Transforming the ASDEX Upgrade Discharge Control System to a General-Purpose Plasma Control Platform

Wolfgang Treutterer[a], Richard Cole[b], Alexander Gräter[a], Klaus Lüddecke[b], Gregor Neu[a], Christopher Rapson[a], Gerhard Raupp[a], Dieter Zasche[a], Thomas Zehetbauer[a] and the ASDEX Upgrade Team[a]

[a] *Max-Planck-Institut für Plasmaphysik, Boltzmannstr. 2, 85748 Garching, Germany*
[b] *Unlimited Computer Systems, Seeshaupter Str. 15, 82393 Iffeldorf Germany*

The ASDEX Upgrade Discharge Control System DCS is a modern and mature product, originally designed to regulate and supervise ASDEX Upgrade Tokamak plasma operation. In its core DCS is based on a generic, versatile real-time software framework with a plugin architecture that allows to easily combine, modify and extend control function modules in order to tailor the system to required features and let it continuously evolve with the progress of an experimental fusion device. Due to these properties other fusion experiments like the WEST project have expressed interest in adopting DCS.

For this purpose, essential parts of DCS must be unpinned from the ASDEX Upgrade environment by exposure or introduction of generalised interfaces. Re-organisation of DCS modules allows distinguishing between intrinsic framework core functions and device-specific applications. In particular, DCS must be prepared for deployment in different system environments with their own realisations for user interface, pulse schedule preparation, parameter server, time and event distribution, diagnostic and actuator systems, network communication and data archiving.

The article explains the principles of the revised DCS structure, derives the necessary interface definitions and describes major steps to achieve the separation between general-purpose framework and fusion device specific components.

Keywords: control system framework, general-purpose platform, customization, interfaces, re-structuring

1. Introduction

The present experimental devices for thermonuclear fusion research are unique facilities. They differ in geometry, actuator instrumentation, as well as structure material like wall and divertor coatings. Thus, each of them is particularly suited to explore dedicated domains in the huge parameter space of plasma physics, like plasma shaping and MHD effects, heat load and distribution, or particle transport effects, which cannot be studied in a single device.

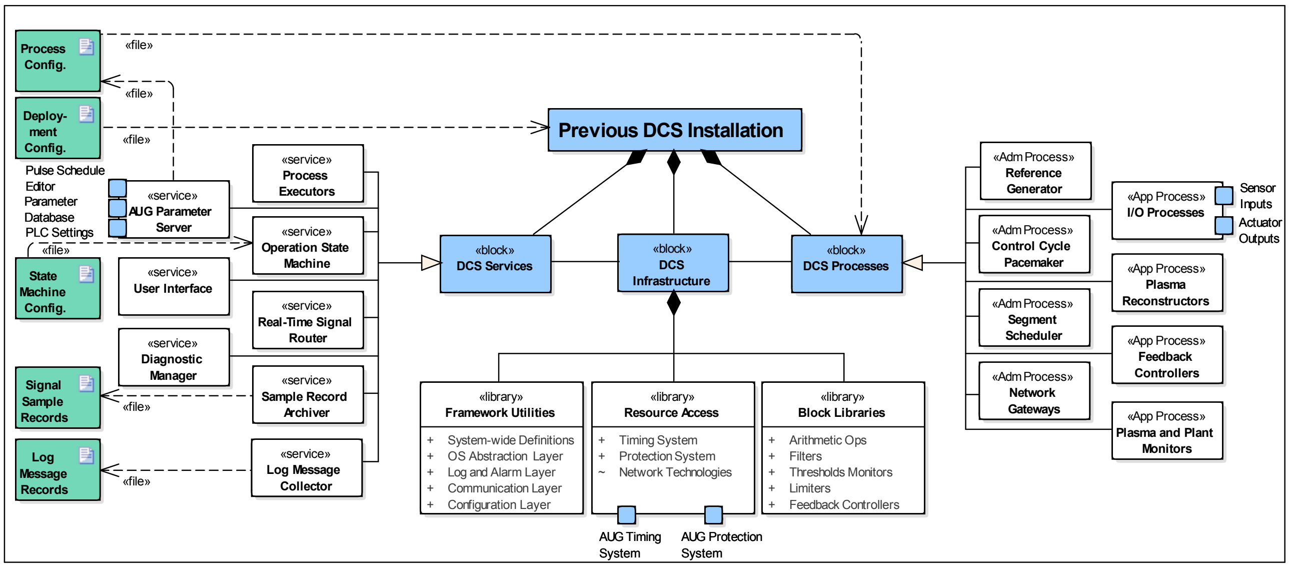
Originally, these devices have been equipped with their own control systems tailored to their respective research objectives and configurations. On the other hand, the core tasks of these control systems resemble each other. Control quantities have to be measured or reconstructed from sensor inputs. Subsequently, a number of single or multivariable feedback controllers, often with PI(D), state-space or relay characteristics, is applied to make these variables track pre-defined reference waveforms. Likewise, quantities are monitored against alarm thresholds, and trespassing them triggers protective or mitigating actions of the control system.

