LWDAQ Controller System Overview

Craig Dowell
University of Washington, Seattle

Version 0.0.1, 21 September 2006

Abstract

The LWDAQ controller is a process that controls the operation of an LWDAQ acquisifier and distributes step results. The execution of the LWDAQ controller process is itself directed by a PVSS panel. This panel receives step results and provides an almost real-time measurement event display; and also controls distribution of step results to a database logging program. This note provides a high-level description of the components in the LWDAQ Controller (LC) system and the communication between the components.

1. System Components

The overall system is shown in figure one. On the top right is an *LWDAQ* that controls the overall measurement process. The LWDAQ must be running its *System Server*. The System Server listens via TCP/IP to the well known port 1090 for connections and control commands.

The *LWDAQ Controller* (LC – top center in figure) takes *DIM Command* inputs and translates them into sequences of LWDAQ System Server commands.

The LC process always tells the LWDAQ to run its *Acquisifier Tool* and enables *Step Results*. Since Step Results are enabled, the Acquisifier Tool of the LWDAQ will send measurement results to LC over well known port 1091 as each measurement step is completed. When the LC receives a Step Result, it logs the result to a local file and also sends the result string via TCP/IP to the *Database Process* over a connection to port 4444. The Database Process talks to an Oracle database and saves each measurement result for future reference.

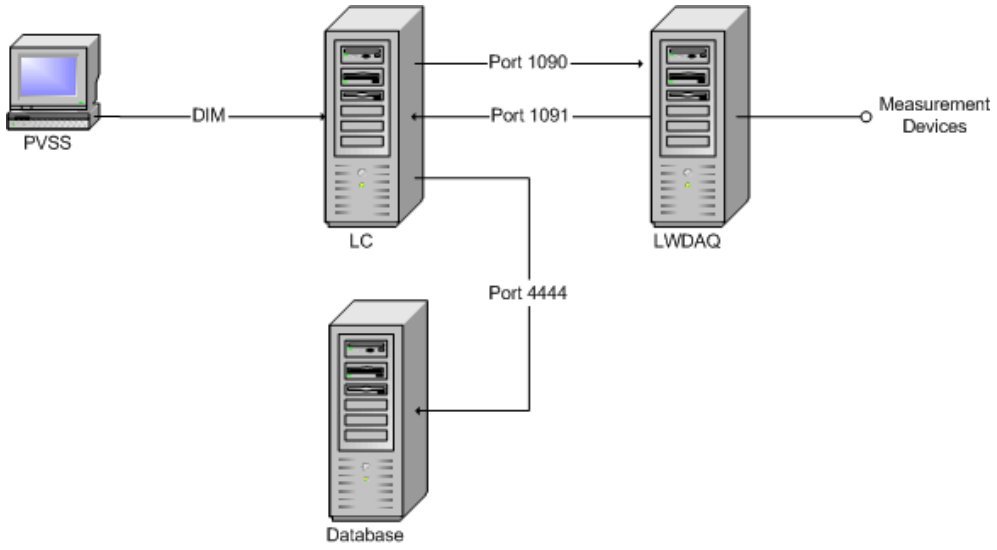

Fig. 1: System Diagram

At the far left in the figure is the *PVSS* system. PVSS is a *SCADA* system standard used at CERN for control and data acquisition. The PVSS panel provides a user interface for the system. CERN has provided a distributed communication mechanism called DIM which allows the PVSS panel to communicate with the LC in a straightforward way.

2. Communications

Figure Two is a UML diagram that shows the basic communications paths in the system. Beginning with the PVSS process (represented by the second block from the left), one can see that asynchronous commands are sent to the LWDAQ Controller (LC) for each user interface action. Most user interface commands are paired with asynchronous responses. The upper command / response pair between PVSS and LC is *Connect*. This particular command tells the LC to connect to an LWDAQ System Server on a given *Remote Host* using Port 1090.

When issuing commands, the PVSS panel acts as a DIM client and the LC acts as a DIM server. Strings are sent between the processes containing any parameter information. When receiving responses, the PVSS panel acts as a DIM server and the LC acts as a DIM client. Both the commands and responses are implemented as *DIM Commands*. The sense of client and server is reversed depending on the direction.

