PROGRAMMING INTERFACES FOR RECONFIGURABLE INSTRUMENTS

Matej Kenda, Hinko Kočev, Tomaž Beltram, Aleš Bardorfer, Instrumentation Technologies d.d, Solkan, Slovenia

Abstract
Application Programming Interfaces (APIs) provided by the manufacturers of the instruments for the accelerators are a very important part of the functionality. There are many interface standards (EPICS, TINE, Tango,...) and even same standard can be used in various ways.

Important features of modern instruments are reconfigurability and embedded computing.

The developers of instruments that need to be connected to a control system are facing different requirements: adherence to standard protocols and support of reconfigurable instruments with diverse capabilities with a consistent interface.

Instrumentation Technologies has implemented a well accepted solution with its proprietary Control System Programming Interface (CSPI) layer and adapters for each standard protocol.

There are new challenges like reconfigurability, quality of service, discovery and maintainability that are being addressed with improved Measurement and Control Interface (MCI).

CONTROL SYSTEM AND SOFTWARE INTERFACES

There are quite some parameters that define environment in which the Control System operates. We can find heterogeneous instruments with different levels of complexity. Beside that the equipment is distributed over large remote regions and needs to provide reliable access regardless of the distance from the control room (see Fig. 1). Another characteristic of such operating environment is that the control is centralized, but the data acquisitions is distributed and to some extent also the data processing.

Based on that we can define interface requirements from the Control System's point that must cover following areas:
- device discovery, identification and capabilities
- operation mode control and configuration parameters
- events, alarms and health state monitoring
- data acquisition and attributes (data type, size, offset, time-stamp)

error handling

INSTRUMENT MANUFACTURER'S VIEW
From the reverse point of view, an instrument can be used in different environments (see Fig. 2). Requests for data can come from different sources for different purposes.

- Control System: Different types of control system protocols
- Other instruments: Instrument interoperability, multiple instruments working together, clustering, shared processing,
- Development Lab: Development, testing of new, updated instruments
- Maintenance: Diagnostics, repair

Not all of the access paths are active concurrently. A great deal of the information access has a common denominator, defined by the type of the information requested.

EMBEDDED COMPUTING

Using embedded computers in the instruments enables instruments to behave as network attached devices with built-in control system interfaces.

Embedded computer can be used to
- control the instrument's operation
- perform a part of digital signal processing
- provide remote access to the instrument

The embedded computer is one of the important components of an instrument, because it provides convenient way to bring all of the parts (hardware modules, FPGA, software) of an instrument together into a working application and perform certain digital signal processing.

Software running on the embedded computer can seen as one of the variable parts of a reconfigurable instrument.

RECONFIGURABLE INSTRUMENTS

Physical setup and behaviour of the instrument is not completely defined during manufacturing.
Modern trends in development of instrumentation encourage modularity with many standards for physical dimensions, electrical interconnectivity and data exchange protocols (see Fig. 3).

This leads to the following combinations:
- **Reuse of modules**: Hardware module MOD_A can be used in instrument INS_A, INS_B, …
- **Behaviour** of the hardware module MOD_A can be altered by loading different FPGA designs
- Instrument INS_A can comprise **variable number of modules** MOD_A, MOD_B, MOD_C, thus defining different **variations of the instrument**.

Design of the software, running on such an instrument, must be done in a way to recognise and make use of these combinations.

In general, the responsibilities of the instrument software can be split in several semi-independent layers: managing hardware platform, instrument application logic, external interfaces.

Hardware flexibility influences all of the software layers, including external interfaces.

**Semantic Types of Information**

The information transferred between the CS and the instrument can be divided into: digital signal acquisition, alarms (notifications), monitoring and control of the instrument state and behaviour (see Fig. 4).

Time considerations in the data transfers involves data rate and frequency. That is the time that is needed to transfer certain amount of data and the repetition speed how often that transfer happens.

Every data has its origin (data provider, source) and its destination (data consumer, sink). Depending on the active or passive involvement of either side in the data flow we can distinguish between data stream (data provider push) or data on demand (data consumer pull) as depicted in Table 1.

**Control System Programming Interface (CSPI)**

CSPI is available on **Platform A** type of instruments (Libera Briliance, Libera Briliance Single Pass, Libera Photon, Libera BunchByBunch). These instruments contain energy efficient ARM based embedded computer with limited computing power.

The operating system, used on the computer, is stripped-down distribution of Debian Linux, running on Linux kernel 2.6.20.

The computer is designated for proper operation of the hardware and FPGA from powering the box on to shutting it down and to provide network connectivity.

Hardware configuration of Platform A instruments is **defined at manufacturing**. Available data and the API are coupled together.

CSPI provides interfaces for:
- **Monitoring, controlling the instrument** through a number of parameters. They are all integer numbers and identified by numeric Ids. The set of parameters is fixed for a certain instrument.
- **Acquisition of the signals**. Functions to easily access pre-defined number of signals are available.

