C^{3}PO

Bernhard W. Adams, April 23, 2016

I. INTRODUCTION

C^{3}PO (Control Code Collection for Process Organization) is a system of python scripts that interface industrial-control devices to a user. The python scripts can be accessed from the command line, through operating-system calls, or a GUI (if someone cares to write one). All implementation-specific information is contained in a PostgreSQL database.

Experiment control can be abstracted as 1) converting real-world items of interest to data (numeric, string, image, etc.), 2) processing the data in a program, and 3) converting data to a real-world effect. A good approach is to layer the control software into low-level drivers that interface with a hardware-abstracted (HA) control software, which allows scripting in any number of levels. Scripting can be done on the level of the operating system, i.e., though system calls. The interface between the low-level drivers and the lowest hardware-abstracted software needs to be standardized, yet flexible. In C^{3}PO, the real world is abstracted though process variables (PVs) that the lowest HA layer reads from and writes to. Data formats of PVs can be numeric, string, or vectors or matrices of these. When a PV is used (written to or read form), the lowest-level HA program looks up the PV name in the database to find the following information:

- name of the driver program, as known to the OS command line
- type: read/write/other. If an invalid operation is attempted, for example, writing to a read-only PV, then an error message is generated, and nothing more happens
- optional side effect: the name of an OS-command-line program that may do other things. This may sound rather harmless at first, but it may be used to ensure database consistency (see below), or for advanced operations, such as sending an email message if a motor is moved, or measured value exceeds a threshold

Drivers are python scripts that can be executed from the command line. Users should never access drivers directly, but only through the HA layer structure.

II. STRUCTURE

C^{3}PO has a layered structure with lower-level, hardware-specific, and higher-level user routines, all of which access a database for specific information, such as PV definitions. A schematic of the structure is shown in Fig. 1. Above the level of drivers lies hardware-abstraction layer 0 (HA0), which, currently, consists of only two scripts, pvw.py for writing to a PV and pvr.py for reading from a PV. The drivers and HA0 scripts reside in a dedicated directory called "C3PO" in this text. HA1 and above reside in a "user" directory. In the user directory, there is a file called "database", whcih contains a single line with the name of the database. By changing that name, a user can easily change the way that the user scripts interface with the rest of the system. It is recommended that users do not change the HA1 scripts in the user directory because these are the standard interfaces to the HA0 scripts in the C3PO directory. Instead, user scripts should be at HA level > 1, and should make system calls to the HA1 scripts.

The HA1 to HA0 interface between the user and system layers is constricted to a few HA1 scripts accessing just two HA0 scripts. This helps to minimize the setup effort, and it also directs the information flow: Complex operations, such as scans of actuators with detector readings at each scan point, etc., are reduced to writing to/reading from PVs. At that point, re-expansion occurs in another dimension, namely that of PV names.

The way that a user script interfaces with the real world of the experiment is as follows: The user script calls an HA1 script in the user directory and passes information (for example a target position of an actuator, or a request to read a voltage) to it via command-line parameters. That HA1 script determines a PV name or an alias for a PV name from the command-line information. Because PVs tend to have somewhat cumbersome names, it is often more convenient to use an alias, which is resolved by the HA0 scripts (see below). PV names or aliases may be passed to an HA1 scripts as a command line parameter, or they may be implicitly known to the HA1 script, depending on circumstances. For example, there may be multiple motors to move in an experiment, but only one detector to read. A script named ma to move an actuator (motor) to an absolute position would need the name of the motor as a command-line parameter, but in the case of the detector, the name would be unnessecary. The

