Data structures in practice

F. Imbeaux

Outline

- Data structure : some general principles
- a few examples
- How to use it in the physics codes

Object oriented data structure : CPOs

- Full description of a tokamak : physics quantities + subsystems characteristics + diagnostics measurements

  → Object oriented data structure :

  - High degree of organisation : several subtrees corresponding to « Consistent Physical Objects » (avoid flat structures with long list of parameter names).
  - Substructures correspond to Consistent Physical Object :
    - Subsystem : (e.g. a heating system, or a diagnostic) : will contain structured information on the hardware setup and the measured data by / related to this object.
    - Code results (e.g. a given plasma equilibrium, or the various source terms and fast particle distribution function from an RF code) : will contain structured information on the code parameters and the physics results.

- Codes communicate by exchanging CPOs only (→ data consistency)

CPOs contain also traceability and data managements information

- In addition to physics / technological data
- CPOs contain also traceability and data management information
  - Comments on the data source (e.g. name and version of the code used, code-specific parameters, involvement of other CPOs ...)
  - Whether the CPO has been modified during the simulation
  - Where the information is stored and under which format (for the access library)
  - At the top level : workflow used for this simulation, reference of the simulation in the data base …
- All this information is included in the data structure
- Most of it is updated automatically (a minimum is done by the physics code)

XML-based handling of a complex data structure

- Full description of a tokamak : physics quantities + subsystems characteristics + diagnostics measurements

  → data structure becomes rapidly complicated

  → several tools (access tools, type definitions, documentation …) must be derived from the data structure without errors

- Database structure is defined using XML schemas (arborescence, type of the objects, …). User-friendly tools (XML editors) allow *fast and easy design of the data structure*.

- Small translations scripts allow an *automated translation of the structure for various purposes :*

  – Generate type definitions in various languages

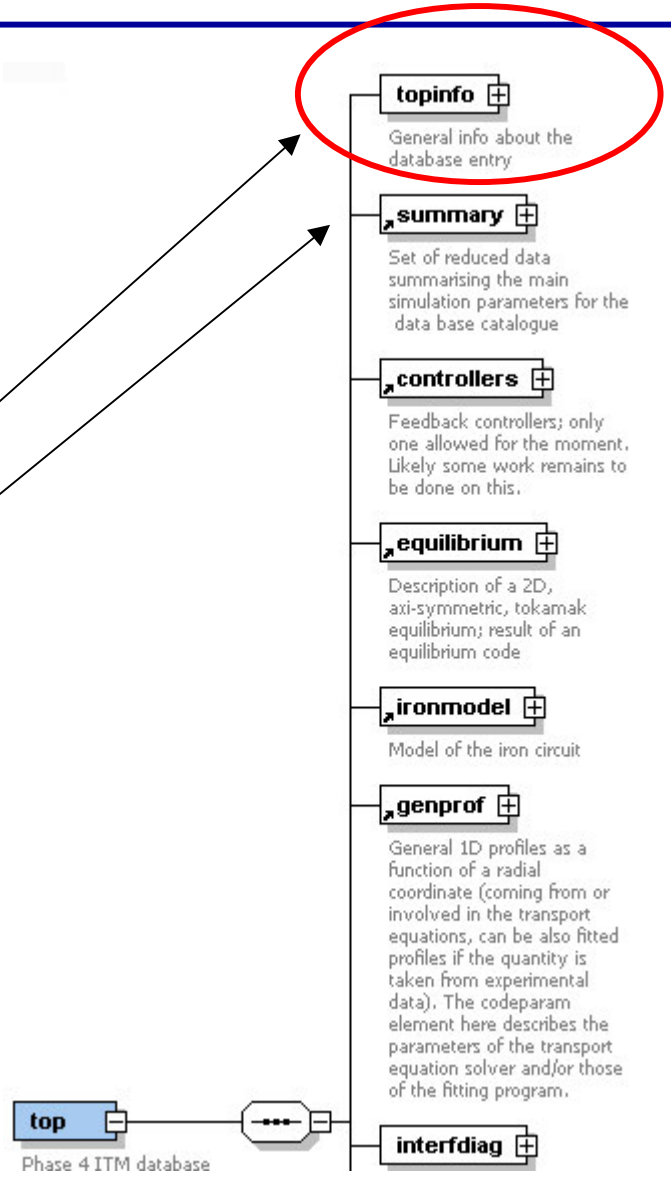
  – Generate access routines to CPO in various languages

