TESLA Report 2006-10

DOOCS patterns, reusable software components for FPGA based RF GUN field controller

Piotr Pucyk[a]

[a]Institute of Electronic Systems, Nowowiejska 15/19, Warsaw, Poland

ABSTRACT

Modern accelerator technology combines software and hardware solutions to provide distributed, high efficiency digital systems for High Energy Physics experiments. Providing flexible, maintainable software is crucial for ensuring high availability of the whole system. In order to fulfill all these requirements, appropriate design and development techniques have to be used. Software patterns are well known solution for common programming issues, providing proven development paradigms, which can help to avoid many design issues. DOOCS patterns introduces new concepts of reusable software components for control system algorithms development and implementation in DOOCS framework. Chosen patterns have been described and usage examples have been presented in this paper.

Keywords: DOOCS, FPGA, control system, software patterns, RF GUN, FLASH, DESY, HEP

1. MOTIVATION

Control systems (such as DOOCS[1] *) provide a software framework in which one can develop applications integrating the hardware with other systems in the experiment. These applications usually consist of device server and GUI panel for operator. The most important is the device server. It should provide remotely the full functionality of the hardware to the operator in the control room and in some cases execute algorithms which are not implemented in the device but interact with it.

For new developments for FLASH† (and future European X-FEL‡), where the new hardware and software is meant to be replacement for the existing system in the accelerator, due to the high reliability and maintainability requirements for the new design - every part of the new system must be fully tested before the integration with the experiment. Usually programmers have limited possibility to test all application features. To perform software tests they must have the hardware connected to the real (not simulated in laboratory) signals. Usually, the access to the real signals is not always possible without interruption of the whole experiment so not all hardware features are used to test the software. There are only a few *machine studies* periods per year during which programmers can test and debug the software in the operation environment. Sometimes, during these studies, there is also a need for quick implementation of some new algorithms or adding new features to the application. In order to minimize the time needed for the software development and debugging one can use well known and proven programming paradigms. They are commonly called *software patterns*. Every software pattern is a solution for common programming issue and is designed on the abstraction level that allows for implementing the pattern in many different software projects.

For FLASH experiment, DOOCS based device servers must provide not only the hardware parameters for the operator but also perform more complex algorithms, real time data acquisition, etc. The complexity of these applications is much bigger than common *instrumentation* servers which provides only few parameters of the device that it servers. Distributed digital systems like SIMCON[2],[3] where the computation power is split between FPGA chip, DSP processor, embedded Power PC and control system application require complex device servers with many sophisticated dependencies between parameters which can be used by all components of such system.

This paper describes the concept of the DOOCS patterns - reusable software components for DOOCS framework. They have been specially designed to be used with FPGA based RF GUN controller , but as can be also applied in most of application based on this control system.

*Distributed Object Oriented Control System

†**F**ree Electron **Las**er in **H**amburg

‡X-ray Free Electron Laser

2. DOOCS PATTERNS

DOOCS patterns are built on the top of DOOCS properties [Fig. 3]. They do not modify DOOCS libraries or other original DOOCS components but only extend the basic functionality. The idea behind the project is to use very basic *D_fct* classes representing *integer*, *float* and *spectrum*. Based on those, a number of new properties, helper classes and data flow templates have been developed. The set of DOOCS patterns can be divided into three groups. First represents the data types and structures for direct accessing SIMCON memory space. They are in most cases overloaded data functions originally provided by DOOCS. The second one includes data structures and routines for server internal data flow automation. Third one is the communication channel[4] - low level interface which provides transparent layer between software and hardware. This solution allows to connect to the hardware through many independent *channels* like VME, Ethernet or RS232.