These similarities and the costly effort for maintenance and upgrade are meanwhile driving development towards more general control frameworks that can be customized to a multitude of experimental environments. The most prominent example is PCS, which is used in US, Chinese and South Korean fusion laboratories [1]. In Europe, the MARTe framework, originally developed for subsystems of the large JET tokamak [2] has been deployed at several smaller devices.

The key principles permitting a more general application of control frameworks are adherence to common standards, use of abstraction layers for hardware and software components, as well as modular and configurable component design:

- Adoption of common standards helps to limit the amount of proprietary hard- and software development in favor of readily available and proven products.
- Hardware, especially for data acquisition, actuator control or real-time data exchange is often a site-specific preference. The same applies also for the operating system in use. Such custom components can be integrated in control frameworks with the help of abstraction layers. These introduce generic framework interfaces, as well as specialized component adapters and thus facilitate portability.
- Modularity allows composing individual control system features with more general pre-fabricated building blocks. Configurable modules offer the highest degree of flexibility. While the algorithm is static part of the framework or a library, the module's behavior can be tuned with custom parameter or even functionality settings.

author's email: Wolfgang.Treutterer@ipp.mpg.de


Process Config.
Deploy-ment Config.
State Machine Config.
Signal Sample Records
Log Message Records
«file»
Pulse Schedule Editor
Parameter Database
PLC Settings
«service» AUG Parameter Server
«service» User Interface
«service» Diagnostic Manager
«service» Process Executors
«service» Operation State Machine
«service» Real-Time Signal Router
«service» Sample Record Archiver
«service» Log Message Collector
Previous DCS Installation
«block» DCS Services
«block» DCS Infrastructure
«block» DCS Processes
«library» Framework Utilities
+ System-wide Definitions
+ OS Abstraction Layer
+ Log and Alarm Layer
+ Communication Layer
+ Configuration Layer
«library» Resource Access
+ Timing System
+ Protection System
~ Network Technologies
«library» Block Libraries
+ Arithmetic Ops
+ Filters
+ Thresholds Monitors
+ Limiters
+ Feedback Controllers
AUG Timing System
AUG Protection System
«Adm Process» Reference Generator
«Adm Process» Control Cycle Pacemaker
«Adm Process» Segment Scheduler
«Adm Process» Network Gateways
«App Process» I/O Processes
«App Process» Plasma Reconstructors
«App Process» Feedback Controllers
«App Process» Plasma and Plant Monitors
Sensor Inputs
Actuator Outputs
Fig. 1: Block Definition Diagram of the original DCS Structure

Recently, the control system framework of the ASDEX Upgrade (AUG) experiment, DCS [3], was adopted as a base control system of the WEST project [4] and moved from a dedicated implementation to a general-purpose platform. From its beginning DCS had been built on the above-mentioned principles for portable design. But a distinction between a generally usable core and site-specific functionality had not been necessary before. Therefore, the separation was a non-trivial effort. In this paper we describe the necessary steps to accomplish this goal. Section 2 summarizes the relevant aspects of the previous DCS implementation. Section 3 identifies the areas, where the design was not sufficient for a general-purpose platform. Measures taken to solve the portability issues and the resulting new structure are finally outlined in Section 4.

2. Original DCS structure

DCS is a holistic control system designed to control and coordinate a complex fusion device. The most important concept of DCS is the separation of user modules, which define control purpose and control algorithms, from common services. User modules, in DCS called Application Processes (AP), such as plasma reconstructors, feedback controllers and plant monitors, rather belong to the custom domain. Figure 1 shows that they rest upon a set of infrastructure libraries for common definitions, utilities, resources and configurable function blocks. The framework services on the left of Figure 1 complement these libraries. APs use and access common resources like time, protection system, memory storage or networks only via abstract framework interfaces. On the contrary, data-acquisition with site-specific hardware is managed by Application Processes directly and thus separated from the framework.

