Configuration-defined control algorithms with the ASDEX Upgrade DCS

Wolfgang Treutterer[a], Richard Cole[b], Alexander Gräter[a], Klaus Lüddecke[b], Gregor Neu[a], Christopher Rapson[a], Gerhard Raupp[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 distributed real-time control system executing complex control and monitoring tasks. Up to now, DCS control algorithms have been implemented by coding dedicated application processes with the C++ programming language. Algorithm changes required code modification, compilation and commissioning which only experienced programmers could perform. This was a significant constraint of flexibility for both control system operation and design.

The new approach extends DCS with the capability of configuration-defined control algorithms. These are composed of chains of small, configurable standard function blocks providing general purpose functions like algebraic operations, filters, feedback controllers, output limiters and decision logic. In a later phase a graphical editor could help to compose and modify such configuration in a Simulink-like fashion.

Building algorithms from standard functions can result in a high number of elements. In order to achieve a similar performance as with C++ coding, it is essential to avoid administrative bottlenecks by design. As a consequence, DCS executes a function block chain in the context of a single real-time thread of an application process. No concurrency issues as in a multi-threaded context need to be considered resulting in strongly simplified signal handling and zero performance overhead for inter-block communication. Instead of signal-driven synchronization, a block scheduler derives the execution sequence automatically from the block dependencies as defined in the configuration. All blocks and connecting signals are instantiated dynamically, based on definitions in a configuration file. Algorithms thus are not defined in the code but only in the configuration.

The concept has been developed in view of Simulink block libraries and MARTe General Application Modules (GAM) but extends these with the DCS virtues of distributed computing and multi-threading.

With growing diversity of general-purpose blocks the DCS framework will reach an unprecedented degree of universality and flexibility. Configuration-defined algorithms will gradually replace many existing DCS applications. Finally, the concept might also become of interest for the upcoming ITER plasma control system.

Keywords: plasma control system, function block, configurable functionality, multi-threading

1. Introduction

Thermo-nuclear fusion reactors are envisaged as future options for large-scale power generation. Currently, research is performed on experimental devices to explore operational aspects like material aptitude, optimal operating conditions, stability and controllability. Given the complex matter of the underlying physics and the large number of involved actuator and diagnostic plant systems the control system of such an experimental device must connect to a large number of heterogeneous subsystems and process huge amounts of sensor data with sophisticated algorithms but at the same time provide flexibility for evolution and a clear structure for operation.

The ASDEX Upgrade Discharge Control System (DCS) addresses this challenge building on a framework concept that supplies infrastructure services and connects pluggable user-defined control and monitoring modules called Application Processes (AP) [1].

Up to now, control algorithms have been coded directly into dedicated application processes. While algorithm properties such as gains, dimensions and the linkage to signals were configurable, changes in the algorithm required code modification and subsequent iterations of testing and correction. DCS already employs libraries with re-usable function blocks for frequently used code patterns to reduce the effort of coding and the risk of introducing new errors. An even higher efficiency paired with better user experience could be gained, however, when application algorithms were formulated entirely in terms of such blocks and could be instantiated dynamically on user demand. This approach would extend the scope of configuration data from parameterization of algorithms further to the definition of algorithms. In the final goal of this vision, also physics operators without programming knowledge would be able to formulate and deploy control algorithms assisted by Simulink-like graphical editors. Potential application fields range from measurement pre-processing over state identification and control to event detection and exception handling.

To reach this goal, DCS is being extended with the capability of configuration-defined control algorithms.

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

These are no longer implemented in terms of individual C++ code but are composed of linked, configurable building blocks providing general-purpose functions like algebraic operations, filters, comparators, feedback controllers, output limiters and decision logic. The DCS approach is inspired by Simulink block libraries [2], which allow to build complex simulation models from a repository of standard function blocks by interconnection with signal lines and translates this paradigm into distributed and multi-threaded real-time control context.

The MARTe control system framework [3] also makes use of this idea. It implements function blocks in terms of Generic Application Modules (GAM), connects them via the Dynamic Data Buffer (DDB), employs a scheduler to sequentially execute the GAMs and provides dynamic instantiation based on a configuration file - concepts which are paralleled also in the new DCS approach. MARTe, however, does not yet provide a comprehensive library of standard function blocks such that GAM algorithms usually are custom codes programmed in C++. DCS block libraries are rather focussed on standard functions and in addition comprise handling of sample metadata like timestamp, quality and activity states, which are integral parts of DCS' local exception handling concept [4]. Moreover, DCS seamlessly integrates block-based algorithms with other control modules in a multi-threaded and distributed computation environment by virtue of its Shared Sample Buffer and sample-driven synchronisation features [5].

