

Unified Approach to Visualizations within the European Integrated Tokamak Modelling Framework

Leon Kos

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

**Hans-Joachim Klingshirn², Pablo Luis García Müller³, Frederic Imbeaux⁴
and EFDA ITM-TF contributors**

²Max-Planck-Institut für Plasmaphysik, EURATOM Association, Garching, Germany,
hmk@ipp.mpg.de

³Centro de Investigaciones Energéticas Medioambientales y Tecnológicas, Madrid, Spain,
PabloLuis.Garcia@ciemat.es

⁴CEA, IRFM, F-13108 Saint-Paul lez Durance, France, frederic.imbeaux@cea.fr

ABSTRACT

Diverse visualization approaches are used for integrated fusion simulations in the scope of the European Integrated Tokamak Modelling Task Force (ITM-TF). In an effort to provide diversity and at the same time universality of visualizations for different backends, the ITMVis library provides a common description of custom visualizations contributed by users. Beside dedicated visualizations many "standard" plots are used and are described within the ITM-TF data structure via an XSD schema that contains plot representation tags. From the representation tags one can generate standard plots for a specific computer language and visualization tool. Such an approach is used for the VisIt visualization tool, where a C++ code for all possible plots in the database is generated with an XSLT transform. Given that the same translation is needed by other tools in different languages to provide standard visualizations, an intermediate, easily interpretable plot description using XML was introduced. With this unified approach standard and custom plots are available for different backends such as VisIt and matplotlib.

1 INTRODUCTION

The European Integrated Tokamak Modelling Task Force [1] (ITM-TF) is developing a framework to enable coupling of physics codes in order to allow flexible modeling, simulation (both interpretative and predictive), verification and validation for fusion research. The approach taken by the ITM-TF is to couple codes via well-defined data structures that consistently describe different physical aspects covered in simulations. These Consistent Physical Objects [2] (CPOs) are used for data storage and interchange between codes in the ITM simulation framework. The collection of CPO descriptions form together a complete data model for diverse simulations that can also contain imported experimental data. This approach enables a direct comparison of results with the experiment and/or use experimental data as an input. The implementation phases and continuous development of the ITM-TF framework are simplified by the fact that the CPO data structures are described by an XML schema definition (XSD), which can be easily modified. This semi human-readable description allows rigorous validation, creation of data bindings and translations for different purposes. The XSD

