages, (ii) general tools, and (iii) specialized tools with capabilities as shown in Table 1. Scripting languages like Python and Matlab are suited for all kinds of data processing. However, their use requires development effort. Non-scripting languages (C++, Java, Fortran) for producing visualizations are even less attractive. They are used for application programming and within visualization tools. In fact, there are no general purpose tools available that can cover all aspects of usage. The Integrated Simulation Editor (ISE) was designed with Kepler [6] workflow control in mind and currently only provides simple 2D signal editing and visualization. For other purposes the VisIt software [7] (open-source, maintained by LLNL) was selected as a general visualization tool. VisIt is popular and extensible visualization software that provides various representation modes and data post-processing in interactive and non-interactive (batch or scripting) mode. Coupling VisIt to the UAL is supported by a custom database plugin *ual_reader*.

One goal of the ISIP sub-project within the ITM-TF is to consolidate and extend the visualization capabilities of the ITM platform, providing developers with tools for (automated) creation of specialized plots that can be easily distributed to end users and tools for creating plots for code/workflow development and debugging. Eventually, users will benefit from easy-to use, interactive tools for data exploration, easily available custom visualizations for specific physics problems/workflows and tools that are capable of producing publication-ready plots. Towards this goal we propose a structured repository of code fragments for visualization-specific data processing, enabling reuse of these operations for various target visualization tools (backends). The following sections describe how this idea is implemented in the ITM-TF visualization tools.

2 ITMVIS PYTHON VISUALIZATION LIBRARY

On the ITM software platform, Python with the matplotlib [8] library can be used to create 2D plots. VisIt combined with the *ual_reader* plugin provides many "standard" 2D and 3D visualizations. During the ITM-TF project Python emerged as scripting language of choice for many tasks, especially for data processing and visualization. There are many examples in ITM [9] sub-projects (IMPs) for custom task-specific plots. In fact, there will always be the need for some kind of "specialized plots" that cannot be covered by "standard" tools. Having this in mind, the ITM-TF aims to provide possibilities for creation of complex problem-specific visualizations as easy as possible for the developer that will (i) make his/her visualization applicable to the ITM data model (ii) make the visualizations readily accessible to the users, and (iii) make it possible to integrate the visualizations in other tools. Observing visualization of various teams in the IMPs that developed a number of custom visualization scripts with Python, generally the following steps are followed:

1. Get CPO(s) from the database:
   - Parameters for UAL database access (run, shot, time, user, tokamak, CPO) name are hardcoded, passed as command line options or obtained through user interaction.
   - CPOs are fetched with the Python UAL interface.
2. Data processing:
   - Compute data required for plot, possibly combining data from various CPOs.
   - Typically uses SciPy/NumPy functionality.
3. Create plot:
   - Transform data to format required for plotting library (typically matplotib)
   - Set up figure (axes, labels, units), plot data.

In order to facilitate use of the above observed procedure, we propose the creation of the ITM Python visualization library (ITMVis) which unifies the existing approaches, provides extensibility and enables use of custom visualizations in different plotting tools. The ITMVis library encompasses the following:

- A standard frontend (e.g. scripts) covering step 1, so that all visualizations are available through a uniform user interface.
- A structured repository for code fragments that perform the processing in step 2.
- A collection of "standard" processing functions which can then be shared among multiple visualization scripts or other software.
- Collects and makes code owned/developed by specific projects available to everybody in a central place.

The proposed approach was first implemented as a set of Python routines that provide a command line interface (CLI) for the plot selection. In addition to CLI, it is easy to use the ITMVis library in other Python projects. Routines for accessing the UAL database (Step 1) and output