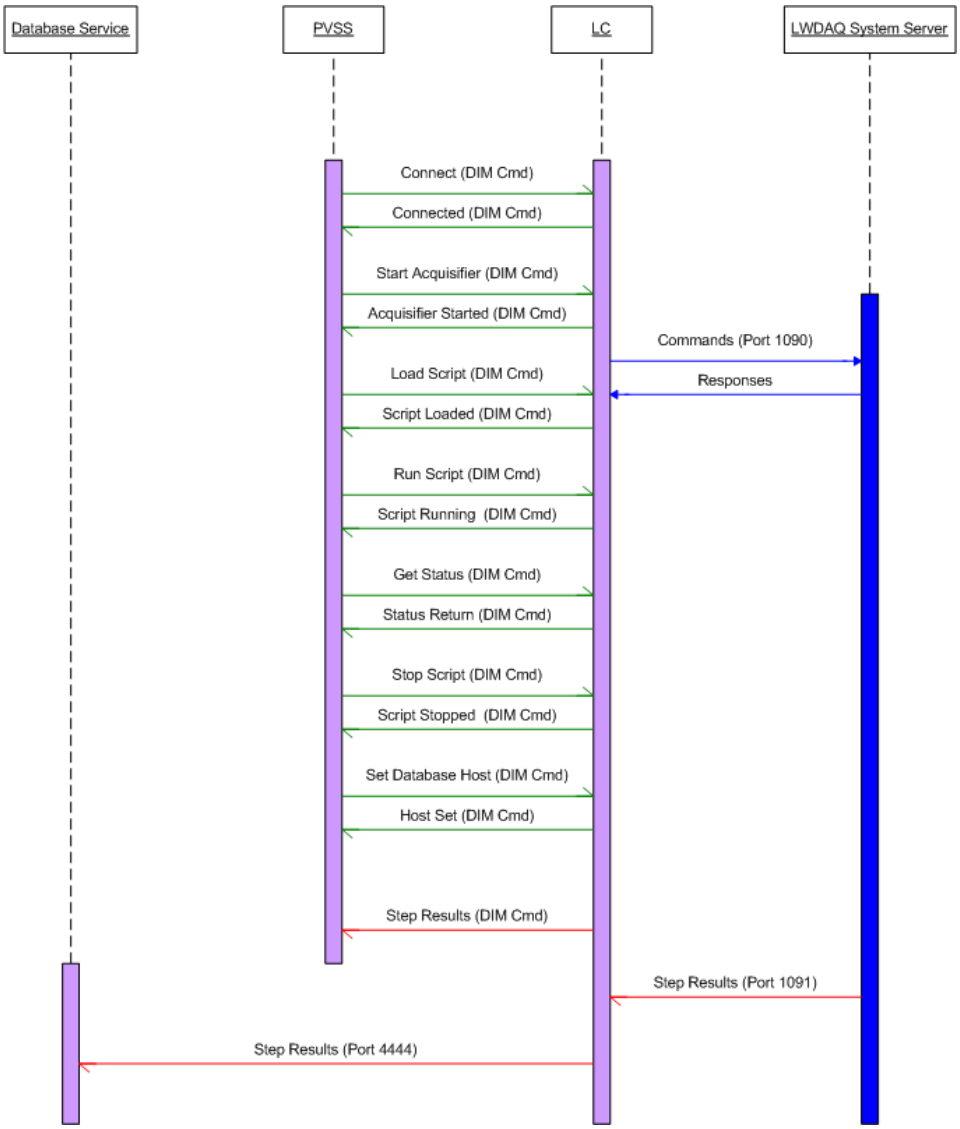

Fig. 2: Communications Diagram

In response to DIM Commands from the PVSS User Interface (UI), the LC issues zero or more commands to the LWDAQ. These commands are actually strings of characters that correspond to Tcl/Tk commands (the LWDAQ is implemented in Tcl/Tk). The strings are sent to the LWDAQ via a TCP/IP connection to the LWDAQ's System Server. Responses from the System Server are received over the same TCP/IP connection and are interpreted by the LC. Responses suitable for the UI are sent back to the PVSS panel using DIM commands (which the panel interprets as responses).

The LC implements a server thread which listens on well known socket 1091 for Step Results messages from the LWDAQ. As the LWDAQ is running a measurement script, it is set up to call back to the LC with the results of each script step. The LWDAQ will establish a TCP/IP connection with the LC, send the Step Results string and disconnect. It repeats this for each measurement result.

When the LWDAQ Controller Step Result Server receives a Step Result string, it will do three things:

1. Write the Step Results to a local file (Step_Results.txt);
2. Send the Step Result string to the PVSS panel via a DIM Command;
3. Send the Step Result string to the Database Service via TCP/IP.

The Database Service includes a TCP/IP Server that is accessed via well known port 4444. When it receives the Step Result string from the LWDAQ controller, it will write the result into an Oracle Relational Database.

3. LWDAQ Controller Internals

The LWDAQ Controller is a process coded for Unix (is actually runs on a Linux server). The software is organized into three main subsystems, *Main*, *DIM Side* and *TCP Side*. This organization is shown in Figure Three.