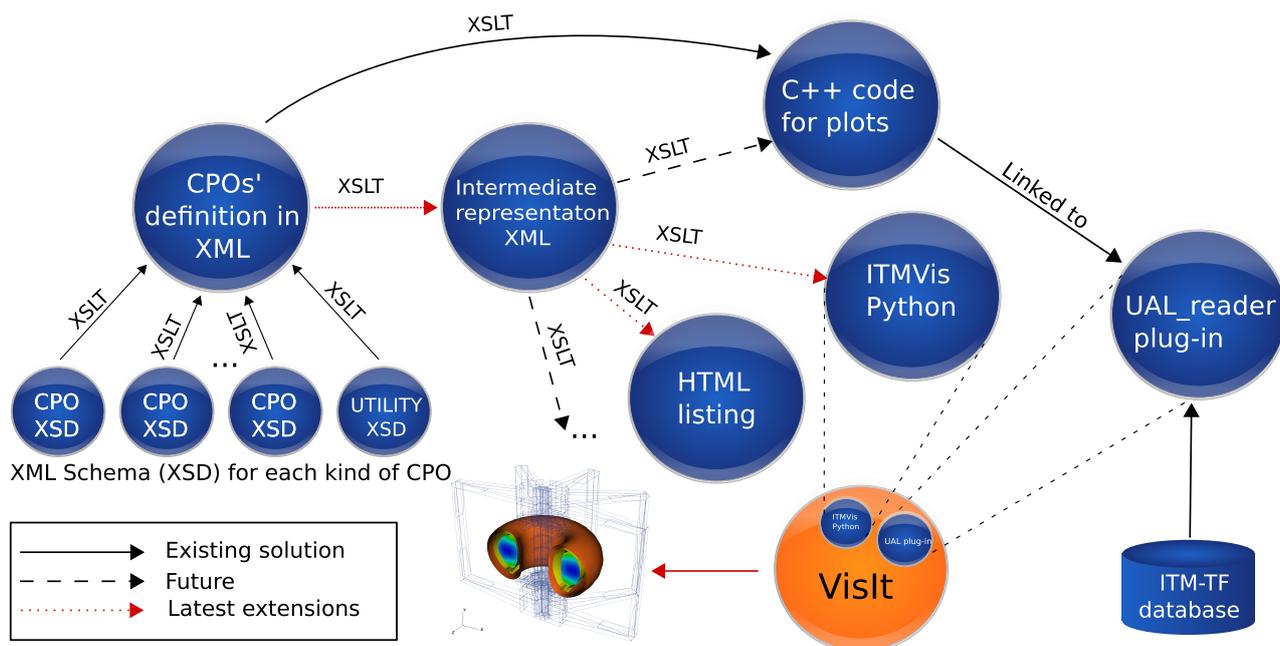


Figure 1: XSLT flowchart for standard representation transformations used for visualization.

data-structure description is mainly used to generate CPO definitions in an XML format obtained by applying the XSL translation [3] (XSLT) language. Consistency of CPO definitions is assured by the XSD description, so that the derived CPO definitions in XML are consistent too. Additionally, the XSDs allow the use of custom *complexType* data-structures that are reused in different CPOs. The resulting *CPODef.xml* database description thus contains expanded and complete information about the ITM-TF data model. From this starting point all other code can be generated through XSLT processing. Figure 1 on the left shows this initial transformation from many CPO definitions into single CPOs' definition in XML. The fundamental purpose of *CPODef.xml* is a description of the ITM-TF persistent storage database, where actual data is stored. Storage can also be in-memory only (i.e. cached) for fast data exchange between coupled codes [4]. *CPODef.xml* is used for the ITM-TF database creation and for describing access to data fields in the database by a variety of programming languages. The multi-language library that interfaces access to CPOs in the ITM-TF database is called Universal Access Layer [5] (UAL) and includes definitions of CPOs in each supported programming language. The included definitions are then used in physics codes for data access with a small set of the UAL subroutines responsible for retrieving and storing CPOs during the ITM simulations. The UAL thus interfaces codes with storage in the hierarchical formats [6, 7], which are commonly used in high performance computing (HPC) and the fusion community. The UAL extends these technologies by allowing cached parallel and distributed access to data between the ITM-TF users. The ITM-TF adapted existing simulation codes to use the UAL and developed new codes [8] that will allow detailed simulations of present and future device scenarios.

As most of the CPOs are time dependent, the ITM-TF database stores *slices* of each CPO during simulation iterations or experiment sampling. Slicing occurs at different time scales for each CPO depending on the physics involved [9, see Fig. 12]. To support the diagnostic of the time-dependent data in CPOs, several visualization tools are used depending on the scales and *representation* of the data that is regularly used in simulations. For many simulations, custom visualization are created using general-purpose graphics libraries. To avoid such custom approaches and to provide *standardized* sets of visualizations, the ITM-TF infrastructure support team (ISIP) aims to provide tools that