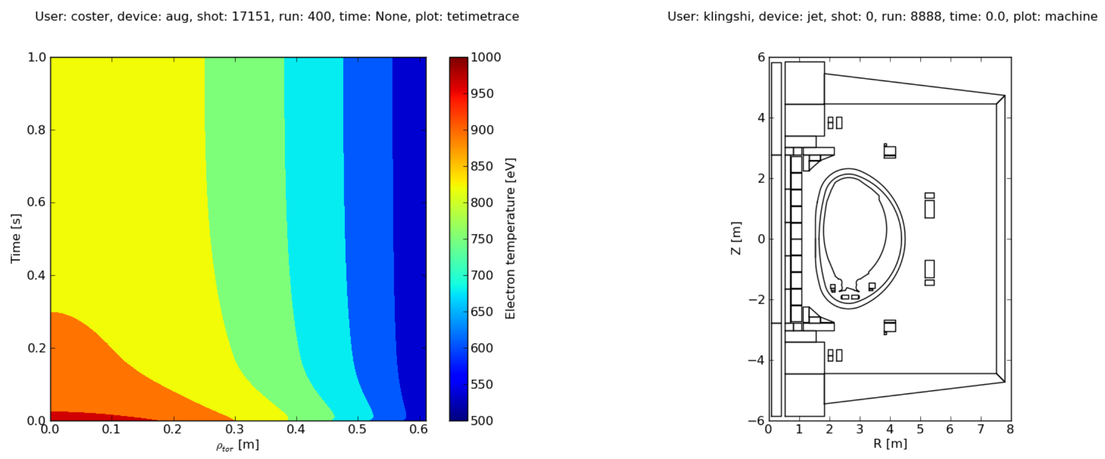

Figure 1: Electron temperature timetrace from ASDEX Upgrade coreprof CPO (left) and JET machine description (right) produced with matplotlib backend.

(Step 3) were developed and are maintained by ITM ISIP-TF. Developers/users need to concentrate only in data processing (Step 2) and plot description by providing *metadata* and *plot data functions*. Metadata describes the plot and verifies data availability so the user is informed whether a specialized plot can be produced or not. Only if a database contains the required data, the *plot data* function can be executed. To detect modules containing metadata and plot data functions in the Python search path, some function naming conventions are prescribed.

Listing 1: Metadata and plotdata function in Python from the ITMvis library. Prepares a plot of the average charge of all charge states of the first impurity species as a radial profile.

```python
def itmvis_coreimpur_average_charge_state_1_metadata(coreimpur):
    if field_filled(coreimpur.cpo.rho_tor) \
        and field_filled(coreimpur.cpo.nz) \
        and field_filled(coreimpur.cpo.z):

        md = dict()
        md["type"] = ITMVIS_PLOTTYPE_XY
        md["axes"] = ("$\\rho_{tor}$␣[m]", "Charge")
        md["legend"] = "Impurity␣#1:␣average␣charge"
        return md
    else:
        return None

def itmvis_coreimpur_average_charge_state_1_plotdata(coreimpur):
    data = {}
    data["x"] = coreimpur.cpo.rho_tor
    data["y"] = ((coreimpur.cpo.nz * \
        coreimpur.cpo.z).sum(axis = 2) \
        / coreimpur.cpo.nz.sum(axis=2))[:,0]
    return data
```

Listing 1 is an example of such functions. While the metadata function is a generic 2D-plot description, data processing in the plot data function shows complex averaging of density of plasma impurities over the available species and with subsequent application of a logarithmic scale to the sum. One can observe that there is no time specified. Proper CPO slice(s) specified as a function argument are provided by the ITMVis library. The plot can then be produced using different “back-ends” chosen by the user. Fig. 1 shows two “non-standard” examples produced with matplotlib [8].

3 ITMVIS VISUALIZATION WITH VISIT

VisIt [7] as a general visualization tool can be used as a back-end for the ITMvis library. While VisIt itself includes Python for scripting and data processing, this processing is limited to data that has already been read into VisIt by a database reader, and can therefore be considered as data post-processing. To provide custom visualizations through ITMVis, another Python interpreter must be embedded in the *ual_reader* plugin. Figure 2 shows the scheme for