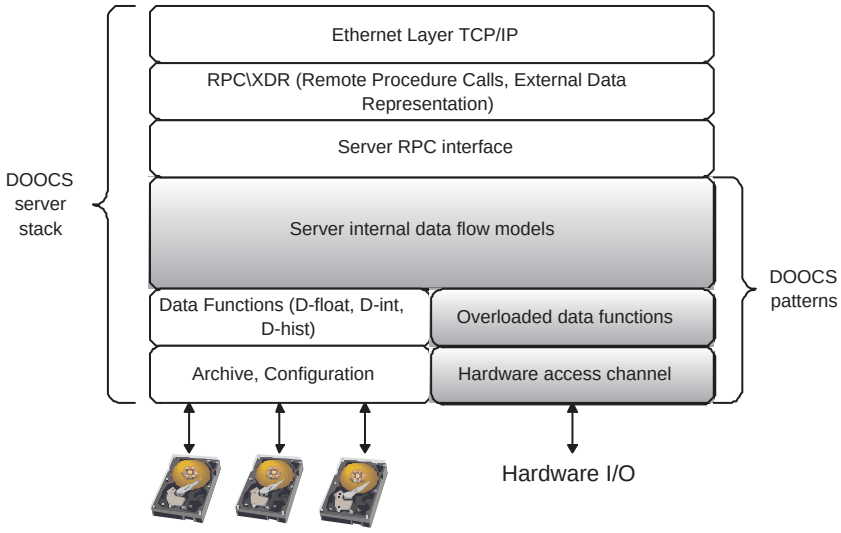

Figure 1. DOOCS patterns placed in the hierarchy of DOOCS server stack. They overload some functionalities and provide new features, but do not modify any of existing DOOCS core libraries and its functionalities.

2.1. DIRECT HARDWARE ACCESS

DOOCS provides an easy interface for accessing device registers. In each *D_fct* class there are two complementary methods for reading and writing the property[§] value from the client. One has to overload those methods and add routines which access device register i.e. through VME bus. There are three main ways through which device parameters can be accessed:

1. Raw data parameter set by the user is written to the hardware "as it is" without any conversion (i.e. memory address). Usually it is not negative integer value of various length (1,8,16,32... bits).
2. User parameter which is converted to device raw data. Usually it can be float value which is scaled to maximum range and then rounded before writing to the device or negative number converted to raw data understandable by hardware.
3. Internal server data can be a value written to the hardware as a algorithm output. These registers are usually not accessible by the user since they have no direct meaning of any physical parameter.

[§]The author uses *property* word in the meaning of *DOOCS property* a data type (implemented as a C++ class) which is available through the network to represent device parameter. This meaning was taken from DOOCS nomenclature.

2.1.1. SIMCON registers, bits and areas

SIMCON memory space is available to the user through the Internal Interface[5] which basic data types are: bits, words and memory areas. Words can be grouped in the structures called *multiple words*. They are then accessible as a memory areas, but have different than areas implementation inside FPGA chip. As it was mentioned earlier in this chapter, FPGA registers can have different meaning. They can represent memory addresses, integer values or scaled and rounded float numbers. Most of them, if they are not positive values, have to be also converted to raw data format. DOOCS patterns introduces new *D_fct* classes which not only allow to write data to hardware and read it back, but also convert raw data to engineering units and vice versa. This is done by adding new parameters to the constructor of the basic *D_fct* class; *ratio* and *offset*. In addition, every value is converted to U2 notation before applying in hardware. This solution allow to use few universal data classes for accessing device registers which have various meaning and different data format (from user point of view). Instead of hard-coding these values inside the functions one can add them as a object constructor parameters.

Non scalar data like vectors can be also easily read from and written to the device. DOOCS patterns introduces new, improved *D_spectrum* classes which allows for plotting data with conversion to engineering units. In addition one can use dedicated spectrum classes, which online calculate amplitude and phase from I and Q components. In new *D_spectrum* all conversion parameters can be DOOCS property too. It means, one can change the conversion values "on the fly" during DOOCS server operation. This solution appeared to be essential for effective device operation (see *use cases* section).

