EPICS DATA ACQUISITION DEVICE SUPPORT

V. Isaev, N. Claesson Cosylab d.d, Ljubljana, Slovenia
M. Plesko, K. Žagar, COBIK, Solkan, Slovenia

Abstract
Every control system has to deal with a large number of input/output devices which offer a similar kind of capabilities. For example, all data acquisition (DAQ) device offer sampling at some rate, which in many cases is configurable. If each such device were to have a different interface, engineers using them would need to be familiar with each device specifically, requiring more time for familiarization, inhibiting transfer of know-how from working with one device to another and increasing the chance of engineering errors due to a miscomprehension or incorrect assumptions, which brings forth integration, maintenance and upgrading (replacement) issues. Also, implementation of device’s interfaces would be more costly than necessary, as for every type of device a whole set of documentation (interface definition, requirements specification, test plans, etc.) would need to be produced.

Here, an attempt to standardize such interfaces and address the mentioned issues is described. Nominal Device Model (NDM) is a model which proposes to standardize the EPICS [1] interface of analog and digital input and output devices, as well as image acquisition devices (cameras).

INTRODUCTION
EPICS provides a native interface for integration of devices which is called device support. Physically, device is a board (or other hardware unit) which can provide various functions. This interface allows the developer to define and register device specific functions for reading data from, and writing data to, an underlying device. These functions can be called from IOC shell commands or through the Process Variable (PV) record (in/out records). In the second case, developer also has to define and register their own EPICS devices. Fig. 1 depicts pure EPICS device support architecture.

First attempt to standardize device interface in EPICS context was made by introducing records which provides an interface for specific devices. E.g. EPICS stepper motor record is an interface for the stepper motor.

In EPICS context, device interface could be standardized in two ways: using a single process variable (PV), or with multiple PVS. In the first case, the device’s parameters are covered by PV record’s fields. stepper motor record is an example of single PV interface. In case of multiple records interface, each device’s parameter is represented by one record (one to one relation).

asynDriver
asynDriver was developed to simplify EPICS device support, which implements core for asynchronous operations, defines set of EPICS devices that covers most developer’s needs and provides a convenient way to connect PV record to handler functions in the source code. It also covers a number of standard communication interfaces (serial interface, Ethernet, Gpib, and etc.). These features eliminate routines related to EPICS device definition and PV record connection establishment. asynDriver assumes that multiple PVs are used to define an interface with a device, and therefore makes use of standard EPICS record types (ai, ao, bi, bo, waveform, etc.). System architecture with device-specific asynDriver is represented on Fig. 1 (b). asynDriver is recommended as a useful and convenient way to bring asynchronous functionality to a driver, and is already used by the multitude of device-specific drivers developed with it.

Figure 1: EPICS device support architectures.

A recent attempt of device class generalization in EPICS was made by Mark Rivers in his application of asynDriver for controlling area detectors (CCDs) which is called areaDetector [2]. areaDetector is a module which provides a general-purpose interface for area (2-D) detectors in EPICS. areaDetector supports a large number of cameras and can be extend with plugins that allow manipulation of acquired images (e.g., image processing or storage). The Nominal Device Model described here-in extends the principles of the areaDetector also to analog/digital input/output devices.

NOmINAL DEVICE MODEL
Nominal Device Model (NDM) provides generalized interfaces for analog and digital input and output devices. If we were to look closely at the device-specific asynDriver code, we can identify the parts of the drivers

ISBN 978-3-95450-139-7
which will be very similar from driver to driver especially if we are talking about devices of similar kind (see Fig. 2).

Firstly, the PV record templates are defined in EPICS database files. Then, asynDriver’s interfaces are implemented. Finally, there is code for interrupt dispatching from the device to the asynDriver, and in turn, the EPICS record.

**Behavioural and Structural Model**

The model is described in terms of *structure* and *behavior*.

Structure is defined by a class inheritance hierarchy of device, channel groups and channels (see Fig. 4). All objects have a list of functions that may be invoked upon them. Functions on objects can be made accessible via EPICS by associating them with write and read operations on a record. In addition, a device can have a number of initialization parameters that defines the device (e.g. the hardware address or device file of the low level driver). Parameter values are set at IOC initialization stage inside EPICS’ *st.cmd* start-up scripts.

The meta-model defined in this way covers a wide range of devices:
- Data acquisition (DAQ).
- Signal generators (analog output).
- Digital input and output devices.
- Cameras and other image acquisition devices.

NDS provides a description of standard **triggering** mechanism. NDS implements software triggering. This mechanism should be overwritten by the device-specific if hardware supports native triggering. However, the specification given by NDS for specifying triggering condition should be adhered to.

**Messaging** is a mechanism which allows sending or receiving text messages to model’s objects. Model has a set predefined messages, e.g., to list the device’s capabilities, to issue a self-test and to upgrade firmware.

**Object state machine** describe possible states and transitions between them. State machine has three kinds of event handlers which allow developers to interfere in transition: possibility to veto the transition, state exit
handler and state entry handler. NDS defines state machine at device and channel levels.

NDS provides **EPICS interface** which is represented by a set of EPICS database templates. Developer can expose required functionality to EPICS by including these template files in the product’s database file.

**Extensibility**

NDS doesn’t define any constraint on the interfaces; therefore a developer is free to extend any interface. Developers can define their own PV record, message types, etc. Extension includes two simple steps: implementing required behavior by implementing a handler function, and registering the handler. Registration requires one line of code, and implementation is a regular member function of a C++ class.

For example, by leveraging the extensibility of NDS, we also plan to standardize support for a “nominal data acquisition system”. It will consist of a timing board and a combination of a digital input/output device, whose actions (sampling, signal generation) will be synchronized to the timing board.

**USER SUPPORT**

Development of device specific NDS driver starts by instantiating the EPICS application template. NDS provides a template of an example application which would then be customized by device-specific drivers. Instantiation of a template provides a ready-to-build device-specific NDS driver and an EPICS application for testing it. So immediately after instantiating application it can be built and run.

**CONCLUSION**

NDS is a model described in terms of structure and behavior. Structure of the driver is defined by C++ classes and can be easily extended. In this way, the default behavior of model’s components can be overridden and specialized for particular device.

NDS thus reduces driver the effort required to develop device-specific functionality, as well as make the use of NDS-based devices more uniform.

**ACKNOWLEDGMENT**

We would like to thank the ITER Organization, in particular Stefan Simrock and Petri Makijarvi, for their input and feedback on the design and implementation of the nominal device model.

**REFERENCES**

[2] Area detector module  
http://cars.uchicago.edu/software/epics/areaDetectorDoc.html
http://www.aps.anl.gov/epics/modules/soft/asyn