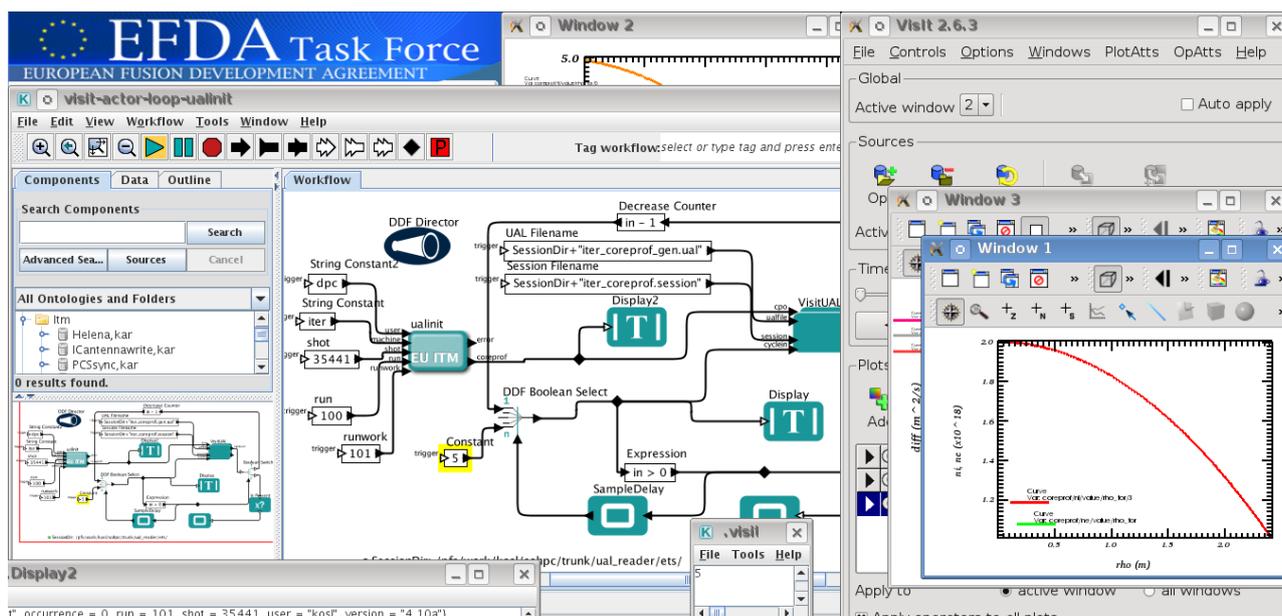


Figure 2: The VisIt Kepler Actor – VisItUAL in a workflow within a loop reading data through the Universal Access Layer.

can be used for visualizations without the need for scientists to manually program each *plot*. The ITM-TF selected the Kepler [10] scientific workflow engine which enables graphical programming of workflows. Data flow between processes (called *actors*) is modelled by wires. Through wires only small amount of data (signals) are passed, usually just single numbers, strings and structures with CPO names, which identify database entries. The Actual data exchange between ITM actors encapsulating fusion codes is then implemented by direct access to the “real” data through the UAL library.

2 ITM-TF VISUALIZATION TOOLS

The ITM aims to provide complex visualizations for physics codes that are adapted to the ITM framework. To provide scientists with general visualization tools in the ITM framework, the EUFORIA/JRA4 [11] project has developed a plugin for the VisIt [12] 3D visualization software that allows reading CPOs from the ITM-TF database directly. For inclusion of VisIt in the Kepler workflows the VisIt Kepler Actor (VKA) was developed. VKA has been recently upgraded to allow visualizations of multiple CPOs and now includes its own graphical user interface (GUI) written in Java. VKA provides seamless inclusion of VisIt into scientific workflows with UAL data access. Figure 2 shows use of the VKA named *VisItUAL* in a workflow that iterates in a loop through all time slices of a database retrieved through the UAL. VisIt is launched from the workflow itself and gets *fired* at each loop iteration to show the data for the *cycle* specified as an input to *VisItUAL*. VisIt has a possibility to save a *session* describing plots that are saved in a *session file* for later reuse for similar visualizations where only the input database changes. Before starting the iterations, the actor *ualinit* copies the requested input CPOs into a local *runwork* from the distributed ITM-TF database. For this, the “tuple” *user*, *machine*, *shot*, and *run* identifying the database has to be specified. Copying and specifying the input CPOs provides *data provenance*. This means that the source is well known and not destroyed by modifications. Workflows prepared in this way can be easily shared and reproduced. *Provenance* is a feature of scientific workflows that is often overlooked, omitting it brings the risk that results

cannot be easily verified.