In some cases, there is a need to upload into the hardware a vector of values. Original *D_spectrum* class provide `set` method which can be overloaded in order to make such transfer possible. The problem occurs however during server restart, because there is no archiving for *D_spectrum* class! Therefore DOOCS patterns introduce extended version of *D_spectrum* class with archiving. One can write spectrum parameters once (i.e. from Matlab), and they will stored locally on hard drive.

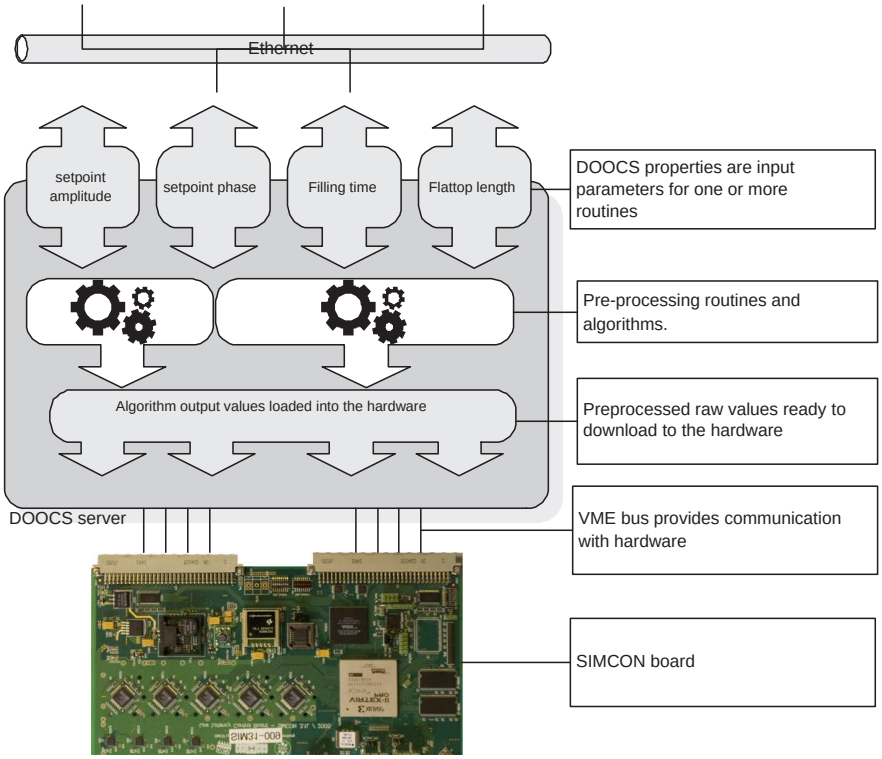

Figure 2. In SIMCON system some data are pre-processed before applying it to the hardware. there are many subroutines which have many - sometimes common input properties.

2.2. SERVER INTERNAL ROUTINES

Accessing direct hardware registers is not always sufficient. There are cases when data must be precalculated before applying them in the device. In addition, some algorithm can be a part of so called *slow control*, that means they do not have real-time requirements for calculation time. In order to save FPGA and DSP resources they are usually implemented in the device server. DOOCS as control system is not designed to manage complex algorithms integration inside the application. Therefore DOOCS patterns introduce several software components and templates which can be solution for common programming problems in algorithms development for LLRF. Below chosen examples have been described.

2.2.1. Property callback system

In the DOOCS framework the property data processing is being done in one of two methods of the property class (usually `value() and set_value()`). One is executed when a client changes the property value, the second one is used when client wants to read back the current value. This is typical for the situation when client wants to put the value to the device register or read it back. Sometimes it is necessary to perform some data *preprocessing* before applying it to the hardware. It can be simple scaling (like in properties for direct hardware access described before in this paper) or complex algorithm with many input parameters adjusted by the user and output values applied to the hardware [Fig.2]. What is more, there can be many algorithms which are using the same property as a input parameter. The update of such parameter value should trigger the algorithm calculation. In DOOCS there is no ready to use framework that manage such dependencies. DOOCS patterns include interface and data types that necessary to implement this scheme.

