Creating an XML Driven Data Bus between Fusion Experiments

J.B.Lister[1], L. Appel[2], G. Huysmans[3], Ch. Schlatter[1], W. Suttrop[4], and members of the European Fusion Development Agreement Integrated Tokamak Modelling Task Force

[1]*CRPP-EPFL, Association EURATOM-Confédération Suisse, 1015 Lausanne, Switzerland*

[2]*UKAEA-Fusion, Culham Science Centre, Abingdon, UK*

[3]*DRFC, CEA-Cadarache, France*

[4]*Max-Planck IPP, Garching, Germany*

ABSTRACT

The large number of existing tokamak experiments and the increasingly large number of international collaborations in fusion research have brought to the surface an incompatibility problem between data, codes, platforms and languages. A recent initiative by the European Fusion Development Agreement has set up a Task Force to handle these features, under the umbrella of "Integrated Tokamak Modelling". A pilot project has just been initiated to seamlessly link several experiments with 2 classes of physics codes, to search and destroy implementation difficulties and to validate a new conceptual approach. Our first problem has been to generate a common data description, which we have based entirely on the XML Schema language. The user documentation of the variables and their addresses in the tree structure is straightforwardly automated. A second problem is to interface the existing codes to such a data structure, for which we are automating the type declarations from the XML files containing the data. A third problem is to agree on the data bus for putting and getting the data. We are considering the MDSplus definition of a tree structure for the physical storage, since many client interfaces exist in our field and since the user interface corresponds well with what we need. Since XML and MDSplus trees are not structured in the same way, we are forced to take the lowest common denominator, allowing implementation details to creep in. The requirements of such a data bus are being assembled.

We are examining the idea of replacing the data bus by Web Services within a year, to reduce the reliance on legacy components. However, performance at this level will be a major concern and the issue of transferring large volumes of scientific data will have to be addressed.

STATEMENT OF THE PROBLEM

Fusion experiments have developed their own data handling structures as they were built. The small number of experiments constructed and their relatively long lifetime have meant that technologies jump rather than evolve gently in a given installation. Construction has used the best technology or methodology available at the time. Analysis of the data has followed a similarly disconnected path and applying an existing code to another experiment has normally simply meant re-writing the code, adapting it to the new data structures. Of course, this is manpower intensive and generates no advantages.

A notable exception, perhaps the only notable exception, has been the adoption of MDSplus [1], a model based description of experimental data and the underlying tools for acquiring and subsequently retrieving the data, the only database used in more than one installation in the world at present. To our knowledge, no two operating tokamaks use the same Supervisory Control And Data Acquisition systems for plant control. However, 3 tokamaks are using the same plasma control software (PCS [2]) developed by General Atomics. In conclusion, we inherit an extremely heterogeneous set of data structures and analysis codes for handling the data produced by fusion experiments.

As theory and modelling improve, there is increasing interest in cross-comparing different models on given sets of experimental data, and in cross-comparing different experimental results within a single model. Our fusion physics community has realised that to be ready for optimal operation of the future

ITER project, we have to do better. As a result, the European Integrated Tokamak Modelling Task Force was created [3].

This is not only a dawning realisation within Europe, since at the same time, similar initiatives have been launched in Japan and in the USA. Integrating these separately integrated initiatives will be a future challenge, unless they can merge during their initial development, which would of course be ideal.

A further constraint is that the solution adopted for integrating the heterogeneous codes and data sets is that we should have a solution which can migrate over 10-15 years, and can handle evolution in methodology and in technology.

Given this challenge, a sub-group of this Task Force set itself the job of looking for the most suitable solutions with the following features:

- Solution suitable for a long term (10 year) programme
- Solution not linked to any particular manufacturer
- Solution not designed around any particular existing device
- Solution should adapt easily to present data structures
- Solution should be capable of interfacing to existing codes

This paper reports on this ongoing work [4].

PHASE 1 DESIGN

The problem was separated into four parts:

- Defining the data to be shared between codes (units, coordinates)
- Defining the structure of the data to be stored (names, relationships)
- Defining an architecture for communicating between components
- Defining a methodology for implementation

Defining the data to be stored appears trivial, but different groups, different codes and different tokamaks use a variety of definitions. We impose SI units (Metre, Kilogramme, Second, Ampere) together with the electron volt (eV) as the unit of plasma temperature or energy and the degree Kelvin as the unit of “normal” ranges of temperature. Coordinate systems are also a nightmare and we chose two right-handed coordinate systems so that the fundamental physics equations should be respected. Although this appears an obvious set of choices to some, it appears unnatural to others.