Other ITM/ISIP visualization tools are mainly based on the Python language, where *matplotlib* [13] is used for creating 2D *plots*. JRA4 developed the Python Kepler Actor [11] that in contrast to the Java based *Jython* Kepler actor can import *matplotlib* for creating custom non-blocking plots in Kepler workflows. Another approach developed by the ITM to provide *standardized* plots is the *ITMVis* [14] visualization library that aims to provide several output *backends* for custom plots scripted in Python. The *ITMVis* code snippets can be shared between users or put into a collective database of custom visualizations. Many of such snippets were translated from other languages like IDL, which is still used in many codes for visualization.

These visualization tools can be used within workflows or standalone. Most visualization tools are diagnostic and reporting tools used standalone and offline. If data is available during the simulation, they can also be used for monitoring of simulations. VisIt as a standalone tool needs an input file (with *.ual* extension), containing *tuples* pointing to database entries. The capability of VisIt to allow code instrumentation through its *SIMlib* was used to couple UAL and VisIt through the *ualconnector* [1, see ITM Grid Service Library] CPO server capable of serving complex unstructured meshes, which are used e.g. in the 2D edge transport code [15] (SOLPS). VisIt as a general 3D scientific visualization tool can be used for many purposes as it builds on top of the Visualization ToolKit [16] (VTK) that has an established visualization pipeline. Another useful visualization principle used in VisIt is data callback separation that a plugin needs to provide for visualizations.

3 STANDARD REPRESENTATIONS FOR VISUALIZATION

The *ITMVis* library tries to follow the VisIt data description by splitting visualizations of CPOs into meta-data and plot data. This separation is a natural choice for all tools that want to prepare a list of possible visualizations depending on data availability. It should be noted that not all CPOs are filled with data when running a particular simulations. The *ITMVis* library presently concentrates on *custom plots*, while the *UAL_reader* plugin works with *standard representations*. A representation is, in fact, a description of the visualization. There can be multiple representations of the same data. The *UAL_reader* plugin for VisIt (see Fig. 1) utilizes *representation tags* that are included in the XSD files describing the CPOs. This means that ITM-TF data structure “knows” how it can be visualized. This is similar to the concept of *views* in SQL, which are stored in a database directly (except that these would be stored directly in the database). Representation tags are just a description that does not enlarge storage but facilitates views on data.

Listing 1: A representation tag provides name, type and links for a visualisation. The tag is added as an `xs:annotation/xs:appinfo` XSD element as shown in the following lines marked with a plus sign.

```
<xs:element name="imp" type="vecflt_type">
  <xs:annotation>
    <xs:documentation>Implicit source term [ $s^{-1.m^{-3}}$ ]. Time-dependent.
      Vector (nrho)</xs:documentation>
  + <xs:appinfo>representation name=rho_tor; var=scalar; meshtype=curve0;
  + link1=/rho_tor;</xs:appinfo>
  + <xs:appinfo>representation name=rho_tor_norm; var=scalar; meshtype=curve0;
  + link1=/rho_tor_norm; </xs:appinfo>
  </xs:annotation>
</xs:element>
```

An example of updated XSD with two representation tags is shown in Listing 1. The representation name is arbitrary and is shown as the last name in the VisIt path when adding plots. The attribute var

can be either a scalar or a vector and describes the kind of data mapped on the mesh represented by one or more links. Several meshtypes exist:

curve0 Curve as a polylines plot.

axis1D0 X axis given as a link1 vs. a bunch of curves from a matrix.

mesh1D0 Contours (closed curves) as a (2D) surface mesh.

rectlinear0 or curvilinear0 Meshes in 2D and 3D.

