The ASDEX Upgrade Parameter Server

Gregor Neu[1], Richard Cole[2], Alex Gräter[1], Klaus Lüddecke[2], Christopher J. Rapson[1], Gerhard Raupp[1], Wolfgang Treutterer[1], Dietrich Zasche[1], Thomas Zehetbauer[1] and the ASDEX Upgrade Team[1]

[1] *Max-Planck-Institut für Plasmaphysik, Boltzmannstr. 2, 85748 Garching, Germany*

[2] *Unlimited Computer Systems, Seeshaupter Str. 15, 82393 Iffeldorf Germany*

Concepts for the configuration of plant systems and plasma control of modern devices such as ITER and W7-X are based on global data structures, or "pulse schedules" or "experiment programs", which specify all physics characteristics (waveforms for controlled actuators and plasma quantities) and all technical characteristics of the plant systems (diagnostics and actuators operation settings) for a planned pulse.

At ASDEX Upgrade we use different approach. We observed that the physics characteristics driving the discharge control system (DCS) are frequently modified on a pulse-to-pulse basis. Plant system operation, however, relies on technical standard settings, or "basic configurations" to provide guaranteed resources or services, which evolve according to longer term session or campaign operation schedules. This is why AUG manages technical configuration items separately from physics items.

Consistent computation of the DCS configuration requires access to all this physics and technical data, which include the discharge programme (DP), settings of actuator systems and real-time diagnostics, the current system state and a database of static parameters. A Parameter Server provides a unified view on all these parameter sets and acts as the central point of access.

We describe the functionality and architecture of the Parameter Server and its embedding into the control environment.

Keywords: configuration, plasma control system, plant systems, schedule, validation

1. Introduction

The ASDEX Upgrade (AUG) Control environment consists of the modern digital discharge (or plasma) control system DCS [1], machine control systems (MCS, about 50 actuator and several infrastructure systems), and around 150 data acquisition systems including ~20 real time diagnostics (rtDiags), which provide input to DCS's control feedback loops [2,3]. Safety and interlock systems ensure that the control environment always operates within safe bounds.

To execute a pulse, all these systems need to be correctly configured: DCS, on the one hand, has to be set up with application processes (APs) to perform specific control tasks and these need to be parameterised to execute the desired scenarios. Actuator systems and real time diagnostics on the other hand need to provide the resources required by DCS to execute its control tasks.

At AUG, the configuration of the control environment is not done from a single pulse configuration description ("pulse type" in JET [4], "experiment program" in Wendelstein-7X [5], or "pulse schedule" in ITER [6]) but implemented as a staged process based on separately managed configuration items: Plant systems are pre-configured with "basic settings"; i.e. specific plant characteristics and operation limits compatible with the pulse goals [7]. The gas inlet system, for example, is configured to provide fuelling capability for specific gas types. In most cases this configuration will not only be valid for the next pulse but for a series of pulses.

The parameters of the pre-configured plant need to be made known to DCS's APs to make sure that these correctly interpret diagnostic input and output correct commands to actuator systems. Another important DCS parameter source are the segmented discharge programs (DP) [8], which usually change from pulse to pulse and contain all reference waveforms to direct the dynamic behaviour of the control environment. The third parameter source for DCS configuration is given byvarious sets of static – not pulse specific – parameters (examples are given in section 2.2). Some of the parameters required by APs may not be available directly from these sources but may have to be computed, scaled, retyped, or reorganised in some way.

The situation therefore is that of several separately evolving data sources (at the top of Fig. 1) and a need to process this data, and tailor its presentation to the clients' need. In information science such a scenario calls for the implementation of a server.

The parameter server not only provides a unified point of access for configuration data but also a natural location for consistency checks (mainly of the DP with the current plant settings). Also, the server itself can be configured to redirect parameter requests to test data instead of the usual source.