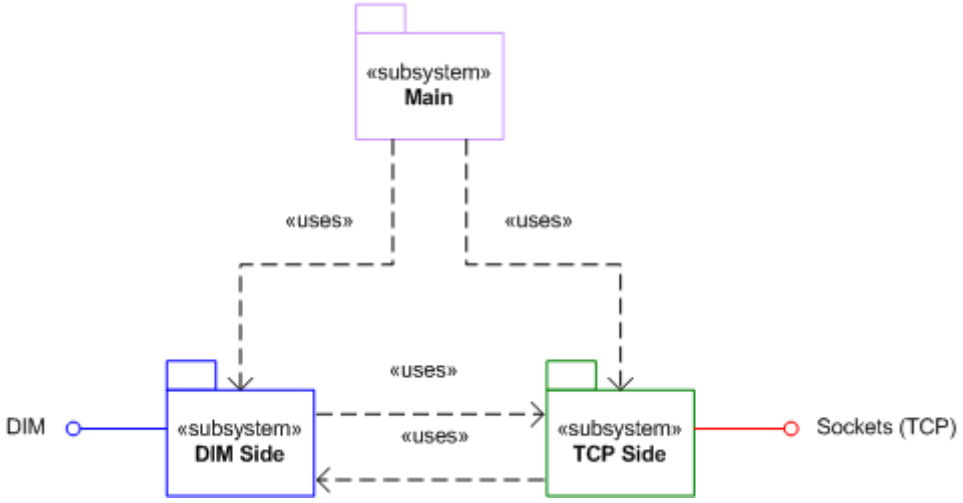

Fig. 3: LWDAQ Controller Subsystems

The Main module contains startup code and is very small.

The DIM Side module contains functions which deal with receiving DIM Commands and sending responses. DIM Commands are hooked directly into *Callback Functions* (using the *DIM Server Library*). These callbacks are implemented on the TCP Side. Most of the DIM Side functionality is in processing lists that associate incoming DIM Commands with TCP Side callbacks.

The bulk of the complexity of the LWDAQ Controller is found in the TCP Side module. There is a function defined on the TCP Side to process each of the PVSS panel UI commands. These *Handlers* translate each user interface action into some number of LWDAQ Commands and send those commands to the LWDAQ over the TCP/IP link. The handlers also call into the DIM Client to return results back to the PVSS panel.

Figure Three shows the basic organization is more detail. The DIM Side functions are on the left side (shown in a blue block if you have the color version of this document). There is a DIM Server subsection which makes calls to the DIM Server Library to associate the handler functions with DIM Commands (this is shown in the center of Figure Three -- in the green block if you have color). As DIM Commands are received, the corresponding command handler is called directly by DIM. In the course of processing a command, each handler will make some number of send() and recv() calls into the BSD Sockets library, shown on the right (in red), to communicate with the LWDAQ.

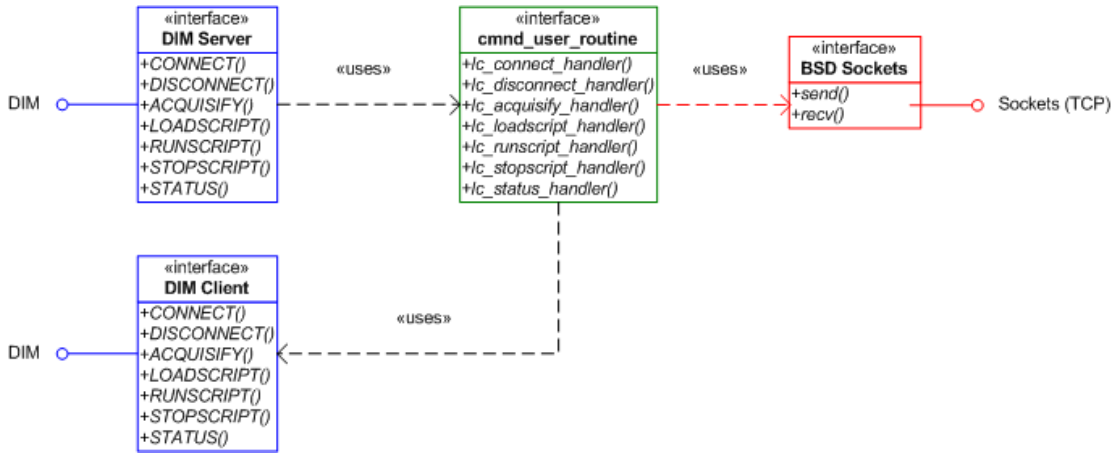

Fig. 3: LWDAQ Subsystem Detail

When each of the command handlers needs to send results back to the PVSS panel, it calls directly into the DIM Client Library to arrange for the results to be sent back to the panel.

There is also a server thread implemented on the TCP Side. It used BSD Sockets to listen on a well known Step Results port (1091). During startup, the host address and socket of the Step Results service is set to the LWDAQ. When Step Results are ready, the LWDAQ calls into the LC service and provides a string containing the results. This path is shown coming into the BSD Sockets section (right side – in red if you have color) of Figure Four.