Meshtype describes the type of variable mapping in space. Each meshtype can have a variable number of links (link1, link2 and/or link3) and thus different kind of visualizations that are constructed as VTK objects in VisIt. Links provide axis mapping to corresponding vectors or matrices that constitute a mesh. The topology name at the end of the meshtype means (0D - points, 1D - lines, 2D - surface, 3D - volume). The number at the end distinguishes variations of the same meshtype (0, 1, ...). If no topology is given, a general description is assumed (2D and 3D).

Depending on the type of data mapped on the mesh, given links are expanded to the proper hierarchical position within the CPO (called a *path*). When a `xs:complexType` is represented, then links are expanded with a local path of the CPO during generation of *CPOdef.xml* from the XSD files. For example `<xs:complexType name="rz0D">` with `link1=r; link2=z` gets expanded to `link1=eqgeometry/geom_axis/r link2=eqgeometry/geom_axis/z` and thus adding the (missing) relative path from CPO to the link. In other words, complexTypes are relocatable as they are reused in many representations without absolute reference to fields in the same CPO. When an element specifies complexType (e.g. `<xs:element name="position" type="rz2D">`), then the representation in complexType is used and not in the element itself. Simple elements (scalars, vectors, matrices) can be directly linked within a CPO by specifying the absolute path to them. There are also compound elements that are reused (expanded) as a whole (e.g. `<xs:element name="coord_sys">`) in many CPOs. To distinguish between them, one has to describe links with absolute or relative paths. This is done with with leading `"/` for absolute references and any other as relative to a compound or *complexType* that is reused in many CPOs.

3.1 Intermediate representation layer

Standard representations are currently used for data visualization within some CPOs, where the structure of the data fields allows this. For visualizations using data from multiple CPOs, the *ITMvis* approach provides scripting/post-processing capabilities that can combine results and output them through different backends. VisIt combined with the *UAL_reader* plugin allows both representations (standard and custom) by embedding a Python interpreter in the plugin itself. Standard representations were not included in *ITMVis* so far, although this was possible by direct interpretation of *CPOdef.xml* or by Python code generation as in *UAL_reader* (where C++ code representing plots is generated from *CPOdef.xml* directly). The process of XSL translation shown in Fig. 1 with solid lines could be repeated for ITMVis too. From the experience gained in developing templates for C++ code used in *UAL_reader*, where 220000+ source lines are generated, we came to a conclusion that introducing an *intermediate XML description* that extracts representation data from the *CPOdef.xml* will reduce the complexity of the XSLT process when applying it to several programming languages. This intermediate description is still in XML, except that it contains the standard representations described more naturally in a custom XML schema that is easily converted further with XSLT. The complexity of the XSLT code is thus reduced and distributed among XSL translations to and from *intermediate.xml*. Such an approach unifies standard CPO representations and can be extended with additional fields in the intermediate XML format for easier translation as required.

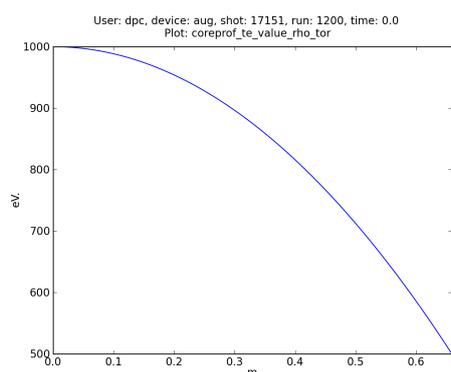
4 RESULTS

The primary goal of the proposed visualization unification shown in Fig. 1 flowchart was to bring standard representations to the *ITMVis* library via an intermediate XML description. To facilitate this, the intermediate XML description was initially generated by adapting existing templates taken from *UAL_reader*. As the output is XML, the translation process is similar to *CPODef.xml* in terms of XML attribute and element specification. Additional complexity occurs where representation links XPath [3] are used to derive linked information. The red dotted lines in Fig. 1 show the implemented XSL Translations. The simplest and most useful translation is generating documentation for standard representations. Instead of using command line XSLT tools, one can translate them into HTML via an XSL translator that is now built into all modern web browsers. All what is needed in the