Compared to traditional application processes, algorithm decomposition in such standard blocks results in a considerably finer granularity and care must be taken to avoid performance penalties caused by scheduling and data transfer with a potentially large number of blocks. Such considerations have considerable impact on design choices, which are further detailed in section 2. Section 3 explains the integration in the global DCS context, while the status of implementation, and future plans and challenges are discussed in the outlook.

2. Building Blocks

Building blocks form the backbone of the configuration-defined algorithm approach. They are derived from existing DCS function libraries. Their function class elements are wrapped into a building block base class defining the uniform interface of all blocks for cloning, customisation, input/output signal administration, initialisation and execution. The block algorithms include local exception handling based on the quality state, which is part of the signal sample metadata. A filter block, for example, thus shows adequate behaviour even in the case of outlier samples marked as invalid. Division blocks react smartly, if the numerator approaches zero. Utilizing general-purpose blocks has the advantage that their correct functionality needs to be tested and commissioned just once but the validated block can be re-used arbitrarily often.

Like application processes also blocks communicate via signals. Major aspects of proven signal concept publish/subscribe based automatic wiring have been adopted from the DCS core framework [6]. Owing to the before-mentioned performance considerations, however, block connection signals are subject to a number of simplifying restrictions:

- Algorithm entities formed by blocks are executed in the scope of an Application Process, which occupies only a single real-time thread. This limitation avoids additional safety measures for thread concurrency and resource locking. Thus, a ring-buffer for sample exchange, associated with extra copy operations, is dispensable. It is sufficient to pass references to the output signal of a block to the consumers.
- Signals between blocks have only internal visibility within the containing Application Process. Therefore, they are called Local Signals. This is a consequence of the lack of accessibility by concurrent threads.
- Sample-driven synchronisation based on semaphores becomes obsolete. Instead, a block scheduler can call the block execution method as soon as the previous block has finished.
- Grouping of Local Signals is not supported. Signal groups play a major role in the overall network exchange of global signals between Application Processes. But their rationales, efficient sample packaging and thread context switching are not relevant in a single thread context.

Currently, a block scheduler derives a static execution sequence automatically from the block signal dependencies as defined in the configuration before the real-time phase. This ensures, that input signals are always updated before they are used for calculation. Algebraic loops are not permitted. The sorting algorithm follows rules outlined in the documentations of Simulink [7] and acslX [8] simulation tools. In a later stage, when support for conditional block workflows will be added, this design might change, because the optimal sequence could be state dependent and scheduling in real-time might become an attractive alternative despite the overhead it implies.

3. Integration in DCS Application Processes

In the DCS concept Application Processes have the role of algorithmic entities whose execution is solely defined by the availability of input data. The execution order of dependent Application Processes is thus determined by the dependency chain of their input and output signals. Independent algorithms, on the other hand, can be run concurrently. Therefore, the framework assigns them separated real-time execution threads, which might run on different CPU cores and even on distributed nodes. The framework supplies signal data transport across CPU and network boundaries transparent to the Application Processes and establishes a synchronising signal sample flow that automatically determines the thread scheduling such that the overall workflow is data-driven.

Building blocks can be used to define the algorithm of an Application Process. However, algorithm inputs and outputs need to be transferred from the global signal domain of the Application Process to the domain of Local Signals exchanged among blocks and vice versa.

Ports, a dedicated class of blocks assume this interface task. They comprise both a classical DCS signal or signal group with synchronisation, time-stamp based sample lookup and inherent signal archiving, and a set of corresponding Local Signals. Thus, they do not only act as gateways between function blocks and the external world but they also can provide convenient services like triggering application execution or externalising intermediate algorithm results for archiving and inspection. Figure 1 shows an example where neutral density calculation from ionization gauge current measurements is modelled in a block diagram of 11