The *property callback system* bases on publisher-subscriber pattern and callbacks paradigm. The DOOCS property is in this solution the publisher which provides *publisher* interface. It publishes the notification about its value change to all subscribers. A subscriber is every subroutine (usually not a DOOCS property) which has implemented *subscriber* interface. As it is shown in Fig. 3, the publisher can have unlimited number of subscribers since it has dynamic linked list implemented inside with pointers to the subscribers objects. The *subscriber* interface includes method which are triggered by publisher every time the value inside the property is changed. In this method, subscriber subroutine can perform some calculations, update data in the device or read it back.

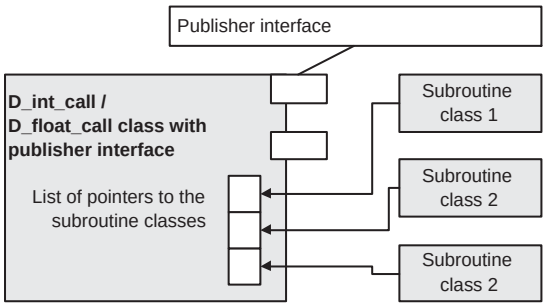

Figure 3. D_int_call and D_float_call classes have publisher interface through which subroutine can subscribe for notification about property value change. One property can be an input parameter for many subroutines. Therefore a dynamic list has been used for adding new subscribers.

This solution allows for creating various algorithms subroutines with many inputs being DOOCS properties and automatically execute calculations if the property changes. What is also important, once created property does not have to be changed. If any subroutine need its value for calculation it can simply register itself in this property, and the property will notify the subroutine about changes. This helps to maintain the code and quickly add new features.

2.2.2. Nonlinearities compensation module

The second example of internal server patterns has more concrete application. In control systems for accelerator, devices often often used to drive other electrical or mechanical devices using current, or voltage. In real world

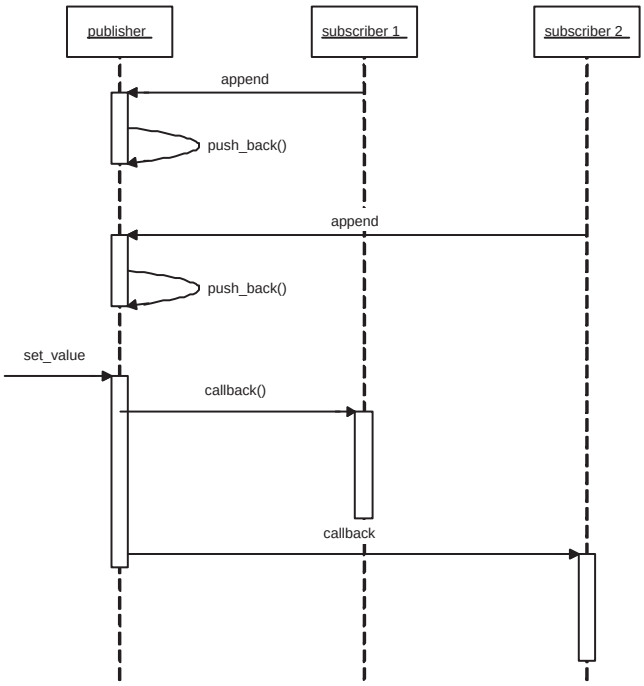

Figure 4. Interaction sequence between publisher and subscriber. Subscriber uses *append* interface to add itself to the subscribers list in the property. When the value inside property is changed, publisher uses *callback* interface provided by subscriber to notify about the change.

most electrical devices have non linear characteristics it means they do not reacts with exactly linear change of output signal for linear change of input signal. The example of such devices are klystron, which saturates, vector modulator with not exactly linear characteristics or even ADCs which are not linear in the whole input voltage range. Usually one can measure the characteristics, calculate correction value and then apply them in the