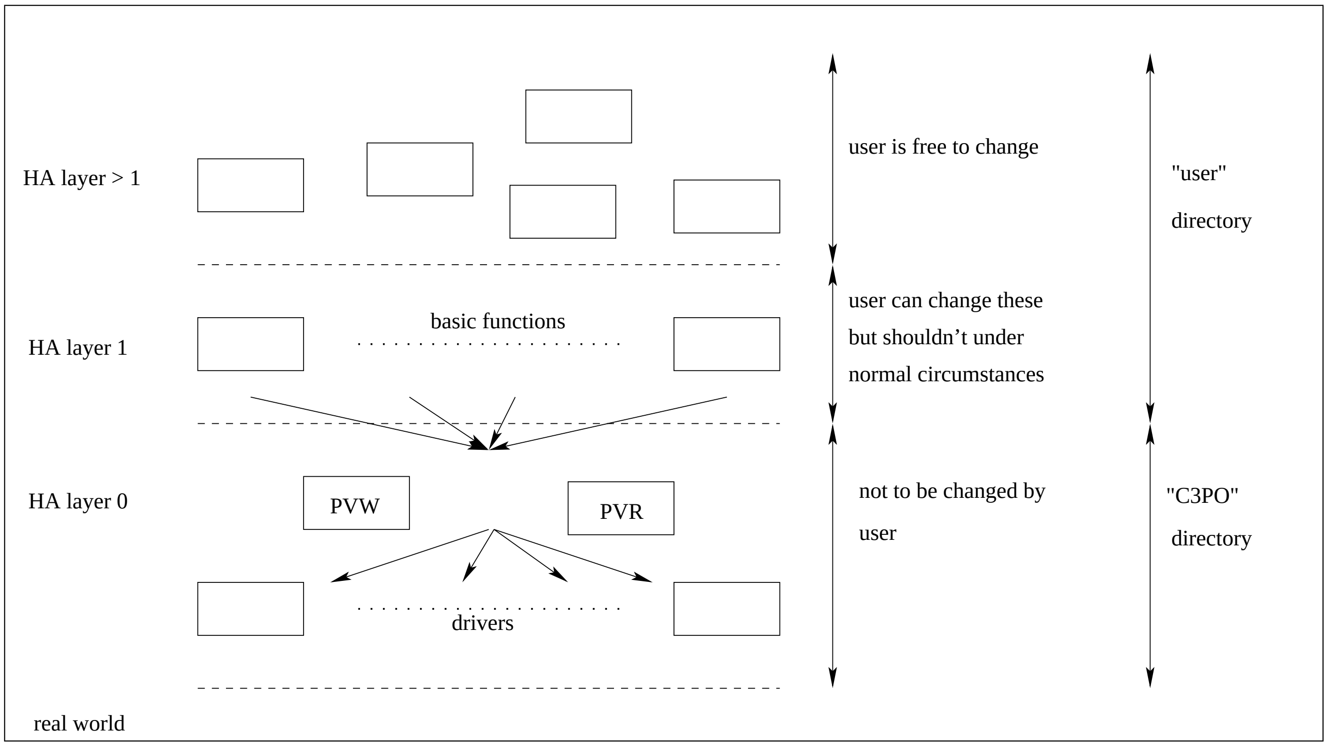

FIG. 1: Layered structure of C3PO. Higher-level functions call lower-level ones. All commands flow through the constriction of a few level-1 scripts calling only two level-0 scripts (see text).

HA1 script then calls the appropriate HA0 script. If something needs to be moved or changed, then it would call the script pvw.py with exactly two command-line paramters: the name (or alias) of the PV to write to, and the value to write. If a reading needs to be taken, then the script pvr is called with exactly one argument, then name or alias of the PV to read from, and the result is returned through the operating system. The HA0 scripts then try to resolve alias names by looking up the corresponding PV in a database table called "alias". Then, they consult a database table called "pvs" to find the driver to call for that PV, and finally call the driver with the PV name, and, in case of a write operation, the value to be written. Each driver has a database table where the PV name is also entered with further specific information, such as physical interfaces on the computer, etc.

III. DRIVERS

A driver program must accept at least one argument that tells it what to do, as well as additional optional arguments that depend on the requested function. All drivers must support the functions listed in Table. I.

1st argument	returns	format
ID	ID string	free
VER	version	major.minor
DATE	date the code was written	yyyy-mm-dd
AUT	code author	free
DBT	name of table in database	string
EXEC		string

TABLE I: Mandatory (above line) and typical (below line) functions that a device driver supports.