Defining the names of variables has not yet converged. Several sets of “universal” naming conventions have been used and all have their weaknesses. Some are linked to old 6-character names, others are generally accepted. None is obvious. Since no universal naming convention can be generated from these sets of names, we chose to adopt new names, apparently self-consistent. In view of the legacy of other names, we have added a synonym table to each set of names, to allow cross-definitions of the names with other frequently used definitions, but only if the physics definitions are the same. We considered case-insensitive a-z0-9 and “_” as suitable. However, even the use of the underscore differs significantly from one group to another and establishing internal consistency is a problem which justifies attention due to the long-term future of this Task Force.

Defining the structure of the data is less critical, but changing the structure later causes hand reworking of the code interfaces, which has to be avoided. A logical grouping in the form of a tree was chosen. Whether or not we can have multiple nodes of the same name was clearly an issue, since this varies from implementation to implementation (MDSplus-no, XML-yes).

Defining the overall architecture has been debated. Two simple solutions are clear:

- A purely relational definition between existing data sources, instructing applications how to find the data they need and leaving applications to store their data where they wish
- A physical representation of the data in appropriate storage, to which applications can read and write.

The second solution was retained, referred to as a Data Bus, underlining its purpose to provide a mechanism for applications to communicate, rather than as a Database, suggesting a too monolithic solution linked to a particular product.

The methodology chosen for the prototype was to represent all the information as XML [5]. Existing XML tools were used to create schemas of the Data Bus and the data themselves were to be stored under MDSplus, due to its wide use in the fusion community.

The choice of XML for this project is analogous to its increasing choice for Enterprise Application Integration, in which a heterogeneous legacy of codes and structures has to be brought together to work coherently. In EAI, there is no attempt to re-write the applications, but to provide a seamless way to allow them to interact. The primary purpose is to define the information which will be transferred between applications, with more emphasis on definition, correctness and control than on performance. XML is gaining ground in the scientific environment in many fields, although often linked to a "private" format for transferring large quantities of scientific data.

These steps chosen during Phase 1 of this project allowed us to feel our way, especially with only basic XML skills.

PHASE 2 DESIGN

In Phase 2, we reiterated our ideas on the design of the Data Bus, selected a 3-Level hierarchy and refined some of the structures. We believed that the choice of XML was suitable on the basis of our Phase 1 experience, with some reservations, to which we shall return.

Level 1 – Abstraction layer

Level 1 provides pure abstraction. It names and defines the variables used, inside a tree structure, and describes their meaning in language, assuming the units agreement. Level 1 is independent of any data from any experiment or model.

Level 1 is implemented as a set of XML Schema language documents. The annotations are used to define the variables in the tree. We chose to avoid all attributes of XML nodes, since this is a notion which does not always exist explicitly in other databases and attributes are only differently named child nodes in an XML Schema. A single Schema defines commonly used utility structures. Sub-trees define functionally grouped structures and a set of top-level Schemas can generate different collections of these sub-trees. The Schemas were graphically edited by the free-ware XMLSpy [6]. The result is a set of schemas which can be quickly inspected graphically.

Documentation of Level 1 is provided by Extensible Stylesheet Language transformations (XSLT) of the XML Schemas. In order to avoid managing the transformed documents, we transform them on the fly using mod-xslt2 (freeware released under the terms of the GNU general public license (GPL) [7]), an Application Program Interface (API), running on the server side (Apache web server). A first approach using PHP was not continued for reasons of effort to maintain PHP, although it provides greater flexibility in principle and in practice. Standard PHP configurations of Linux distributions do not come with XSLT support enabled out of the box and need recompilation every time PHP needs an update. Using XSLT, each Schema can be inspected for a) its source and b) an HTML rendering to provide free-reading documentation.

In view of the evolving nature of this project, we use XSLT to generate the Fortran 90 structure definitions on the fly, avoiding any manual editing of a definition of the full Data Bus. Similarly, we generate a Matlab structure on the fly using XSLT. Run-time generation of these structures is being investigated, but is language dependent.

We are attempting to restrict ourselves to standard W3C functionality, regardless of how attractive a particular proprietary feature might appear to be, to avoid being locked into a subset of products.

Level 2 – Description layer

Level 2 has data in it, and knows about a physical experiment. This information is partly structure, such as the number of items of particular classes (measurements, coil systems, power supplies) and partly values (sizes and positions of physical components). We have decided to implement Level 2 as pure XML. It is a natural choice given the XML Schema definitions and is generated automatically from the Schemas. However, the performance of the XML parsers means that care must be taken not to over-fill the XML data representation. We feel that nodes with more than 1000 characters of data are to be avoided, although this may relax as XML parsing improves. To counteract the parsing performance, we can cache the XML data at the level of the application getting the data. Our XML files can be read into Matlab (for example) and saved as Matlab save files. Reloading checks the version dates and loads the save file, gaining a speed factor of 1000.

