ITM Visualization Tools – 2012 program

04/16/12

This document covers the status and development plans of visualization tools in the ITM.

Table of Contents

Available tools as of February 2012

Generic tools

- **Python** (+SciPy/NumPy/Matplotlib libraries)
  - very powerful for data post-processing and creating visualization scripts for specific tasks
  - interactive visualization (for expert users)
  - users: development by advanced users/experts, usage by everybody
  - workflow integration: yes (Python actor)
  - can use general grid description (Python grid service library)
  - Issues:
    - ~~Should have recent versions of Python, Numpy, Matplotlib on Gateway (→ adressed by software upgrade beginning 2012)~~
- **VisIt**
  - GUI for interactive creation of visualizations/data exploration
  - also for scripted visualization (saved sessions, VisIt Python scripting)
  - capabilities: complex plots, some basic data processing, scripting
  - access to UAL through UAL database reader plugin (UAL plugin)

- users: everybody, requires very little training, good documentation for GUI
- workflow integration: yes (VisIt actor)
- can use general grid description (ualconnector/Python GSL)
- Issues:
  - ~~UAL plugin not available for data version 4.09a, not available in central installation~~
  - UAL plugin has no support for selecting user, tokamak, UAL backend (? check!)
  - ~~currently not straightforward to use (user has to create special input file)~~
  - general grid description currently not available through UAL database reader, only available through ualconnector (→ to be merged in UAL database reader, see 7)

- **Matlab**
  - capabilities similar to Python, but better IDE/GUI
  - users: development by experienced matlab users, usage: everybody
  - Issues
    - Requires licenses (only limited number available on GW), vendor lock-in
    - No support for general grid description
- **Integrated Simulation Editor (ISE)**
  - basic visualisation of 1d data
  - users: requires basic ISE training
  - Issues:
    - only basic plots of 1d quantities
    - currently not available for 4.09a (→ addressed in ISIP 2012 timeline)

Overview table – generic tools

	Capabilities					Workflow integration	Availability	Training material	
	Interactive	Non-interactive	Data processing	Publication-ready	General grid description				
Python	Yes (expert)	yes	yes	yes	yes	Yes	unlimited	TODO	
VisIt	Yes (basic)	yes	limited	yes	yes	Yes	unlimited	TODO	
Matlab	Yes (expert)	yes	yes	yes	no	No	limited licenses	TODO	
ISE	Yes (basic)	no	no	no	no	Yes	unlimited	TODO	

Table explanations:

- **Capabilities**
  - Interactive: supports interactive plot generation/data exploration
    - *basic* means no or very limited advanced knowledge/training required
    - *expert* means advanced knowledge/training required
  - Non-interactive: supports automatic generation of plots
  - Data processing: supports post-processing of data
  - Publication-ready: allows sufficient control over plot details and quality to achieve results suitable for publications
  - General grid description: can visualize CPOs that use the general grid description.
- **Workflow integration**: can be used to visualize data within Kepler workflows
- **Availability**: restrictions for operation on Gateway
- **Training material:** availability of training material

Training: some material is available from the 2012 March ITM Training session in Garching.

Task specific visualization tools

Inventory ongoing, see separate document.

ITM Visualization roadmap

Goal

To consolidate and extend the visualization capabilities of the ITM platform. The platform should provide:

- **for developers**:
  - tools for (automated) creation of specialized plots for
    - workflow integration
    - sophisticated data analysis

    that are to be distributed to end users.
  - tools for creating plots for code/workflow development & debugging
- **for users:**
  - easy-to use, interactive tools for data exploration
  - easily available custom visualizations for specific physics problems/workflows
  - tools that are capable of producing publication-ready plots
  -

An issue which is currently not addressed is the visualization of large data sets through parallel & hardware accelerated visualization. This should be put on the agenda once the parallel version of the UAL has progressed.

Milestones

February/March 2012

- Inventory of available visualization tools (coordinated with IMPs)
  – Ongoing, moved to April
- Start reorganizing visualization documentation website
  – Ongoing, moved to April/May (Kudowa code camp)
- ~~Visualization training at Garching CC (CPT)~~ – Done.
- ~~Consolidate existing visualization tools:~~
  - ~~VisIt database reader:~~
    - ~~update to data version 4.09a (L. Kos?)~~ - Done.
    - ~~provide central installation for multiple data versions (preferably 4.08b and 4.09a)~~ - Done. Still needs more work, though (4.10a, documentation → see May/June)
  - ~~Python:~~
    - ~~verify updated Python and numpy/matplotlib installation on GW~~ – Done.

April 2012

- Inventory of available visualization tools (coordinated with IMPs)
  - Ongoing. Collected some feedback on tools & wanted standard visualizations in Visit.

May/June 2012 (including Kudowa code camp, 29.05.-8.6.)