CPO name	Description
antennas	<ul style="list-style-type: none"> • Meshtype: point0 • Path: antennas/antenna_ec(loop2)/position • Documentation: Launching position in the global reference system; Time-dependent
	<ul style="list-style-type: none"> • Meshtype: mesh1D0 • Path: antennas/antenna_ic(loop2)/setup/straps(loop4)/coord_strap • Documentation: Coordinates (R,z) of polygon describing the antenna in the poloidal plane; rz1d vector (ncoord_strap)
	<ul style="list-style-type: none"> • Meshtype: curve0 • Path: antennas/antenna_ic(loop2)/setup/straps(loop4)/coord_strap • Documentation: Coordinates (R,z) of polygon describing the antenna in the poloidal plane; rz1d vector (ncoord_strap)

Figure 3: List of standard CPO representations included within the ITM-TF database produced by browser-builtin XSL translation.

intermediate XML description is to specify the XSL stylesheet in the XML header and open the file in a browser. XSL and XML are then translated on the fly as shown in Fig. 3. The HTML output lists all representation tags within each CPO that are listed on the top of the page. Standard representations were generated by XSLT from intermediate XML directly into Python scripts that *ITMVis* can easily include into its library of available plots. Python is sensitive to source code indentation so the XSL templates were required to carefully generate the code. As the CPOs are hierarchically described with structures that are often generalized into arrays of structures one needs to specify required indexes of the structure. Required indexes are described in representation *path* by parentheses (loop#) in Fig. 3. It should be noted that probing for data availability in CPOs is required for VisIt that builds *metadata* information of available (valid) “plots”. To prepare valid metadata one needs to scan CPO by “looping” through all available arrays of structures for which data might be available. Such “scanning” can take great deal of time, so it is advisable to know in advance available data ranges in arrays of structures. Figure 4 shows an example *matplotlib* plot with the corresponding generated code snippet that was used within *ITMVis* for the plot. Metadata is returned only if plot is valid (when data for the plot is available). Metadata provides plot description, axes names and units. *Plotdata* prepares required data for plot creation. For field data this means also preparing mesh information on which values are mapped. Further processing to produce visualization is backend (VisIt, matplotlib) dependent. Currently 38 CPOs are supported with 639 standard representations,



```

def itmvis_coreprof_te_value_rho_tor_metadata(coreprof):
    '''Signal value; Time-dependent; Vector (nrho)'''
    if field_filled(coreprof.cpo.te.value) \
        and field_filled(coreprof.cpo.rho_tor):
        md = dict()
        md["type"] = ITMVIS_PLOTTYPE_XY
        md["axes"] = ("Toroidal flux coordinate (not
            normalised, equivalent to rho_tor_norm) [m]", "
            Signal value")
        md["axes_units"] = ("m", "eV.")
        md["axes"] = md["axes_units"]
        md["legend"] = ["Signal value"]
        return md
    else:
        return None

def itmvis_coreprof_te_value_rho_tor_plotdata(coreprof):
    data = {}
    data["x"] = coreprof.cpo.rho_tor
    data["y"] = coreprof.cpo.te.value
    return data

```

Figure 4: The coreprof CPO electron temperature ITMVis plot generated from standard representations via intermediate.xml and *ITMVis* Python code generated by XSLT used to create the plot.

not including time-dependent representations that can be produced by expanding the plot along the time-axis. For example, curves over time can be represented as rectilinear meshes.