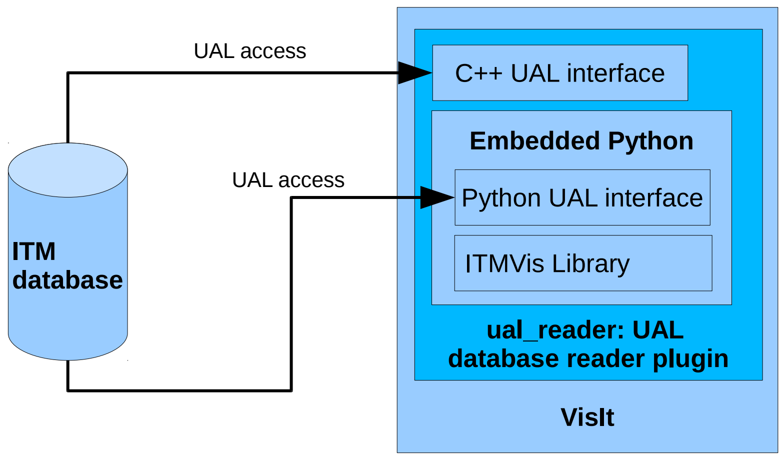

Figure 2: VisIt UAL database reader plugin provides "standard" visualizations with C++ interface and "custom" visualizations through *embeded* Python interface.

the proposed Python embedding. Data exchange between Python and C++ is minimized by separating the UAL database access. Similarly to the separation of the plot description as shown in Listing 1, a VisIt database plugin [10] requires the following functions to be provided by the ITMVis library:

PopulateDatabaseMetaData(*md); ITMVis collects the plots from modules according to the search path and naming conventions. If the plot is applicable, it is shown as available in the VisIt graphical user interface (GUI) plot selector. For this, ITMVis returns metadata as required by the C++ plugin. Plot name, type, topology, data validity, labels and units are populated at this stage and returned to VisIt as a structure (*md) of all available plots (including standard and ITMVis plots, see below).

GetMesh(meshname, time); ual_reader calls ITMvis with a request to provide a mesh specified with the *meshname* and *time*. In order to minimize complexity when passing data between Python and C++, ITMVis is responsible for building VTK [11] structures as required by VisIt.

GetVar(varname, time); For plots that in addition to the mesh also require data, ual_reader calls this routine which returns a VTK array of data that VisIt then maps on the mesh.

One complication when embedding ITMVis in the VisIt plugin is proper memory management of the created objects. Careful referencing and dereferencing of Python and VTK objects is required to account for the different *garbage collection* mechanisms. Embedding the Python