- Reorganization of visualization documentation website
- Develop structure for shared Python visualization library based on example scripts from IMP3 (ETS diagnotics scripts)
- Develop prototype for integrating Python interpreter in VisIt UAL database reader
- Extend datastructure XSDs with standard visualization tags for VisIt UAL database reader plugin, investigate possible extensions to these tags (based on feedback from IMPs).
- Install VisIt UAL database reader for 4.10a, investigate better treatment of arrays of structures

Second half of 2012

- Integration of user-defined custom Python plots in Visit:
  - demonstrate passthrough of a simple custom plot with data postprocessed in Python (ETS diagnostics scripts)
  - demonstrate passthrough of a simple grid stored in the general grid description (simple 2d structured grid example)

Proposed development projects

ITM Python visualization library

The various teams in the IMPs developed a number of custom visualization scripts with Python that generally follow theses steps:

1. Get CPO(s)
    - run/shot/time/user/tokamak/CPO name are set using hardcoded parameters, command line options or 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, …), plot data
    - Outputs plot to graphics file or GUI window

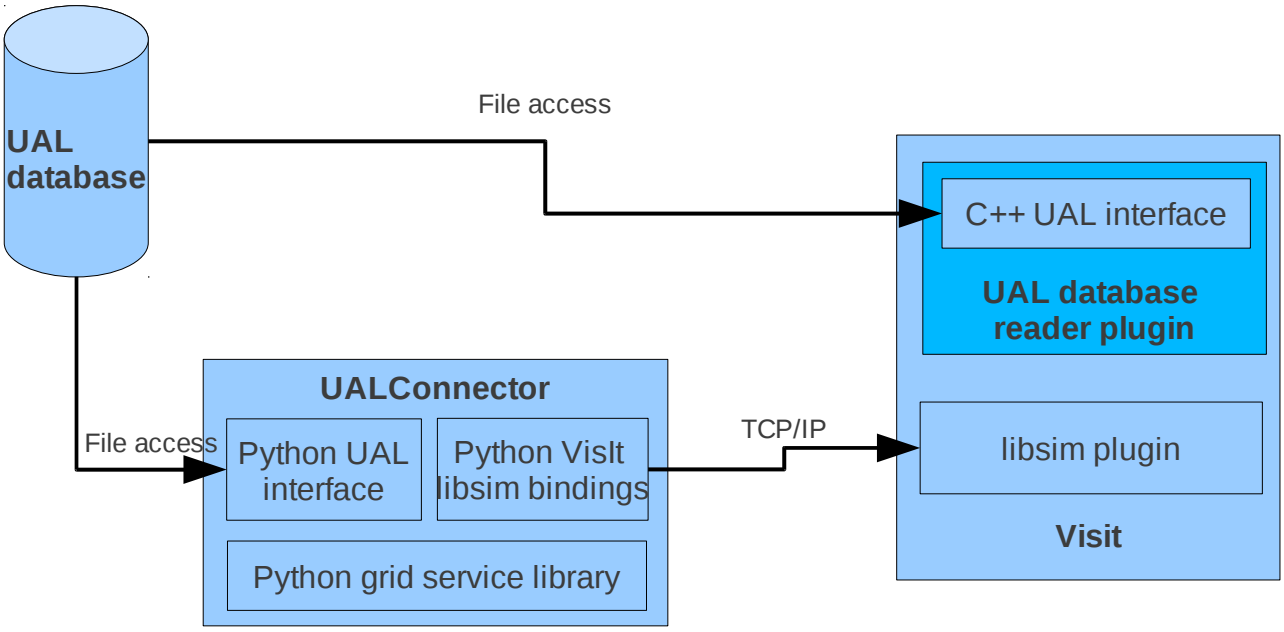
The ITM Python visualization library would provide the following:

- A standard template script(s) covering step 1
    - so that all visualization scripts get a uniform user interface and capabilities
- a structured repository for code fragments that perform the processing in step 2
    - to build a collection of “standard” processing functions, which can then be shared among multiple visualization scripts or other software
    - to make code owned/developed by specific projects available to everybody in a central place (ownership/maintenance questions have to be addressed!)

VisIt UAL database reader plugin ("UAL plugin")

The following proposal addresses the current issues with the VisIt database reader plugin.

Current UAL ↔VisIt setup



There are currently two ways to get data from the UAL into VisIt:

- The UAL database reader plugin: builds on data mappings specified in the data structure definition (XSD files)
- UALConnector: transforms grids and data stored in the general grid description format to VisIt

The only reason for the existence of UALConnector is that it allows the ITM-specific data processing to be done completely with Python. This has the following advantages:

- The Python inspection capabilities can be used to analyze the structure of CPOs to → independent of data structure information in XSDs, **no code generation step required.**
- The Python implementation of the Grid Service Library (GSL) can be used to access data in CPOs using the General Grid Description (GGD)
- Development in Python is much simpler and faster than in C++ (with some drawbacks on performance).