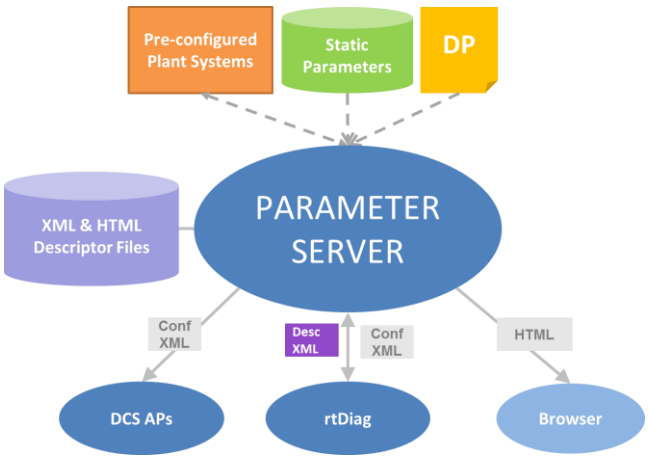

Fig. 1: Parameter server clients and data sources

2. Clients and Data Sources of the Parameter Server

2.1 DCS Application Processes

DCS is the most important client of the Parameter Server. It implements dedicated APs for various control tasks such as I/O (digitised input from sensors or rtDiags, digitised output to actuator systems), evaluation of physical and technical quantities, monitoring, feedback and feedforward control, and actuator management [1]. A reference generator AP computes reference waveforms from set points specified in the DP and a segment scheduler AP directs the reference generator to new scenarios on the occurrence of predefined exeptions (also defined in the DP).

2.2 Parameters for Application Processes

Before each pulse, after a DP has been selected, APs have to configured:

Technical parameters are required by various control tasks: Settings from pre-configured actuator systems, for instance, are used as clipping limits in feedback control, monitoring limits or to inform actuator management APs with actuator characteristics and priorities. Gain matrices for specific feedback operation, machine and default technical limits, linear or non-linear actuator scalings, ADC/DAC scalings etc. are examples of static data, that is typically stored in parameter files or databases and used by control, monitoring, and I/O APs.

Physics parameters come from the DP, which is an XML file edited by session leader to reflect the intended DCS behaviour for a pulse. It is segmented into time slices for phases of a discharge with a specific goal. Some segments e.g. for ramp-up, repair [1], and termination are re-used by many DPs, and have their own life-cycles (ramp-up segments for instance, are typically developed during the first phase of a campaign). Therefore, these segments are not physically contained in DPs but stored in separate segment containers and referenced ("linked segments").

For selected parameters DPs also allows to override defaults from the static parameter database or to specify single values that are used to modify actuator settings (e.g. assignement of valves to a particular control task: main chamber density, divertor neutral gas density, or radiation control) or to check compatibility with these. DPs and other segment containers are all managed in a version controlled repository.

2.3 Other clients

Some key DP parameters of the upcoming pulse are visualised and used by plant system officers to make adjustments to their systems' basic settings (either automatically, or manually). The DP editor uses current plant system data obtained via the parameter server to provide guidance to the editing process. Various sets of parameters are also displayed on the control room screen, the operator user interface, or in browsers to keep people outside the control room informed on the experimental workflow.

3. Functions of the Parameter Server

3.1 Providing Configurations

To describe their configuration all DCS APs and all rtDiags have so-called descriptor files that define mappings of internal quantities, such as inputs, outputs and other process parameters as used in the code, to unique parameter names in a global namespace In some cases values for the global namespace parameter are provided.

It is the key purpose and main task of the Parameter Server to fill in the missing values for all parameters in the descriptor files, providing a complete configuration file to requesting clients (Fig. 2).

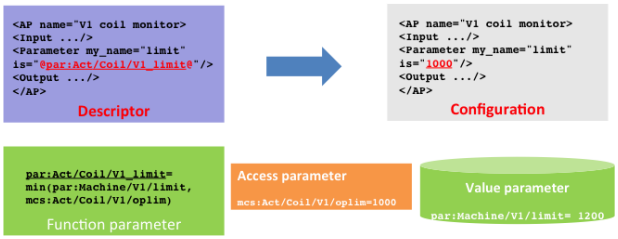

Fig. 2: Mapping descriptor to configuration files

3.2 Validation and Testing

Being the unique point of access for all parameters makes the Parameter Server the natural site to perform checks and validation tasks. It also provides an efficient and elegant solution for several test use cases, which can be realised by simply re-routing parameter requests to test data in a way completely transparent to accessing clients.

4. Parameter Server Architecture

The Parameter Server has several HTTP service ports.