3. Required upgrades

These design decisions implement the previously mentioned key principles for generally applicable control frameworks. They already permitted to connect DCS to other external frameworks like MARTe [5] and design tools like Simulink [3]. An analysis of the necessary interfaces between the DCS framework and a different control environment like the WEST project, however, revealed, that these measures were not sufficient. Adaptation is not limited to just the user-defined APs but must be also extended to framework services. Many of these services needed to be linked to existing external systems in the environment of the control framework.

- In both AUG and WEST a timing system built on dedicated hardware is responsible for distribution of an experiment-wide common time information. The framework as a timing system client needs matched adaptors to the respective timing system or, alternatively, to the built-in high-resolution CPU time.
- The sophisticated Parameter Server [6] for the configuration of APs, is an integral component of the AUG DCS framework. Other framework users, however, operate their own parameter service, to which the ported framework must connect, instead. Similar situations exist for message logging and real-time signal data archiving.
- Finally, due to differences in the experimental setup and in the working schemes, the workflows for pre- and post-discharge operations and, hence, the DCS state machine needs adjustment possibilities to the respective operation practice.

In addition, some older parts of the DCS framework required modernization to become mature enough for a broader distribution. The distributed object management middleware CORBA [7] had to be replaced by a more modern tool. Due to historical reasons, the real-time network communication layer was different for Application Processes and for real-time diagnostics. For instance, only real-time diagnostics could be attached to Ethernet networks. To exploit the full power of the distributed structure of DCS this artificial separation had to be removed.

In summary, the basic DCS concepts remained untouched, but the framework boundary and its interfaces had to be re-interpreted to enable adaptation to novel operation environmental conditions.