Problems of the UALConnector/libsim approach

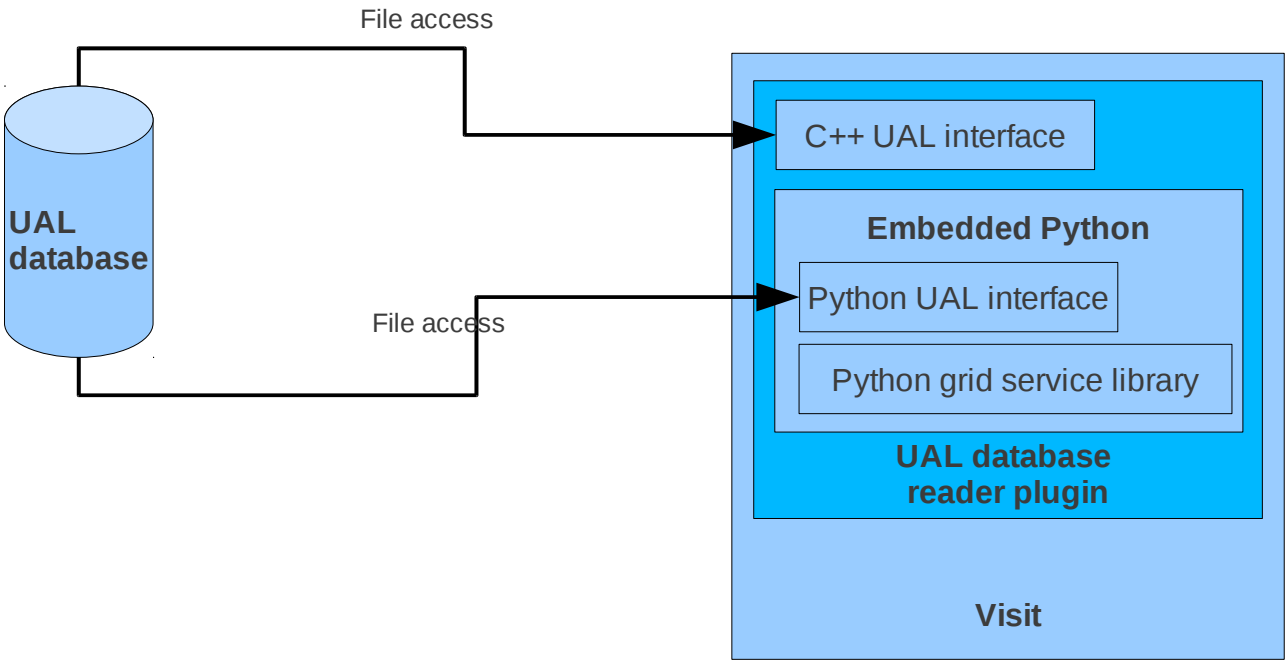
- UALConnector is a standalone process, startup has to be coordinated with VisIt (currently UALConnector launces a VisIt instance)
- libsim does not support time-dependent data → cannot use time-dependent visualization features of VisIt

- Connection between UALConnector and VisIt is established with temporary “simulation” session files, which have random names → makes using VisIt sessions and scripting much harder
- Some arbitrary limitations in the libsim interface (number of variables is limited, not all grid types are supported)

Problems of the current UAL plugin

- The automatically generated C++ code is very hard to extend and maintain
- only a limited number of basic plots are possible
- no support for CPOs using the general grid description

Proposed UAL ↔VisIt setup



This approach embeds a Python interpreter in the UAL database reader plugin. This kind of embedding is a standard feature of Python. The interface is done through the Python/C API.

http://docs.python.org/extending/embedding.html

http://docs.python.org/c-api/index.html

With this approach both the flexibility of Python and the standard plugin concept of VisIt can be used. The functionality of the existing C++ plugin and the Python tools (visualization library, grid service library) can be used at the same time.

Before going ahead with this a prototype is required to show whether the Python embedding

actually works as intended.

Data passed from C++ to Python:

- shot/run/user/tokamakname/backend type
- list of CPO names (possibly 'all')
- time

Data passed from Python to C++:

- Grid & variable metadata
- Grids & Variables (in some VisIt-specific standardized intermediate representation)

Problem(s) of this approach:

- The C++ and Python parts cannot share a CPO data structure → all data has to be read twice (can be avoided by selectively disabling either part, or by supporting the XSD map tags in the Python part)

Further improvements

- Extend current UAL plugin file format to supporting specifying
  - user and tokamak name
  - database backend (MDSPlus or HDF5)
  - multiple or “all” CPOs → data browsing
- combine the embedded Python interpreter with the ITM Python Visualization Library to offer complex composite visualizations through VisIt (achievable through “virtual CPO objects”)
- The complexity of the automatically generated C++ code can be avoided by covering the same functionality with Python:
  - adapt the C++ XSLT transform to produce a simple text file holding the data/grid mappings defined in the CPOs
  - use this data in the embedded Python interpreter to provide the same plots (also achievable through “virtual CPO objects”)