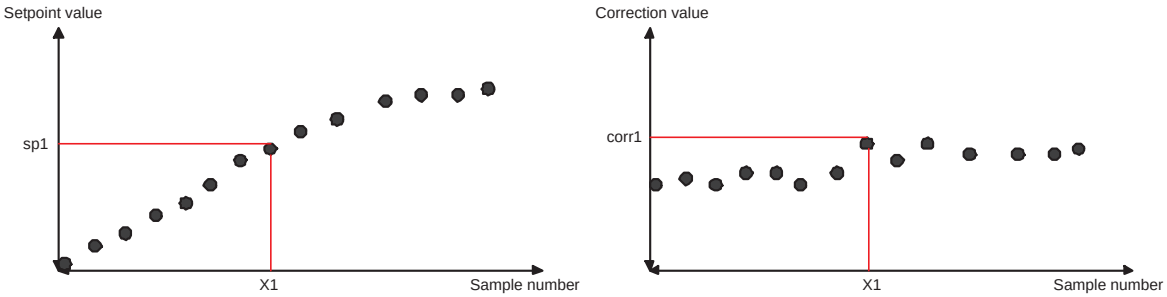

Figure 5. Two tables for user data contains setpoint values for which the correction was calculated and correction coefficients. For the sp1 value in the setpoint vector the corresponding correction value is under the same index in correction table (x1) as the sp1 is in the setpoint table.

algorithm. The problem occurs when the nonlinear element is changed - then one have to change the correction values in source code. The nonlinearity compensation module introduces an easy way to add flexible software components for compensating any nonlinearities. It gives the user freedom of choosing how the correction curve should look like, how many points it should have and how are they spread. The idea of the module is presented in Fig. 5. There are two *D_spectrum* objects; one is the setpoint table with values for which the correction has been calculated. The second one is the table with calculated correction coefficients.