Listener threads spawn an individual service thread for each client request. The service threads use the request

URL and its parameters to trigger the activity of objects in a global data structure: the Object Store.

4.1 Parameter objects (value, access, computation)

All objects in the Object Store are instantiations of a C++ base object class or a derivate thereof. All objects have a "process" (aliased "value" in parameter objects) method, which performs the object's main function and of utility methods – e.g. to represent a self-description of the object in a web browser or an XML file, to signal to the notification service (see below) that the object's value has changed, or to turn caching of values on or off.

Different Parameter Objects populate the Object Store (Fig. 2):

- "Value parameters" are instantiated with constant values. The "value" method simply returns the stored value.
- "Access parameters" implement access functions, e.g. for reading actuator settings. Their "value" function reads the value anew everytime it is called – unless caching is turned on.
- "Function parameters" are objects, whose "value" function will typically perform some computation accessing other objects in the object store. Again, these can be subject to caching.

The Parameter Server also features generic classes, whose "process" method is instantiated from inline Ruby or bash code in a configuration file. When "process" of such an object is called, control is handed over to the Ruby interpreter or a shell, which can read/write and otherwise manipulate the object store using SWIG (REF SWIG) generated wrapper functions. This mechanism opens the path to the use of other scripting languages such as Perl, PHP, Python, Tcl, and many more.

4.2 Other objects

Apart from the parameter objects, the Object Store contains numerous utiliy objects for reading and writing files from/to external sources (DP repository, Apache web server) and validation and consistency check objects.

4.3 Object Store Structure

Objects in the object store are organised as a stack of frames (maps of names to pointers to objects). Objects are always searched by entering the stack from the top (i.e. accessing the last frame pushed on the stack). An object in a higher frame thus overrides objects with the same name in lower frames:

- The lowest frame is the "built-in frame" that is created when the Parameter Server is started. It contains objects, which define basic functions, e.g . for fetching and processing the DP, or for providing the setup of DCS. Other objects define the environment for the operation of the Parameter Server itself.
- The "initialisation frame" is populated with default parameter objects (value, access, and function objects) and standard validation objects. Configuration files drive the instantiation of objects in built-in and initialisation frame. The XML elements in these files correspond to object class names, the name attribute gives the object name and the remaining attributes describe the data type, the value (for value objects only), and the parameters required by the object's "process" method.
- The "DP frame" is filled when the Parameter Server processes a discharge program. Objects in this frame are used to override the default parameters of the initialisation frame, to write new values to actuators, or to store quantities derived from the DPs reference waveforms (e.g. requested plasma current flattop, max. min. of feedforward currents for poloidal field coils).
- The "descriptor frame" hosts objects that originate from AP or rtDiag descriptor files.

At the end of a pulse all frames from the stack except for the built-in frame are popped from the stack thus providing a clean slate for the configuration of the next pulse (all objects are reset to their default).

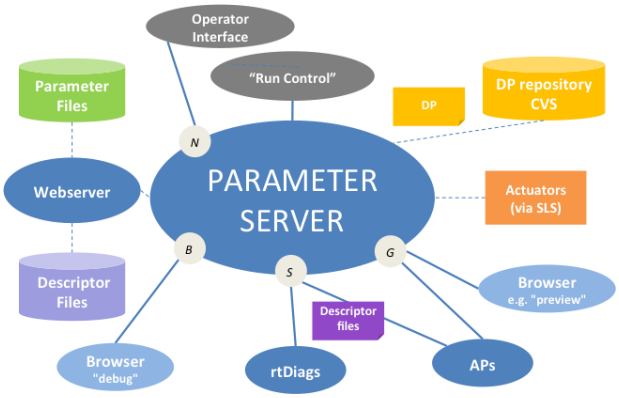

Fig. 3: Parameter server services and interfaces. N: notfication service; B: browsing service, S: standard configuration service, G: generic parameter service

4.4 Services

The Parameter Server currently implements four services (Fig 2):

4.4.1 Standard configuration service:

This service allows submitting descriptor files complying with a particular XML schema (DESC). The files are parsed and an object is created for the file itself and for each parameters for which a value is specified.

When at a later stage APs and rtDiags retrieve their configuration files (schema CONF), the Parameter Server accesses the object store to find the previously submitted file and to resolve all references to parameters before returning it to the client.