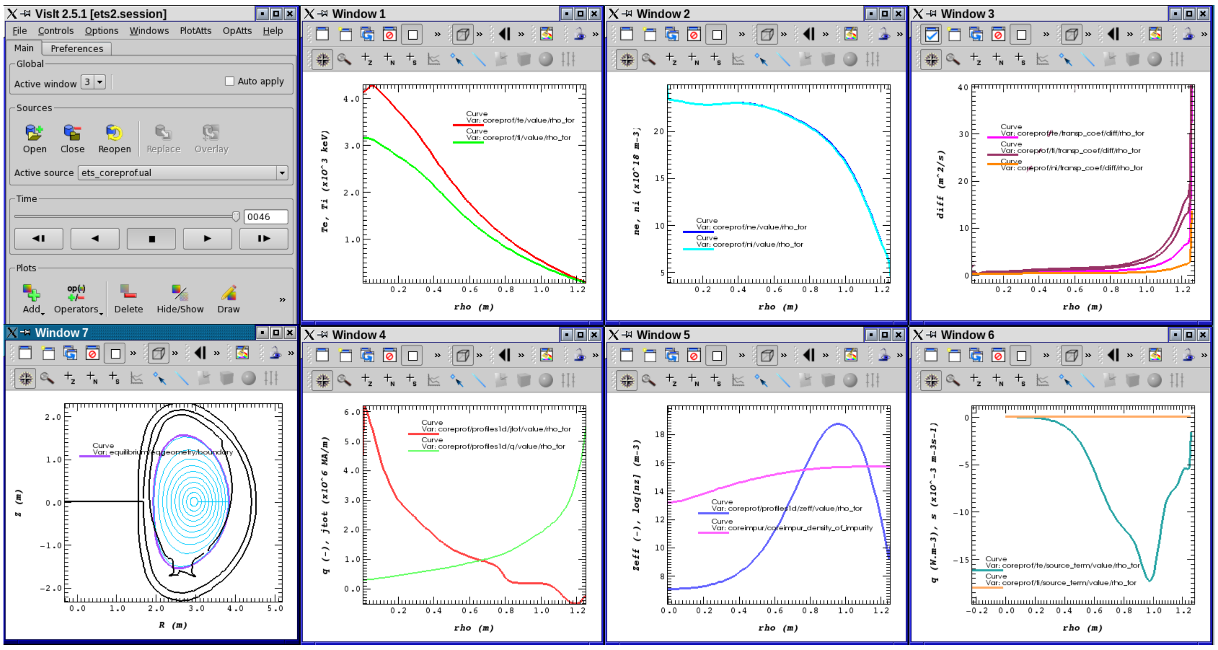
VisIt 2.5.1 [ets2.session]
File Controls Options Windows PlotAtts OpAtts Help
Main Preferences
Active window 3
Auto apply
Open Close Reopen Replace Overlay
Active source ets_coreprof.ual
0046
Add Operators Delete Hide/Show Draw
Window 1
Window 2
Window 3
Window 4
Window 5
Window 6
Window 7
Var: coreprof/te/value/rho_tor
Var: coreprof/ti/value/rho_tor
Var: coreprof/ne/value/rho_tor
Var: coreprof/ni/value/rho_tor
Var: coreprof/te/transp_coef/diff/rho_tor
Var: coreprof/ti/transp_coef/diff/rho_tor
Var: coreprof/ni/transp_coef/diff/rho_tor
Var: equilibrium/eqgeometry/boundary
Var: coreprof/profiles1d/jtot/value/rho_tor
Var: coreprof/profiles1d/q/value/rho_tor
Var: coreprof/profiles1d/zeff/value/rho_tor
Var: coreimpur/coreimpur_density_of_impurity
Var: coreprof/te/source_term/value/rho_tor
Var: coreprof/ti/source_term/value/rho_tor
rho (m)
R (m)
Z (m)
Te, Ti (x10^3 keV)
ne, ni (x10^18 m-3)
diff (m^2/s)
q (-), jtot (x10^6 MA/m)
Zeff (-), log[nz] (m-3)
q (W.m-3), s (x10^-3 m-3s-1)

Figure 3: European Transport Solver (ETS) convergence visualization under VisIt.

ITMVis library into the VisIt UAL plugin also enables use of the the *ITM general grid description(GGD) and grid service library(GSL)* which is currently only available for Python and Fortran. The GGD was developed in the ITM for describing complex meshes, which due to C++ bindings unavailability could not be handled easily by the ual_reader plugin directly.

Standard plots can now be complemented by custom plots from ITMVis. An example of such use is shown in a screen-shot (fig. 3) of various plots required to follow convergence of the *European Transport Solver* [12]. Window 5 contains an ITMVis plot produced with the *custom* routines as shown in Listing 1 together with the *standard* plot of the effective charge profile z_{eff} in the plasma core as obtained when solving the transport model. Several CPOs were accessed simultaneously to produce a complete window arrangement that can be stored in a session file for reuse on similar data.