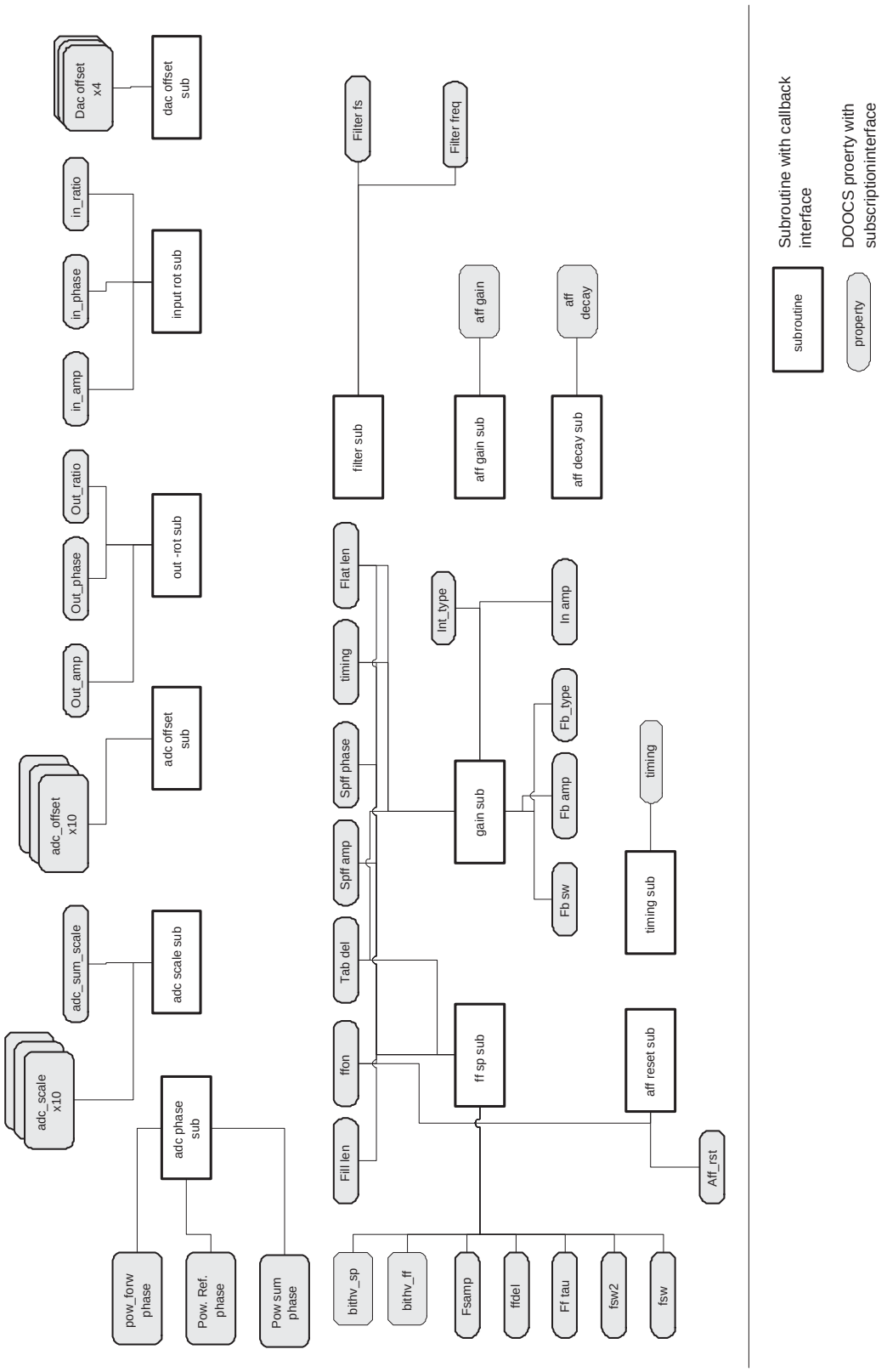

Figure 6. Simplified diagram of dependencies in DOOCS server for RF GUN controller. One can see usage of callback property pattern. Some of DOOCS properties are used by more than one routine.

If one wants to get correction value for a given setpoint *sp1* the module looks for the *sp1* value in the setpoint table. If it find the exact value it read its index in the table. Then the correction value is under the same index in correction table. If the given setpoint is not in the setpoint table, the module uses the nearest setpoint values, next read back corresponding correction values and calculate the final coefficient using linear interpolation. Both, setpoint and correction tables can be updated during operation. Presented pattern store data locally, so they can be automatically restored with server restart.

3. USE CASES AND EXAMPLES

Every routine, and data structure described in previous section has been developed and used in the real experiment. Most of them are currently used as a part of DOOCS server for RF GUN controller in FLASH. They create complex and efficient device server (Fig. 6). In this chapter several usage examples of implemented patterns are presented.

Nonlinearity compensation module has been used to solve the following problem. The setpoint of the gradient of field in the RF GUN is set from DOOCS gui panel. Then it is translated into voltage at the output of the DAC. This voltage is then driving vector modulator and amplified in the chain of pre-amplifiers. At the and it is driving the klystron which provides power to the GUN. Unfortunately, the linear change of the DAC voltage does not cause the linear change of klystron voltage. In this situation, the real power provided to the cavity was different than the one set on the panel. The solution was to use nonlinearity compensation component. The correction values have been calculated with step of 0.01 MW between 0 and 4MW. This gives 400 points. Measured values have been loaded into *D_spectrum* tables and used as a additional parameter for calculating feed forward tables.

In the second example of nonlinearity compensation, the problem occurred during the change of setpoint phase. It appeared that with phase change, the amplitude of the signal changes in a range of 12% (Fig. 7). The reason was the nonlinearity of vector modulator and not exact 90 degree angle between I and Q components. Again the solution was to implement nonlinearity compensation. This time the setpoint values were the angles and the correction values were applied during feed forward tables generation. The compensation of the nonlinearity decreased the gradient change down to 2%.