Note that the decoupling of submission (descriptor) and retrieval (configuration) allows clients to mutually provide locally defined parameters to each other.

4.4.2 Generic parameter service:

All clients that request files, which are not compliant to the DESC/CONF schemata described above, use this service. A typical example is the request of a Preview by a data acquisition system or the request for the current plant state by the DP editor.

When a client requests a file, the object store is first checked for an object entry with the specified filename. If such an entry exists, its process method is executed – the scheme is used, for instance, to return the DCS setup. Otherwise the file request is redirected to an Apache web server, the file parsed for occurences of parameter references, and the references replaced by the corresponding values before returning the file to the client.

The interface can also be used to browse files with parameter references. File contents are displayed with all references resolved.

4.4.3 Notification Service

In the past, the ease with which clients could request information from the Parameter Server has led to (occasional and accidental) "denial-of-service-attacks" on the Parameter Server, typically through polling of plant system parameters.

Polling is usually done by clients that want to detect the change of a particular parameter in a timely way. This is highly inefficient, especially when several clients poll the same parameters, and hence a notification service was implemented in the latest generation of the Parameter Server. Clients now simply register (and unregister) for notification when an object of the object store is modified. When registering, the client specifies the name of the object, the URL where it wants to be notified, and the protocol to be used for notification (currently only http).

The service is implemented by maintaining a linked list of notification requests and using an internal polling thread, which periodically calls the "value" method for all objects in the list. When the value changes a callback to all requesting clients is executed.

4.4.4 Object Store Browsing Service

This service renders the content of the object store as tree structures of the object names in a web browser. Two views are available: frame-by-frame, where a separate tree is displayed for each frame (the same name can show up in several frames), or global, where a single tree displays all parameter names (if objects with the same name exist in several frames only that of the highest-level frame is shown). The service is used mainly for debugging purposes.

5. Conclusion

Configuring a control environment like that of ASDEX Upgrade is a process and cannot be conceived as "downloading a global configuration data structure" (pulse schedule [6] or experiment program [5]) to the plant. With our flexible experimentation scheme, [9] a better approach is to allow plant systems to (pre)-configure themselves from proprietary datastructures and according to own time schedules, guaranteeing resources to a pulse or a series of pulses. The method of choice appears to be to implement a Parameter Server that allows clients (notably the discharge control system) to access relevant parameters. This unique point of access also permits to check the consistency and validity of configuration data irrespective of its source, and provides a simple and elegant means of implementing in-situ tests by redirecting configuration requests.

Acknowledgments

This work has been carried out within the framework of the EUROfusion Consortium and has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 633053. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

References

[1] W. Treutterer et al., ASDEX Upgrade Discharge Control System—A real-time plasma control framework, Fusion Eng. Des., 89 (3), (2014), pp. 146-154

[2] K. Behler et al., Review of the ASDEX Upgrade data acquisition environment − present operation and future requirements, Fusion Eng. Des., 43 (3‑4), (1999), pp. 247‑258

[3] K. Behler et al., Update on the ASDEX Upgrade data acquisition and data management environment, Fusion Eng. Des., 89 (5), (2014), pp. 702–706

[4] H van der Beken et al., Level 1 software at JET: a global tool for physics operation, Proceedings., IEEE Thirteenth Symposium on Fusion Engineering Volume 1 (1989), 201-204

[5] A. Spring et al., A W7-X experiment program editor‑A usage driven development, Fusion Eng. Des., 87 (12), (2012), pp. 1954–1957

[6] T. Yamamoto et al., Designing a prototype of the ITER pulse scheduling system, Fusion Eng. Des., 87 (12), (2012), pp. 2016-2019

[7] G. Raupp et al., ASDEX Upgrade CODAC overview, Fusion Eng. Des., 84 (7-11), (2009), pp. 1575–1579

[8] G. Neu et al, The ASDEX Upgrade discharge schedule, Fusion Eng. Des., 82 (5–14), (2007), pp. 1111-1116

[9] G. Neu, et al. Experiment planning and execution workflow at ASDEX Upgrade, Fusion Eng. Des., 86 (6‑8) (2011), pp. 1072‑1075