  – Generate documentation

  – Extract specific parts of the data structure (e.g. machine description)

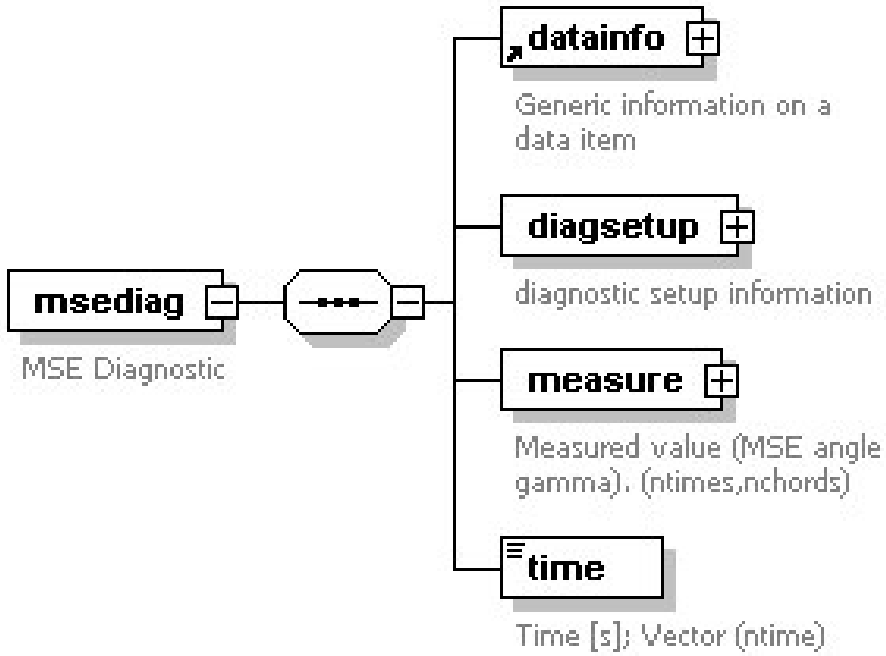
Detailed data structure : TOP

- Just below the top : subtrees representing Consistent Physical Objects : subsystems of the tokamak, or topical code results (equilibrium, MHD, RF, …)
- One general bookkeeping node (contains in particular the reference GPN)
- Set of reduced data summarising the main simulation parameters ("0D") for the data base catalogue



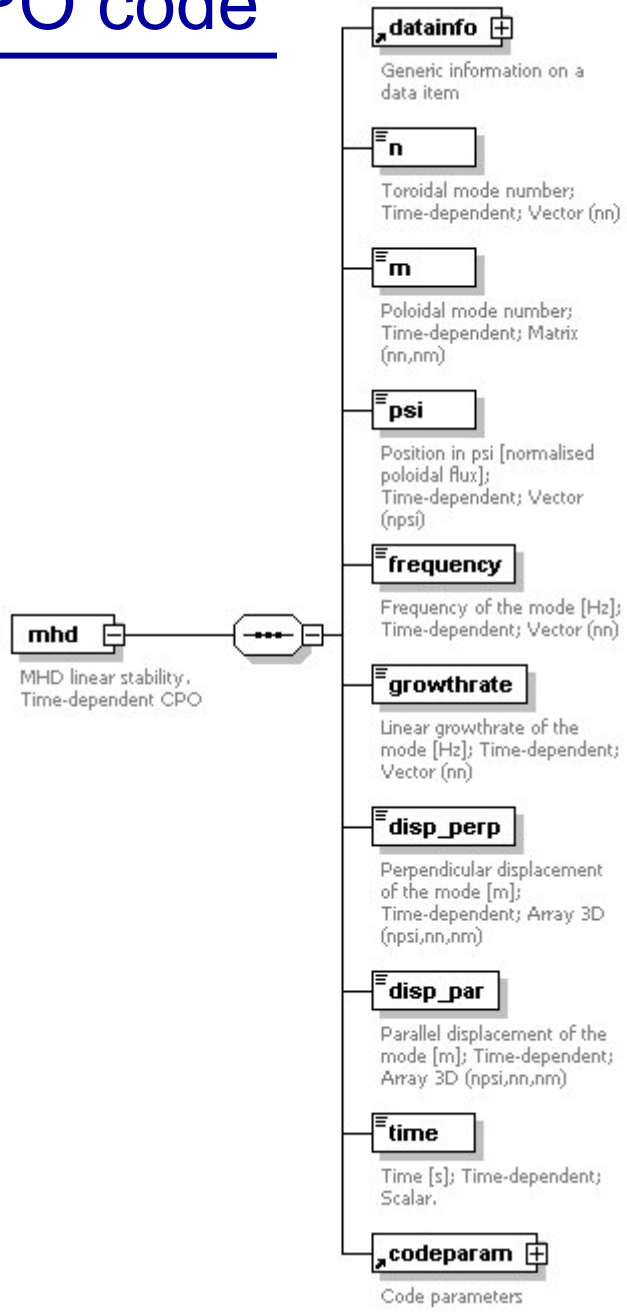
Detailed data structure : CPO diagnostic