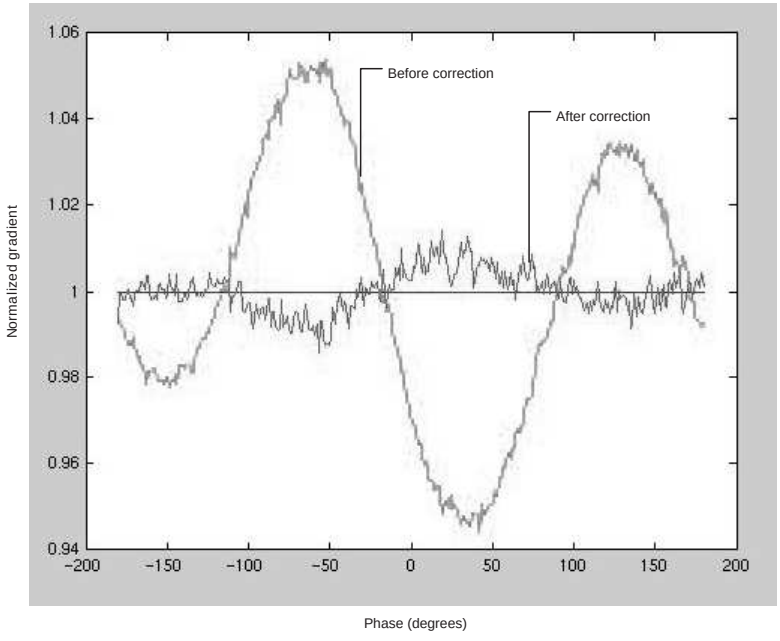

Figure 7. Two plots represent the change of the gradient casued by the change of the set point phase. Before the correction the error was about 12%. After correction it went down to 2%.

Property callback system was used in many subroutines inside the DOOCS server for RF GUN (Fig. 6). The output of each subroutine is loaded into hardware (which is not showed in the picture to keep it readable). The most complex subroutine is the *"sub ff sp"* which is responsible for calculating feed forward and setpoint tables. It uses over ten properties, all created as a "publisher" properties. The algorithm for control tables generation evolved during many operation tests.