Physics data includes not only floating-point numbers and integers, but also vectors and matrices of both floating-point numbers and integers. We have searched for a universal representation of these numbers in XML and we have not yet found what we need. As a result, we have declared (unfortunately in our opinion) a local definition based on it being an ASCII expression which can be evaluated (*value=expression* inside a script) inside TDI (MDSplus) IDL interactive data language and Matlab. This can be summarised as:

Type	**Example**	**Regular expression**
vecint	[1,2,3]	\[(\+|-)?\d+(,(\+|-)?\d+)*\]
vecflt	[1.0,-3e5,-4.0e-3]	\[(\+|-)?((\d+(\.\d+)?)|(\.\d+))((E|e)(\+|-)?\d{1,2})?(,(\+|-)?((\d+(\.\d+)?)|(\.\d+))((E|e)(\+|-)?\d{1,2})?)*\]
matint	[[1,2,3],[4,5,6]]	\[\[(\+|-)?\d+(,(\+|-)?\d+)*\](,\[(\+|-)?\d+(,(\+|-)?\d+)*\])*\]
matflt	[[1.0,2.0,3.0],[5.0,6.0,7.0]]	\[\[(\+|-)?((\d+(\.\d+)?)|(\.\d+))((E|e)(\+|-)?\d{1,2})?(,(\+|)?((\d+(\.\d+)?)|(\.\d+))((E|e)(\+|-)?\d{1,2})?)*\](,\[(\+|-)?((\d+(\.\d+)?)|(\.\d+))((E|e)(\+|-)?\d{1,2})?(,(\+|-)?((\d+(\.\d+)?)|(\.\d+))((E|e)(\+|-)?\d{1,2})?)*\])*\]

Entering the data into the Level 2 XML files can be done in 3 ways. Firstly, we can edit the XML source by hand, which is clumsy, error-prone, user-unfriendly and does not verify the edited structure. Secondly, we can use a form-editor, but no excellent product has been found so far, although the entered data can be checked against the schema for content. Thirdly, we can generate the structured content in a manipulative language, such as Matlab, and then create the XML representation of this data, although this does not allow automatic verification of the content, nor of the structure, although these actions can be performed afterwards, but with no recovery method. The third method is more suited to restarting from scratch while prototyping, although all could use XSLT to rebuild.

The content of the Level 2 XML can be transformed on the fly and presented as user-friendly documentation, as for Level 1.

Initially, we had imagined that the data would stored, at least for an initial period, in the form of an MDSplus data structure and we had developed the way of automatically generating the MDSplus data structures from the Level 1 abstraction, by scripting rather than by transformation. However, it was clear that the need to incorporate heterogeneous data stores was going to arrive and we re-implemented the Phase 1 design with full freedom to choose the data storage. Candidates to be considered include HDF4, HDF5, netCDF, jpeg, flat-files. This means that each data item requires multiple properties including

- One of a restricted list of storage methods (XML, MDSplus…)
- Method of initialising the storage method
- Name of the data within the storage method (/top1/equilibria/eq/nnn001/ip)

- Date of the put and the identifier of the person putting the data

This essentially means generating a description like a URL for obtaining data inside a browser. Generating a unique data item description makes updating this structure easy. Level 2 must therefore be populated with the first 3 of these properties.

XML allows multiple nodes of a given name, which cannot be directly implemented everywhere, specifically not in MDSplus. We therefore generalised a convention that xxx.node[i].yyy would become node xxx.node.NNNiii.yyy with no loss of flexibility and avoiding any MDSplus specific properties in the abstraction layer.

Level 3 – Data layer

Defining a data layer is ambiguous, since Layers 1 and 2 already contain some sort of data and we wish to avoid the use of the term meta-data. Some of the Layer 3 data resides in the Level 2 structures, a type of “intrinsic” data. We find this odd, but have not yet found a reason for querying this choice. It appears to be convenient.

The structure and content of Level 3 are fully defined in Level 2 and the extent of Level 3 is open. The only restriction is that the data extraction should be performed in a way which is describable in Level 2. If the data are to be stored in Level 3 in flat files, then we need a description of the structure and content inside Level 2. We have not so far checked all likely sources of anguish.