- Typical diagnostic structure
- Each subtree (CPO) has its own time array.
- Each subtree (CPO) has one bookkeeping structure.



Detailed data structure : CPO code

- Typical code structure
- Each subtree (CPO) has its own time array.
- Each subtree (CPO) has one bookkeeping structure.
- Stores code results and code-specific parameters



Individual signal documentation

- Name
- Definition
- Units
- Dimensionality
- Time-dependent or not
- Machine description or not (hidden in normal view)
- Topical group for display in graphical interface (for CPOs only)

Working with ITM data structure

- The physics code developer does NOT need to know anything about XML
- The physics code developer must only make his code comply with
  - The logic of the CPOs
  - The hierarchy of data inside CPOs
  - Provide accurate version release and documentation of his code
  - Provide the list of arguments (CPOin, CPOout, number of time slices needed for each of them) and code-specific parameters, in a format to be defined
- ISIP support then wraps the code into the framework and links it to the access libraries (UAL)
- No call to UAL inside the physics code

Example of an ITM code (Fortran) : I/O

- Code must have I/O as :

```
Subroutine Physics_module(CPOin1,….,CPOinN, CPOout1,…, CPOoutM)
 use euitm_schemas     ! contains the type definitions of all CPOs
```

- Declaration of variables (CPO) must be as :

```
type (type_equilibrium) :: my_equilibrium
```
for one time slice

```
type (type_equilibrium), pointer :: my_equilibrium(:)
```
if the code handles several time slices at once

(type_equilibrium is a derived type defined in euitm_schemas.f90, which is generated dynamically from the schemas … but the physics user may ignore this completely)

Example of an ITM code (Fortran) : access to input

- Inside the code, individual signals are accessed by the usual Fortran syntax :

```
my_iplasma = my_equilibrium%global_param%i_plasma
```

- The data tree is described by the schemas and can be found on the ITM web documentation (Data structure page)
- The physics code receives a list of CPOin, from which it reads its input.
- The physics code receives a list of CPOout, from which it reads its code-specific parameters : codeparam structure.
  (codeparam%parameters is filled in advance by Kepler/User Interface)
- Then the code does its usual physics calculations …