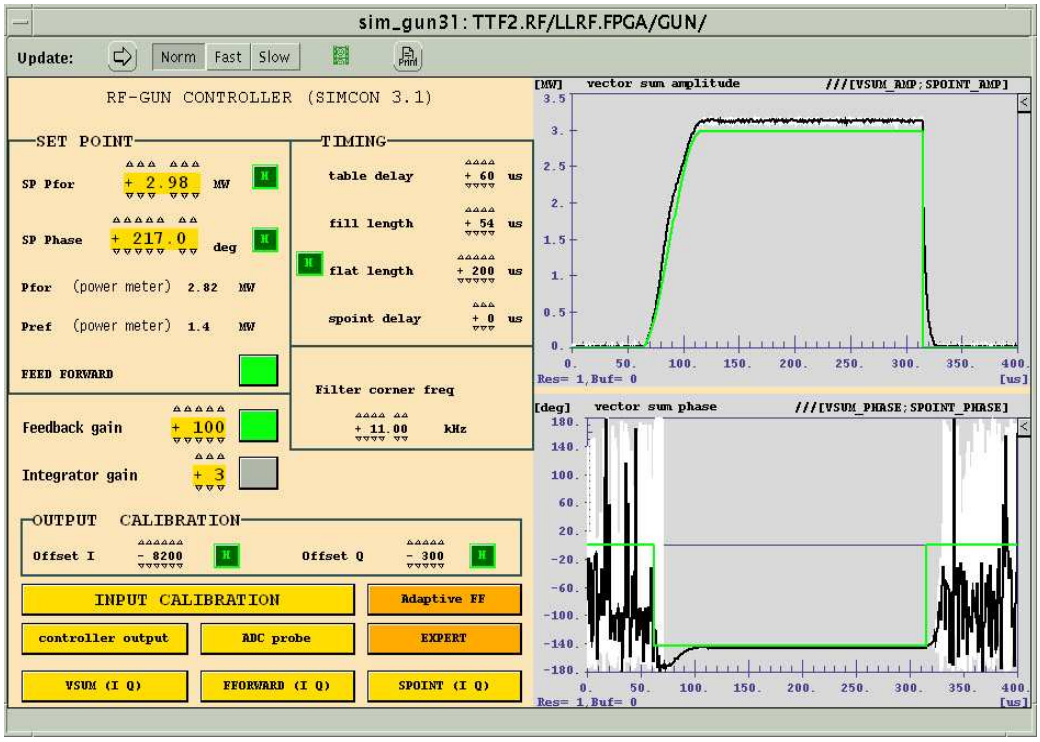

Figure 8. DDD(DOOCS Data Display) panel for RF GUN controller in FLASH. One can see a number of widgets representing various input parameters for control tables generation. Most of these parameters are implemented as callback properties and every user change of theirs values cause the recalculation of new control tables (right site plots).

DOOCS patterns has been also used and tested in FLASH linac in Hamburg, In PITZ institute (Zeuthen, Germany), and FNAL (Chicago, USA) as a part of front end servers controlling the FPGA based hardware. They helped to adapt the system to unique conditions (software and hardware environment) in all those places by providing maintainable source code and clear interfaces with modular design. Any changes in servers that were done in order to integrate SIMCON and the software with different environment have been done using mainly DOOCS patterns.

4. SUMMARY

Design patterns can speed up the development process by providing tested, proven development paradigms. Effective software design requires considering issues that may not become visible until later in the implementation. Reusing design patterns helps to prevent subtle issues that can cause major problems and improves code readability for coders and architects familiar with the patterns. DOOCS patterns presented in this paper are the result of over two years of experience in DOOCS applications development for distributed, digital systems like SIMCON. They were tested in operation conditions, helped to keep the source code maintainable, providing modularity and unified interfaces for communication and device operation.

Results presented in this paper prove the usefulness of reusable components in algorithm development in control system environment. The architecture of these modules ensures the compatibility with future DOOCS libraries and allows to add new components. Currently running software, like DOOCS servers in FLASH are

already using the DOOCS patterns which benefits in easier code maintenance, shorter time needed for debugging, and more flexibility for improvements. They helped to solve many existing problems in LLRF system improving overall operation of the linac.

REFERENCES

1. O. Hensler and K. Rehlich, "Doocs manual," 2001. http://doocs.desy.de.
2. K. Pozniak, T. Czarski, W. Koprek, and R.Romaniuk, *DSP Integrated, Parameterized, FPGA Based Cavity Simulator & Controller for VUV-FEL SC Cavity SIMCON version 2.1. re. 1, 02.2005 User's Manual*, TESLA technical note, 2005.
3. W. Koprek, P.Pucyk, T. Czarski, K.T.Pozniak, and R.S.Romaniuk, "Dsp integrated parameterized fpga based cavity simulator & controller for vuv fel - simcon ver.2.1. installation and configuration procedures; user's manual)," Tech. Rep. 6, DESY, 2005.
4. W. Koprek, P. Kaleta, J. Szewinski, K. T. Pozniak, T. Czarski, and R. S. Romaniuk, "Software layer for fpga-based tesla cavity control system (part 1)," Tech. Rep. 10, DESY, 2004.
5. K.Pozniak, M.Bartoszek, and M.Pietrusiski, "Internal interface for rpc muon trigger electronics at cms experiment," in *Photonics Applications II In Astronomy, Communications, Industry and High Energy Physics Experiments,*, *Proc. SPIE* **5484**, 2004.

ACKNOWLEDGEMENT
This work was partially supported by the European Community Research Infrastructure Activity under the FP6 "Structuring the European Research Area" program (CARE – Coordinated Accelerator Research in Europe, contract