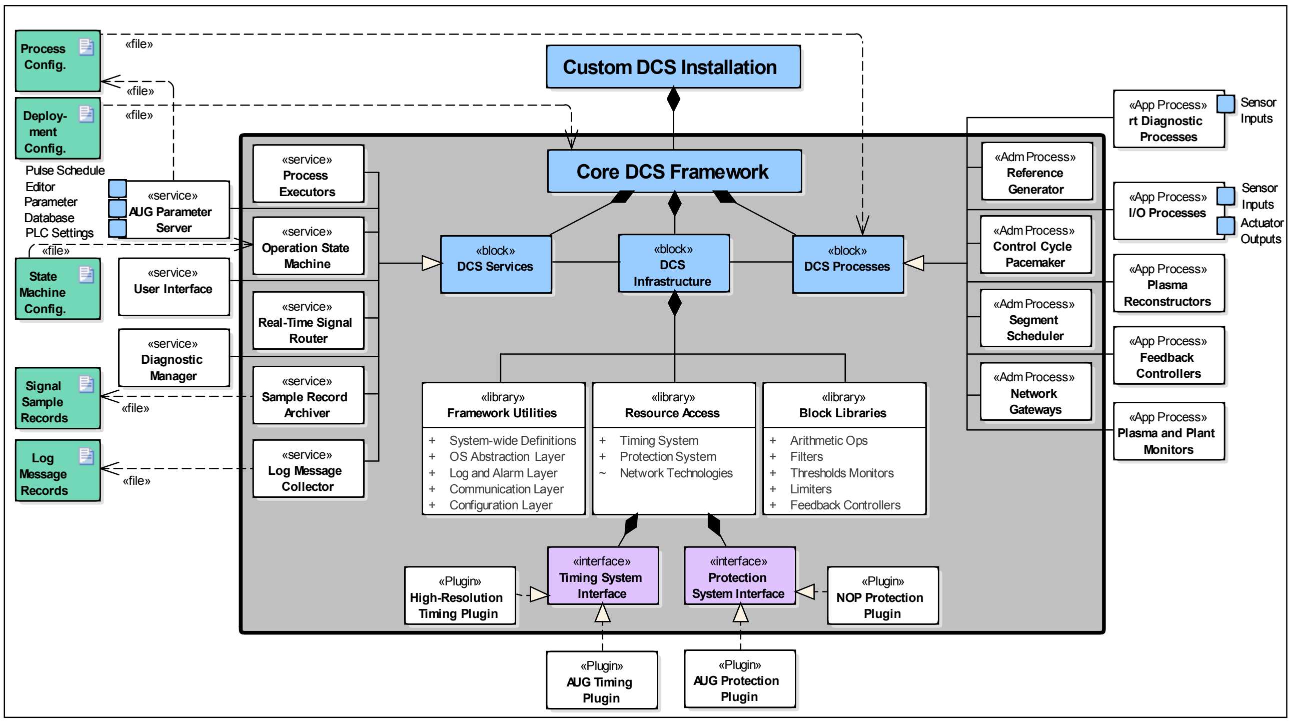

Process Config.
«file»
Custom DCS Installation
«file»
Deploy-ment Config.
«file»
Pulse Schedule Editor
Parameter Database
PLC Settings
«service» AUG Parameter Server
«file»
State Machine Config.
«service» User Interface
«service» Diagnostic Manager
Signal Sample Records
«file»
Log Message Records
«file»
Core DCS Framework
«service» Process Executors
«service» Operation State Machine
«service» Real-Time Signal Router
«service» Sample Record Archiver
«service» Log Message Collector
«block» DCS Services
«block» DCS Infrastructure
«block» DCS Processes
«library» Framework Utilities
+ System-wide Definitions
+ OS Abstraction Layer
+ Log and Alarm Layer
+ Communication Layer
+ Configuration Layer
«library» Resource Access
+ Timing System
+ Protection System
~ Network Technologies
«library» Block Libraries
+ Arithmetic Ops
+ Filters
+ Thresholds Monitors
+ Limiters
+ Feedback Controllers
«Plugin» High-Resolution Timing Plugin
«interface» Timing System Interface
«interface» Protection System Interface
«Plugin» NOP Protection Plugin
«Plugin» AUG Timing Plugin
«Plugin» AUG Protection Plugin
«Adm Process» Reference Generator
«Adm Process» Control Cycle Pacemaker
«Adm Process» Segment Scheduler
«Adm Process» Network Gateways
«App Process» rt Diagnostic Processes
Sensor Inputs
«App Process» I/O Processes
Sensor Inputs
Actuator Outputs
«App Process» Plasma Reconstructors
«App Process» Feedback Controllers
«App Process» Plasma and Plant Monitors

Fig. 2: Block Definition Diagram of the revised DCS structure (ASDEX Upgrade Installation)

4. Revised DCS structure

4.1 Restructuring

The most substantial change was necessary in the code structure of DCS. The distributable general-purpose platform comprises only the core framework. All ASDEX Upgrade specific parts had to be separated. For this purpose, an additional system boundary for the core had to be defined. Subsequently, interdependencies between components on both sides of the boundary were re-assessed. Due to the modular design it was in most cases sufficient to expose and streamline already existing interfaces. Some functions like access to time information, however, required definition of device-independent abstract interfaces and custom plug-ins.

Figure 2 illustrates the new structure using the example of the ASDEX Upgrade installation. The outer box with white background shows the custom domain, while the core framework, accentuated with a grey background, contains the framework infrastructure libraries with augmented interfaces to plug-in custom components and external systems. Basic services are included, as well. Apart from Process Executor services running the real-time processes, they comprise the Operation State Machine, the Real-Time Signal Router, the Sample Record Archiver and the central Log Message Collector. Some useful real-time processes, responsible for administrative and coordination tasks like control cycle pace-making, reference generation, segment scheduling and network transport are also delivered with the core framework, but could be replaced by site-specific custom modules, if desired.

On the client side, the core framework needs to be complemented with application processes implementing the custom control task, with configuration data, but also with modules for dedicated hard- and software support, in particular, client time distribution system, protection system and peripheral I/O devices. On grounds of the before-mentioned integration of the framework in existing environments, services with external linkage are not shipped with the framework and need to be supplied by the customer. The Parameter Server with its strong relations to ASDEX Upgrade's pulse schedule editor and plant PLCs, Diagnostic Manager service, as well as the ASDEX Upgrade shotfile archiving scripts represent this sort of site-specific utilities, which are currently too closely linked to local installations to become part of the core framework and therefore reside in the site-specific AugDcs.

4.2 External Interfaces

Along with framework component restructuring, the framework interfaces have been subject to overhaul. In the end, three interface types are used for customization and extension. The first, already existing, type is based on the object-oriented methods of inheritance and polymorphic function overriding and is used for application processes and custom function blocks which a user derives from base classes defined in the core framework and customizes with his implementation of virtual functions. Here, the C++ base class declaration simultaneously serves as interface definition.

Components created using this classical method, however, need to be compiled and linked together with the framework. The new plug-in feature adds a registration and loading formalism and allows to dynamically load custom binary code implementing a framework interface and call its functions.

The second form, files or file streams in standardized XML formats, is used to exchange structured non-real-time data between the framework and its environment. Application processes, for example, use generic framework methods to read and parse setup data from one or more configuration files and subsequently parameterize or even create process components. In the ASDEX Upgrade case, the Parameter Server provides these files on demand, collecting and pre-processing the contents from various sources. A simpler setup without the need for pre-processing would just use static files. This interface existed already before restructuring but was internal. Now it has been exposed to permit operation without the AUG Parameter Server. Conversely, for each real-time signal the core framework's Sample Record Archiver generates an archive file with all recorded samples in a pulse. Site-specific tools, which may even be simple scripts, can then be employed to convert this information to a custom format and store it in a local database.