The goal of programming interfaces on both platforms is similar: implementation of as much functionality as possible in a common fashion and converting that information to a specific control system protocol as late as possible.

Both types of interfaces provide access to the semantic types of information described above (see Table 2).

**Table: 2**

<table>
<thead>
<tr>
<th>CSPI</th>
<th>MCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Networked API</td>
<td>Yes (Generic Server)</td>
</tr>
<tr>
<td>Number of signals</td>
<td>Dynamic (based on setup and processing)</td>
</tr>
<tr>
<td>Processing</td>
<td>Control loops</td>
</tr>
<tr>
<td></td>
<td>Control loops, more DSP algorithms (depends on application)</td>
</tr>
<tr>
<td>Configuration</td>
<td>Numeric identifiers</td>
</tr>
<tr>
<td></td>
<td>Registry (dynamic structure)</td>
</tr>
<tr>
<td>Notifications</td>
<td>CallBacks</td>
</tr>
<tr>
<td></td>
<td>Registry (built-in functionality)</td>
</tr>
<tr>
<td>DLI</td>
<td>Libera</td>
</tr>
<tr>
<td></td>
<td>Multiple</td>
</tr>
<tr>
<td>Control system interface</td>
<td>EPICS driver (two versions), Tango driver (3rd party)</td>
</tr>
<tr>
<td></td>
<td>MCI to EPICS adapter, more planned</td>
</tr>
<tr>
<td>OUI</td>
<td>EPICS EOM, Matlab, custom as implemented by CS integrators</td>
</tr>
<tr>
<td></td>
<td>QI, QI, EPICS EOM</td>
</tr>
</tbody>
</table>

**Table: 1**

<table>
<thead>
<tr>
<th>Data access</th>
<th>Streaming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pulled by user (on demand)</td>
<td>Pushed by instrument (on trigger)</td>
</tr>
<tr>
<td>Large</td>
<td>Small</td>
</tr>
<tr>
<td>Turn by turn, ADC</td>
<td>Slow acquisition, fast acquisition, events</td>
</tr>
</tbody>
</table>
• Change notifications. A callback function can be registered, which is called with the ID of the parameter that was modified.

Remote access is provided by:
• Generic server: transparent CSPI API access over the TCP/IP.
• Embedded EPICS driver: EPICS IOC driver that utilizes CSPI API. Alternative implementation was developed at Diamond Light Source that by-passes CSPI and communicates with the hardware in more direct fashion.
• Embedded Tango Server, developed by Elettra institute.
• External Tango Server, developed as a collaborative effort between Alba, Desy, Elettra, ESRF and Soleil institutes.
• External TINE Server, developed by Desy institute.

Measurement and Control Interface (MCI)

MCI is the interface of the Platform B instruments (Libera LLRF, Libera Brilliance+, Libera Single Pass H). Platform B instruments contain various types of i386-based embedded computers. These computers run standard Ubuntu Server edition (Linux kernel 2.6.26 or 2.6.32).

Dynamic nature of Platform B instrument required different design approach of the software and its API.

MCI has separated classes and functions of the API from the information that they are used to access. MCI is networked by design.

The following concepts have been introduced in the API:
• Registry: tree-structured representation of information, used to monitor and control parameters of an instrument.
  • The tree nodes are populated by the instrument software dynamically, depending on the hardware setup and type of the instrument
  • Nodes can emit notifications (for example: value change). Callbacks functions can be registered to nodes to receive those notifications

• Data Streams

Remote access is provided by
• Directly by MCI
• EPICS adapter: lightweight server without a database maps MCI registry and signals to EPICS PVs
• Tango, Tine adapter: will be developed when needed

Examples

Sample command line tool for reading the Libera unit environment parameters with CSPI

$ net-libera -i 10.0.0.100 -l
Temp [°C]: 45
Fans [rpm]: 4590 4560
Voltages [mV]: 1489 1782 2439 3233 4892 11865 -12020 -5089

Example of source code:

// Connect to the Libera unit at IP address 10.0.0.100
server_connect("10.0.0.100", 23271, "224.0.1.240", 0);
// Allocate the environment handle
epics_cspa_openhandle (CSPI_HANDLE_ENV, 0, henv);
// Prepare variables for environment parameter readout
CSPI_ENVPARAMS_params;
CSPI_BITMASK mask = -(0LL);
// Acquire the parameter
cspi_envgetparm (henv, params, mask);
// Release the environment handle
cspi_freehandle (CSPI_HANDLE_ENV, henv);
// Disconnect from the libera unit
server_disconnect ()

Structure of MCI registry as presented by a sample command line tool.

$ ./libera-ireg dump -h 10.0.3.40 -l 3
IP_10-0-3-40
boards
raf5
chassis:0
chassis:1
chassis:2
chassis:5
os

REFERENCES