$$n_0 = \frac{K_1 I_{el} I_{ion}}{(I_{el} - I_{ion})(K_2 I_{el} + 1)}$$

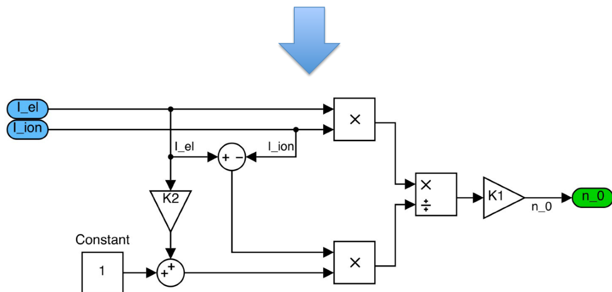

Fig. 1: Example algorithm (neutral density calculation from ionization gauge currents) and corresponding block diagram

standard function blocks. Figure 2 illustrates, how the block diagram translates to DCS building blocks in an Application Process, which communicate with each other facilitated by a Local Signal Exchange Layer and with other control tasks via port blocks and the framework's Shared Sample Buffer.

The Application Process is also responsible for dynamically instantiating and customising blocks as specified by the configuration. Their configuration is specified in a dedicated XML style called AP_CONF, which, apart from support for basic DCS elements like parameters and signals, also allows specification of composite types called DcsObjects [1]. As the Block class is derived from this base it can easily make use of the associated generic parsers and object factories. To complete the picture, each Application Process built from blocks features a block scheduler and a Local Signal Agent for setting up signal data exchange as described below. After the AP configuration has been parsed, block objects are dynamically instantiated by the DcsObject factories, and stored in an AP inventory for further management.

Subsequently, all blocks announce their local signals to the Local Signal Agent which is in charge of the "wiring" from block output to input signals. This task is analogous to the one that Signal daemon (SignalDM in Fig. 2) performs for global signal communication across network boundaries. The block scheduler re-uses the signal map computed by the Local Signal Agent to compute the execution order sorting input ports to the beginning of the sequence and output ports to the end. Finally, the global signal components of the ports are registered at the framework's signal routing service for