Finally, Google Protocol Buffers [8] have been adopted as a successor of CORBA based remote procedure calls. Being designed to efficiently convey data structures between processing units in the form of serialized streams, DCS applies them for internal interaction between framework components as well as for communication with external services. Similar to the xml format mentioned above, protocol buffer message types define the interface for data exchange. A typical use case for this interface in a custom environment would be a graphical user front-end that, communicating with the Operation State Machine, visualizes the framework operation state and conversely translates user commands from mouse-clicks to DCS workflow commands. A number of core framework services, such as Log Message Collector and Sample Record Archiver use Protocol Buffers for input and XML files for output. Connecting directly to the protocol buffer interfaces users can opt for even higher customization replacing these core framework services with their own utilities.

4.3 Build and installation utilities

Core, as well as custom framework components are organized in projects, each producing a build artifact which can be a binary executable, a static or dynamic library or just a set of header files. As before, artifacts need to be installed to a dedicated location, the "Release" folder, after compilation and linking to become visible for dependent projects. This two-step mechanism keeps changes to headers and binaries private during the coding and validation process and thus facilitates concurrent development of projects. The revised build and installation system improves this method by introducing a package management for the installed artifacts. Moreover, project dependency records have been added to the build specifications and enable automatic setup of a hierarchical build and installation chain. The build process follows unified, framework-wide rules and accounts also for build platform peculiarities. Only settings like artifact names, code locations and dependencies are defined by the projects. Thus, the redesign considerably simplifies the build process definition and reduces the effort to create and maintain projects attached to the framework.

Conclusions

Segregating site-specific components from a core framework transformed the original ASDEX Upgrade discharge control system into a general-purpose platform for plasma control with extensive customization facilities. In combination with well-defined and consolidated interfaces the revised DCS framework has versatile features for adaption to other fusion devices.

The restructured DCS has been deployed at ASDEX Upgrade and WEST. ASDEX Upgrade will operate the new system in the forthcoming experimental campaign.

Acknowledgements

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] D. A. Piglowski, D. A. Humphreys, M. L. Walker, J. R. Ferron, B. G. Penaflor, R. D. Johnson, B. Sammuli, B. Xiao, S. H. Hahn, D. M. Mastrovito, "Accumulated experiences from implementations of the DIII-D plasma control system worldwide," Fusion Eng. Des., vol. 85, no. 3–4, pp. 451–455, Jul. 2010.

[2] A. C. Neto, D. Alves, L. Boncagni, P. J. Carvalho, D. F. Valcarcel, A. Barbalace, G. De Tommasi, H. Fernandes, F. Sartori, E. Vitale, R. Vitelli, L. Zabeo, "A Survey of Recent MARTe Based Systems," IEEE Trans. Nucl. Sci., vol. 58, no. 4, pp. 1482–1489, Aug. 2011.

[3] W. Treutterer, R. Cole, K. Lüddecke, G. Neu, C. J. Rapson, G. Raupp, D. Zasche, T. Zehetbauer, "ASDEX Upgrade Discharge Control System—A real-time plasma control framework," Fusion Eng. Des., vol. 89, no. 3, pp. 146–154, Mar. 2014.

[4] R. Nouailletas, N. Ravenel, J. Signoret, W. Treutterer, A. Spring, M. Lewerentz, C. J. Rapson, H. Masand, J. Dhongde, P. Moreau, B. Guillerminet, S. Brémond, L. Allegretti, G. Raupp, A. Werner, F. Saint Laurent, E. Nardon, M. Bhandarka, "From the conceptual design to the first mock-up of the new WEST plasma control system", this conference.

[5] C. J. Rapson, P. Carvalho, K. Lüddecke, A. Neto, B. Santos, W. Treutterer, A. Winter, T. Zehetbauer, "Coupling DCS and MARTe: two real-time control frameworks in collaboration", submitted to Fusion Eng. Des.

[6] G. Neu, R. Cole, A. Gräter, K. Lüddecke, C. J. Rapson, G. Raupp, W. Treutterer, D. Zasche, T. Zehetbauer, "The ASDEX Upgrade Parameter Server", this conference

[7] CORBA Basics, http://www.omg.org/gettingstarted/corbafaq.htm

[8] Protocol Buffers, https://developers.google.com/protocol-buffers