Fig. 4: LWDAQ Step Results Service Detail

When the service has new step results lc_step_results_thread (center / green – located on the TCP Side) will use BSD Sockets to echo those results to the Database via its well

known port (4444). It will also call into the DIM Client Library (blue, to the left) to send the step results to the PVSS panel via the STEPRESULTS command.

4. Running the LWDAQ Controller

The LC process uses DIM dynamically loaded libraries and therefore has a prerequisite regarding the LD_LIBRARY_PATH environment variable. This path must include a reference to a DIM/linux directory that contains a libdim.so library. The LC is typically run from the `align` account on LXPLUS and this is done during the login processing. Additionally, in the align account the java environment required by the database program is also setup. After connecting to LXPLUS in the align account (all of the following examples assume the align account) you should see something like the following displayed after the usual LXPLUS banner:

```
Suceeded. JDK is now at /afs/cern.ch/sw/java/i386_redhat61/jdk/sun-
1.5.0_01
java version "1.5.0_01"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_01-b08)
Java HotSpot(TM) Client VM (build 1.5.0_01-b08, mixed mode)
PATH changed -- added . ~/software_dev/release
LD_LIBRARY_PATH changed -- added ~/software_dev/dim/linux
LD_LIBRARY_PATH changed  -- added ~/software_dev/release

[lxplus001] ~ >
```

To run the LC, you must change into the correct directory. This is done by executing the following command (i.e., the LC executable is found in the release directory `~align/software_dev/release`)

```
> cd software_dev/release
[lxplus001] ~/software_dev/release >
```

You type the text in **bold** (or **bold blue** if you have a color version of this note).

The simplest way to run the LC is using the defaults. This implies having an environment variable called `DIM_DNS_NODE` set to a hostname on which is running a DIM Name Server (DNS).

```
[lxplus001] ~/software_dev/release > lc
LWDAQ Controller Version 0.1.38 (Aug  1 2006 16:00:54)
LWDAQ Controller (pid 27567) starting up on DIM_DNS_NODE =
"pcatm19.cern.ch"
```

In this example, the LC is running, communicating with the DNS running on host "pcatm19.cern.ch". There will be no further output from the lc. It is possible to ask the lc program to print more information regarding connections and commands using the verbosity flag. A verbosity level is provided, with increasing levels corresponding to

increasing levels of detail about the program operation. For example, verbosity level one will cause printouts corresponding to a high level outline of the LC operation:

```
[lxplus001] ~/software_dev/release > lc -v 1
LWDAQ Controller Version 0.1.38 (Aug  1 2006 16:00:54)
giVerbose = 1
dis_add_cmd(LC_PROTOCOL/PING/COMMAND, C, NULL, 0xdefaced)
dis_add_cmd(LC_PROTOCOL/CONNECT/COMMAND, C, NULL, 0xdefaced)
dis_add_cmd(LC_PROTOCOL/DISCONNECT/COMMAND, C, NULL, 0xdefaced)
dis_add_cmd(LC_PROTOCOL/ACQUISIFY/COMMAND, C, NULL, 0xdefaced)
dis_add_cmd(LC_PROTOCOL/LOADSCRIPT/COMMAND, C, NULL, 0xdefaced)
dis_add_cmd(LC_PROTOCOL/RUNSCRIPT/COMMAND, C, NULL, 0xdefaced)
dis_add_cmd(LC_PROTOCOL/STOPSCRIPT/COMMAND, C, NULL, 0xdefaced)
dis_add_cmd(LC_PROTOCOL/ACQSTATUS/COMMAND, C, NULL, 0xdefaced)
dis_add_cmd(LC_PROTOCOL/SETFILE/COMMAND, C, NULL, 0xdefaced)
dis_add_cmd(LC_PROTOCOL/STEPRESULT/COMMAND, C, NULL, 0xdefaced)
dis_add_cmd(LC_PROTOCOL/SETDATABASEHOST/COMMAND, C, NULL,
0xdefaced)
LWDAQ Controller (pid 27681) starting up on DIM_DNS_NODE =
"pcatm19.cern.ch"
dis_start_serving()
```

In this example, the verbosity control flag is specified (-v) and level one is selected. This causes the LC to display the commands that it will export via DIM, and shows that the DIM server has been called and the LC is ready to start "serving" (processing commands). Additionally, when the commands are executed, messages are displayed on the LC showing that this has happened.

Verbosity level two corresponds to addition of the display of return values to the output of level one. For example, the output may show the values returned from the LWDAQ as responses to LC initiated commands. Verbosity level three adds internal LC function execution information to the amount of displayed information; and also begins displaying "ping" command and response information.