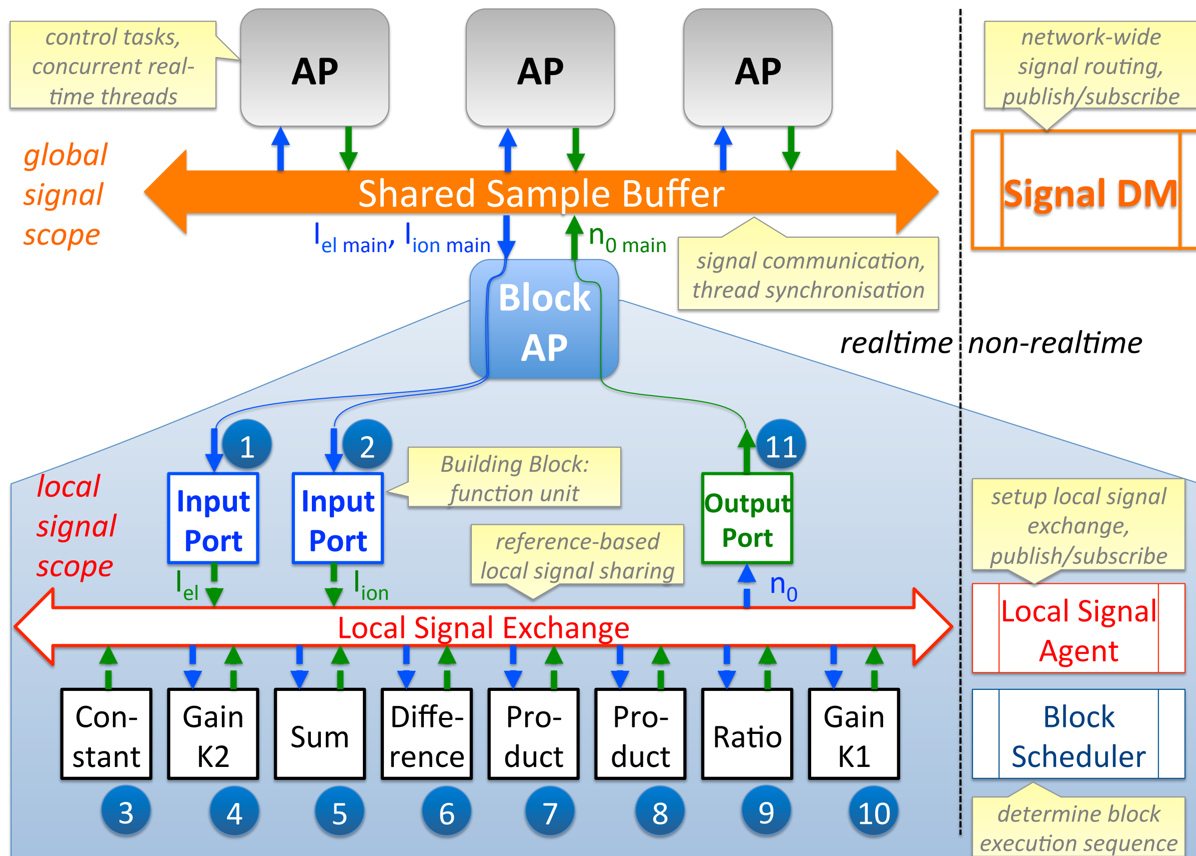

Fig. 2: DCS with Application Process consisting of building blocks

connections with other control applications.

During runtime operation, the first input port, designated as pacemaker, synchronises the application by waiting for its inputs, while the block scheduler will sequentially call each block's execution method according to the previously determined order. Output ports conclude the block chain execution and publish the algorithm outputs such that they become visible to the external world. Other control application threads waiting for the results can now be scheduled.

4. Outlook

Major parts of the configuration-defined algorithm concept such as the DcsObject base class, the AP inventory, Local Signals and the Local Signal Agent, as well as an initial stock of standard function blocks have been already implemented. The block scheduler and the factories will follow by September 2015. First operation is envisaged for the end of 2015.

Future development is foreseen to augment the variety of functions in the block library. Advanced features like conditional branching and super-blocks consisting of other blocks will pose novel challenges. It is also planned to develop graphics-based and text-based editing tools, to ease the construction of configuration files even for non-expert users making rapid development and on-the-fly modifications of control algorithms feasible.

Given the tempting possibility of accelerated development, however, the risk of introducing semantic errors in configuration-defined algorithm should not be underestimated. The fact that individual block behaviour is tested must not distract from the possibility that the combination of blocks can be defective. Thus, careful validation and commissioning of modified configurations using e.g. flight simulators is still indispensable.

The new concept will further boost the universality and flexibility of the DCS framework. Control processes composed from building blocks will gradually replace many of the current DCS applications.

Acknowledgement

This work has been carried out within the framework of the EUROfusion Consortium and has received funding from the Euratom research and training programme 2014-2018 under grant agreement No 633053. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

References

[1] W. Treutterer, R. Cole, K. Lüddecke, G. Neu, C. Rapson, G. Raupp, D. Zasche, and 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.

[2] The Mathworks Inc. , Simulink product website, 2015, http://www.mathworks.com/products/simulink.

[3] A. C. Neto, F. Sartori, F. Piccolo, R. Vitelli, G. De Tommasi, L. Zabeo, A. Barbalace, H. Fernandes, D. F. Valcarcel, and A. J. N. Batista, “MARTe: A Multiplatform Real-Time Framework”, *IEEE Trans. Nucl. Sci.*, vol. 57, no. 2, pp. 479–486, Apr. 2010.

[4] W. Treutterer, G. Neu, C. Rapson, G. Raupp, D. Zasche, and T. Zehetbauer, “Event detection and exception handling strategies in the ASDEX Upgrade discharge control system”, *Fusion Eng. Des.*, vol. 88, no. 6–8, pp. 1069–1073, Oct. 2013.

[5] W. Treutterer, G. Neu, G. Raupp, D. Zasche, T. Zehetbauer, R. Cole, and K. Lüddecke, “Management of complex data flows in the ASDEX Upgrade plasma control system”, *Fusion Eng. Des.*, vol. 87, no. 12, pp. 2039–2044, Dec. 2012.

[6] W. Treutterer, G. Neu, G. Raupp, T. Zehetbauer, D. Zasche, K. Lüddecke, and R. Cole, “Real-time signal communication between diagnostic and control in ASDEX Upgrade”, *Fusion Eng. Des.*, vol. 85, no. 3-4, pp. 466 – 469, 2010.

[7] The Mathworks Inc. , Simulink documentation, 2015, http://www.mathworks.com/help/simulink/ug/controlling-and-displaying-the-sorted-order.html.

[8] AEgis Technologies Group, Inc., "acslX Language Reference Guide", Version 3.0, 2010, http://www.acslx.com/support/Documentation/Language%20Reference%20Manual.pdf, Chapter 3.3, p. 25.