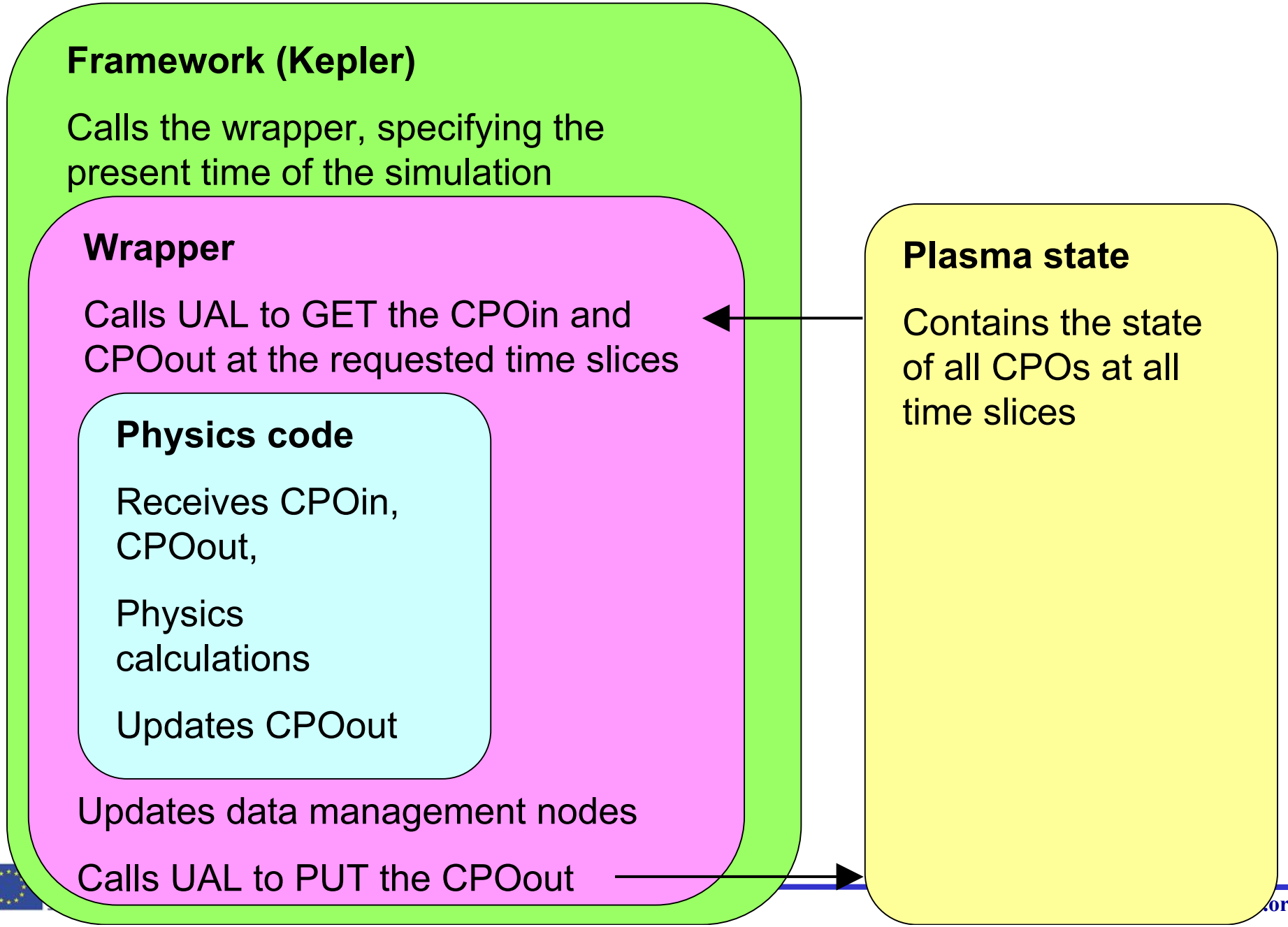
Example of an ITM code (Fortran) : filling output

- By convention, empty signals are coded in the following way :
  - Empty allocatable are unallocated (unassociated)
  - Empty reals are initialised as : -9.0D40
  - Empty integers are initialised as : -999999999
- In most cases, physics code write out a new CPO time slice from scratch :
  - They receive an empty CPO (only codeparam and time are non empty)
  - Fill the variables with the code results :

```
my_equilibrium_out%global_param%i_plasma = my_iplasma
```

  - Can ignore the other variables (which are set by default as empty)
  - Fill the code-specific output (diagnostic on the run , …) : codeparam%output_diag
  - Fill some additional information / comment on the run : datainfo%comment

Outside the physics code



Present status and perspectives

- Data structure completed for IMP #1 (equilibrium). Design ongoing for IMP3 (integration – core and edge transport) and IMP5 (H&CD sources)
- XML tools exist :
  - Schemas
  - HTML documentation
  - Type definitions for C++ and F90
  - MDS+ model tree (database)
  - UAL : first prototypes have been produced and are being tested (C++ and F90)
- Extend DS to other physics areas (IMPs 2 and 4, diagnostics)
- First test of the whole set Framework + DS + UAL : benchmarking equilibrium codes (IMP1 task) : end of this year

Present status and perspectives

- The whole system is under design – some parts significantly advanced
- Practice is needed as the tools become available
- Constructive iterations between users and ISIP are expected to test the relevance of the system – and to improve it