4 EXTENSION OF THE STANDARD ITM VISIT VISUALIZATIONS

So-called “standard” visualizations provided by the UAL VisIt plugin (ual_reader) are generated directly from the CPO data structure description with an XSL transformation. This means that C++ code for each plot is generated from XSL code templates. The existing (hard to maintain) XSLT prototype implementation was rewritten for the sake of clarity and with added EXSLT user functions providing a basis for further representations/plot types. Describing “standard” plot types is handled with *representation tags* directly in the XSD description of the CPOs. Representation tags describe “natural” plots of data of interest and are related to conventions and a physicist way of thinking. For example, one needs to decide which field represents x and y-axis for a *1D curve plot*. XML representation tags consists of name, variable type, meshtype and corresponding axis links. The representation name is arbitrary and is shown as the last name in VisIt path when adding plots. The attribute describing the variable type can be scalar or vector and describes the kind of data mapped on mesh represented by one or more links. There are several meshtypes (curve0, axis1D0, rectlinear0, curvilinear0, ...), describing

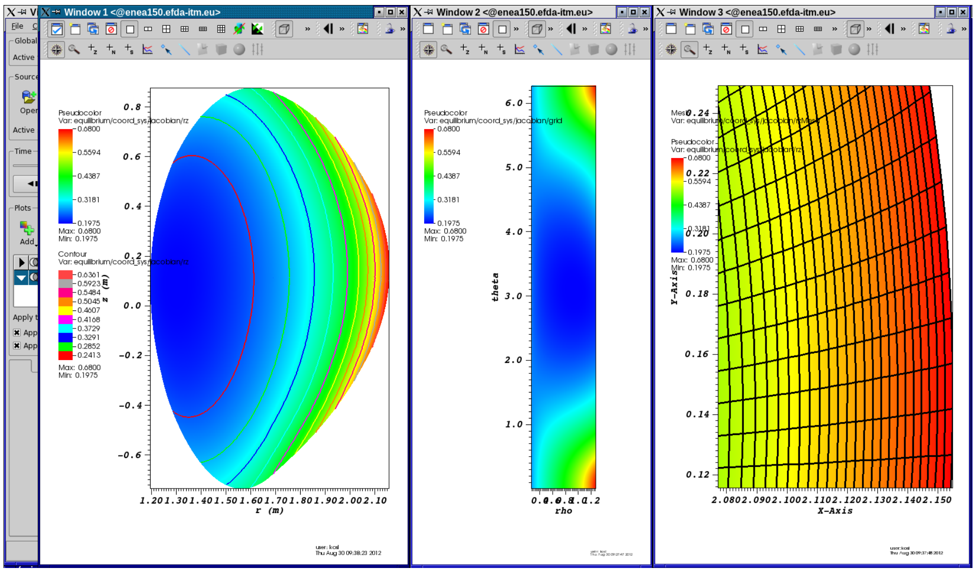
Window 1 <@enea150.efda-itm.eu>
Window 2 <@enea150.efda-itm.eu>
Window 3 <@enea150.efda-itm.eu>
Pseudocolor
Var: equilibrium/coord_sys/jacobian/rz
Var: equilibrium/coord_sys/jacobian/grid
Contour
r (m)
z (m)
theta
rho
X-Axis
Y-Axis

Figure 4: Equilibrium rectilinear mesh in Window 2 (center) is mapped onto curvilinear (r,z) grid shown in Window 1 (left) as pseudocolor plot overlayed with contour plot. Detail of r,z-mesh with Jacobian is shown in Window 3 (right).

the type of variable mapping in space. Each meshtype can have a variable number of links (link1, link2 and/or link3) and thus different kinds of visualization that are constructed as VTK objects. Links provide axis mapping to corresponding vectors or matrices that constitute the mesh. As the ITM project evolved, more representation tags were added. Some representations like time dependence plots of scalar values and curve0 are implemented without the need of representation tags. 3D timed data can be visualized by slicing in VisIt [13]. The power of XSLT code generation resulted in over 300000 lines of C++ generated code. We have developed missing visualization types as shown in Fig. 4 where additional representations and mesh relations (rectilinear, curvilinear) are used for the same data. *Standard* representations were amended with ITMVis custom plots to round up missing links. However, there are still some open issues like representation completion, database correlation, multiple CPO handling and UAL database selection.