Generating the layers

Level 1 is generated from scratch graphically. Level 2 is generated automatically from Level 1 and then filled in by hand or by script. Level 3 contains two categories of data, one which is created by an application, such as a code output flat file, the other which is filled by an application, such as an MDSplus data structure. The first does not need to be primed, the second needs to be created as an instance of the Level 2 structure, exactly as performed by MDSplus.

LESSONS LEARNED

Good news

XML has proven to be friendly and generating the XML Schemas has proven to be straightforward and convenient. On the fly XSLT transformations provide a maintenance-free access to these schemas. Standard tools for manipulating the Schemas are good.

Less good news

Generating the XSLT transformation files (.xsl files) has proven to be “interesting” due mainly to the XPath syntax, to the difficulty of debugging and to the difficulty of reading the finished product. This does not appear to be just a learning curve problem.

Intelligent editing of the XML files, with a Form interface hiding the XML nature of the data is needed. No product appears to be ideal, but an extensive market search has not been performed.

FUTURE

We feel that we are pointing in the right direction and that prototyping is showing us the strengths and weaknesses. We are convinced that the Layer 1, Layer 2 and Layer 3 data will not be lost if we are forced to make a major migration from this architecture. Upgrading the Layer 1 and Layer 2 can be performed in principle using XSLT, although this has not yet been done in anger with a real migration of real data and therefore remains an open question in practice. Upgrading the Layer 3 data would require a specific migration path, provided read/write access is available to all storage standards in a single environment.

Although not part of the initial activity of this Task Force, we have carefully retained compatibility with evolving access to the Data Bus via Web Services. The convenience, platform independence and implementation simplicity make a homogenous interface based on Web Services extremely attractive. Our primary worry is that this work involves circulating Megabytes of physics data between applications and the Data Bus, excluding the use of simple SOAP messages. Some form of private binary "attachment" by MIME/DIME appears appropriate, but we have not yet delved into this. Implementing a call to retrieve data from MDSplus using a Web Service showed a factor of 1000 performance hit compared with a direct server call [8]. The dilemma between the possibilities offered by Web Services and the performance provided by, and required of MDSplus remains a challenge for the future.

The purpose of this conference paper is to incite colleagues with more experience of these environments to come forth with different and hopefully improved ideas.

ACKNOWLEDGEMENTS

We acknowledge the stimulation from the Task Force leadership, especially Alain Bécoulet and Par Strand. Discussions with Christine Vanoirbeck and Basil Duval were appreciated. The work was partly funded by the Swiss National Science Foundation, the UK Engineering and Physical Research Council and EURATOM.

REFERENCES

[1] MDSplus software tools > http://www.mdsplus.org/

[2] PCS http://fusion.gat.com/pcs/

[3] EUITMTF documentation > http://www.efda-taskforce-itm.org/

[4] http://crppwww.epfl.ch/~lister/euitmschemas/

[5] The World Wide Web Consortium > http://www.w3.org/

[6] XMLSpy > http://www.altova.com/

[7] mod–xslt2 > http://www.mod-xslt2.com/

[8] T.W. Fredian, MIT, private communication

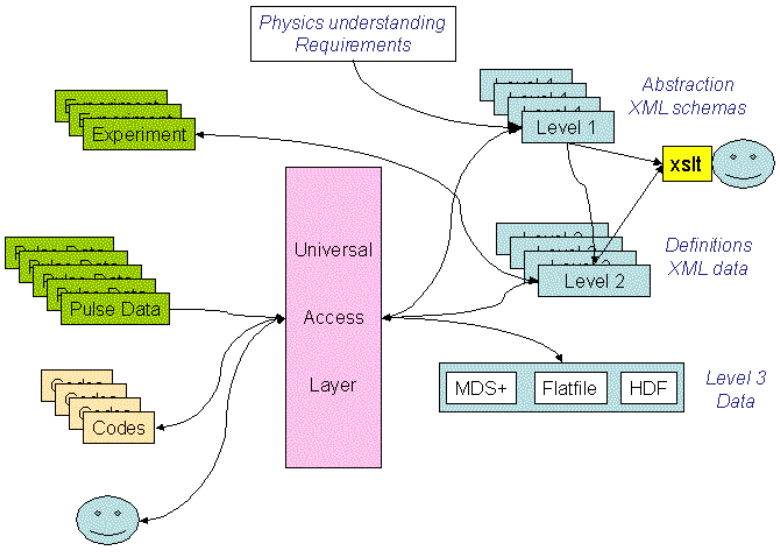

Figure 1 Illustration of the data flow for the Data Bus