5 CONCLUSION

Splitting the implementation of standard visualizations into two parts resulted in an intermediate layer that can be easily used for further translation into target libraries used for visualization. Including information about standard visualizations in the data structure descriptions allows to efficiently propagate information on how to represent the data to various visualization tools. The use of XSL translation assures correct resulting code during, even when changing the XSD schema files defining the data structure. This is an advantage over custom visualizations as e.g. implemented in *ITMVis*, which can require updates when the schemas are changed. Standard and custom visualizations are now both possible within the *ITMVis* and *VisIt* approach. The introduction of an intermediate description layer simplifies the XSL templates and reduces template replication required for the target languages. This unified approach to visualizations in the *ITM-TF* framework now leads to the availability of diverse visualization tools supporting both standard and custom representations.

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. First author was partially supported by the PRACE 3IP project funded in part by the EU's 7th Framework Programme under grant agreement no. RI-211528 and FP7-261557 while mentoring Summer of HPC students Evguenia Usoskina and Luna Backes Drault that worked on the subjects related to this article.

REFERENCES

- [1] “EFDA Integrated Tokamak Modelling Task Force website,” <http://www.efda-itm.eu/> (2013).
- [2] 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).
- [3] Ivan Herman, “Overview of XSLT and XPath,” http://www.w3.org/Consortium/Offices/Presentations/XSLT_XPATH (2006).
- [4] A. Galonska, P. Gibbon, F. Imbeaux, Y. Frauel, B. Guillerminet, G. Manduchi, F. Wolf, and ITM-TF contributors, “Parallel universal access layer: A scalable I/O library for integrated tokamak modeling,” *Computer Physics Communications* **184**, 638–646 (3 2013).
- [5] 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).
- [6] “HDF – hierarchical data format,” <http://www.hdfgroup.org/> (2013).
- [7] “MDSplus – data acquisition and storage tools,” <http://www.mdsplus.org> (2013).
- [8] G. L. Falchetto, D. Coster, R. Coelho, et al. ITM-TF Contributors, and JET-EFDA Contributors, “The European Integrated Tokamak Modelling (ITM) effort: Achievements and first physics results,” in *Proceedings of the 24th IAEA Fusion Energy Conference* (San Diego, CA, 2012) pp. TH/P2–25, http://www-naweb.iaea.org/napc/physics/FEC/FEC2012/papers/439_THP225.pdf.
- [9] D.P. Coster, V. Basiuk, G. Pereverzev, D. Kalupin, R. Zagoórksi, R. Stankiewicz, P. Huynh, F. Imbeaux, and ITM-TF contributors, “The european transport solver,” *Plasma Science, IEEE Transactions on* **38**, 2085 –2092 (sept. 2010), ISSN 0093-3813.
- [10] 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.
- [11] Matthieu Haeefele, 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.
- [12] “VisIt official homepage,” <http://visit.llnl.gov/> (2013).
- [13] “matplotlib – a Python 2D plotting library,” <http://matplotlib.sourceforge.net/> (2013).
- [14] Leon Kos, Hans-Joachim Klingshirn, Pablo Luis García Müller, Frederic Imbeaux, and EFDA ITM-TF contributors, “Extension of the visualization capabilities within the european integrated tokamak modelling platform,” in *21st Int. Conf. Nuclear Energy for New Europe* (Ljubljana, Slovenia, 2012) pp. 1117.1–1117.8, <http://lecad.si/~leon/research/1117-final.pdf>.
- [15] R. Schneider, X. Bonnin, K. Borrass, D. P. Coster, H. Kastelewicz, D. Reiter, V. A. Rozhansky, and B. J. Braams, “Plasma edge physics with B2-Eirene,” *Contributions to Plasma Physics* **46**, 3–191 (2006), ISSN 1521-3986, <http://dx.doi.org/10.1002/ctpp.200610001>.
- [16] “Visualization toolkit,” <http://www.vtk.org/> (2013).