5 CONCLUSION

The latest enhancements of ITM-ISIP VisIt-related tools concentrate on the ual_reader UAL database reader plugin to provide easy plotting of simple and complex data to end users as a set of predefined plots available through VisIt GUI. VisIt can be used as an UAL data exploration tool, as all meaningful plots can be made readily available and can be saved for reuse in Kepler workflows and for stand-alone use. As with the current visualization tags there are no possibilities for cross-CPO representations, such requests are now handled by embedding the ITMVis library. However, ITMvis misses the “standard” plots available through the UAL VisIt plugin, which users would like to have available through ITMVis in Python as well. A solution for this could be an XSLT transformation for Python. With these standard representations also available in ITMVIs, the C++ part of plugin might become obsolete. However, performance penalties of this approach first have to be explored further.

ACKNOWLEDGMENTS

This work was supported by the European Commission under: (i) Contracts of Associations between MHEST, CEA , IPP and The European Atomic Energy Community (EURATOM), and (ii) Ministry of Higher Education, Science and Technology of the Republic of Slovenia No. FU06-CT-2007-00065. It was carried out within the framework of the European Fusion Development Agreement, under the EFDA Task Force on Integrated Tokamak Modeling. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

REFERENCES

[1] Frederic Imbeaux, J. B. Lister, G. T. A. Huysmans, W. Zwingmann, M. Airaj, L. Appel, V. Basiuk, David Coster, Lars-Goran Eriksson, Bernard Guillerminet, Denis Kalupin, C. Konz, Gabriele Manduchi, M. Ottaviani, G. Pereverzev, Y. Peysson, O. Sauter, J. Signoret, and Par Strand, "A generic data structure for integrated modelling of tokamak physics and subsystems," Computer Physics Communications **181**, 987–998 (2010).

[2] "MDSplus – data acquisition and storage tools," http://www.mdsplus.org (2012).

[3] "HDF – hierarchical data format," http://www.hdfgroup.org/ (2012).

[4] G. Manduchi, F. Iannone, F. Imbeaux, G. Huysmans, J.B. Lister, B. Guillerminet, P. Strand, L.-G. Eriksson, and M. Romanelli, "A universal access layer for the integrated Tokamak Modelling Task Force," *6th IAEA Technical Meeting on Control, Data Acquisition, and Remote Participation for Fusion Research*, Fusion Engineering and Design **83**, 462–466 (2008).

[5] Matthieu Haefele, Leon Kos, Pierre Navaro, and Eric Sonnendrücker, "Euforia integrated visualization," in *PDP 2010 - The 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing* (Pisa, Italy, 2010) pp. 498–502.

[6] Ilkay Altintas, Chad Berkley, Efrat Jaeger, Matthew Jones, Bertram Ludascher, and Steve Mock, "Kepler: An extensible system for design and execution of scientific workflows," in *SSDBM '04: Proceedings of the 16th International Conference on Scientific and Statistical Database Management* (IEEE Computer Society, Washington, DC, USA, 2004) p. 423, ISBN 0-7695-2146-0.

[7] "VisIt official homepage," http://visit.llnl.gov/ (2012).

[8] "matplotlib – a Python 2D plotting library," http://matplotlib.sourceforge.net/ (2012).

[9] "Integrated Tokamak Modelling website," http://www.efda-itm.eu/ (2012).

[10] Brad Whitlock, *Getting Data Into VisIt*, LLNL, Livermore, CA, 2nd ed. (July 2010), `https://wci.llnl.gov/codes/visit/manuals.html`.

[11] "Visualization toolkit," http://www.vtk.org/ (2012).

[12] D.P. Coster, V. Basiuk, G. Pereverzev, D. Kalupin, R. Zagórksi, R. Stankiewicz, P. Huynh, and F. Imbeaux, "The european transport solver," Plasma Science, IEEE Transactions on **38**, 2085 –2092 (sept. 2010), ISSN 0093-3813.

[13] Leon Kos, Olivier Hoenen, Simon Kulovec, Philippe Huynh, Jožef Duhovnik, and Frederic Imbeaux, "Towards 3D time dependent visualization within ITM-TF infrastructure," in *20th Int. Conf. Nuclear Energy for New Europe* (Bovec, Slovenia, 2011) pp. 1008.1–1008.8.