IV. THE DATABASE

The SQL (structured query language) database is a central component of the control-software suite. In a small-scale experiment, the database would usually reside on the same computer that is interfacing with the experiment hardware. However, it is possible to interface with a database on another computer on the network. The database is organized in tables of objects that are abstracted to have the same structure. The rows of the table contain the different objects (data items), and the columns the attributes that are relevant for the objects in the given context. The control program uses one mandatory tables One, called admin, contains some administrative information, most importantly the directory path where HA Level 0, and below, reside. The admin table has two columns: key and value.

The other mandatory table is called PVs. It lists all the PVs used in a given implementation. The columns in that table are:

- index no. (integer)
- PV name (string)
- type (string)
- name of driver program (string)
- execution code for driver (string)
- optional: name of side-effect program no. 1 (string)
- optional: name of side-effect program no. 2 (string)
- optional: name of side-effect program no. 3 (string)
- optional: name of side-effect program no. 4 (string)
- target value when last accessed (string)
- actual value when last accessed (string)
- optional: comment

Any additional information that may be needed to use the PV is contained in the secondary table, the name of which is passed to the driver, and which the driver knows how to use. The target and actual values when last accessed are mainly useful to hold motor positions. When a value is written to a PV, that value is always entered into the target value-field. The actual-value field may be populated from the return value of side-effect no. 1 (if present, see below). When a value is read from a PV, that value is always entered into both the target and actual value fields.

When a PV is accessed, the main HA program looks up the PV name and checks if the requested operation is compatible with the type (if not, it exits with an error message). Then, it looks up the name (to the OS) of the driver program, and passes the name of the PV, optionally the name of a secondary table where further information can be found, and the value (in case of a write operation) to the driver. The structure of the secondary table depends on the type of PV. It is usually defined when a new type of PV comes into use, possibly years after the initial installation. There are four columns for optinal side-effect programs. In the unlikely case that more side effects are needed, they must be handled through indirect calls, i.e., one of the four side effects calling more, secondary ones. Side effects may be needed for several reasons. For example, when a motor is moved to an absolute target position, there is no guarantee that it will actually reach it. Then, a side effect could be to call a program that reads back the actual value. This function is reserved for side effect no. 1

V. DATABASE DETAILS

A. Manual Access

The database can be accessed manually with the command `psql -d xcap`. It is helpful to do so now and again just to check on how the control software has modified the database. Databases that exist for the current user are listed with the command (on the psql prompt `xcap=>`) \d. The structure of table pvs is displayed with `\d pvs;`. All entries in table pvs can be shown with `SELECT * from pvs`.

B. Experiment Setup

To set up an experiment, one needs to create the database structure. This could be done manually on the postgrsql prompt, but it would be a very tedious process. Instead, one can run a setup python script. However, to alter an existing setup, for example, to set-up a new device, mayy be easier to do on the psql command prompt.

Entries into the table are made with the INSERT INTO command at the database command prompt. For example, `INSERT INTO pvs (pvname,type,drname,sectable) VALUES ('htr.mva', 'W', 'DCH', 'DCH');`

VI. INSTALLATION

System requirements are a contemporary Linux distribution with python and postgresql software installed. Pyhthon package psycopg (interface to postgresql) is also required.
To install postgresql:

```
sudo apt-get update
sudo apt-get install postgresql postgresql-contrib
```

or use a package-management tool such as synaptic.
To install python:

```
sudo apt-get install python python-psycopg2
```

Next, postgresql must be told to give access to the user. This is done by logging into the postgresql account, which should have been created automatically upon software installation:
log into postgresql account: `sudo -i -u postgres`
allow access: `createuser --interactive`

```
Enter name of role to add: <your user name>
Shall the new role be a superuser? (y/n) n
Shall the new role be allowed to create databases? (y/n) y
Shall the new role be allowed to create more new roles? (y/n) n
```

Log out of postgresql account `exit` Create database: `createdb <your database name>`

Next, the database must be populated with entries. There are several scripts in the C3PO directory that read the information from their specific data files and write it to the respective tables in the database. These are:

script	reads from	writes table	function
setup_PVs.py	pvsdefs	pvs	list PV names
setup_alias.py	aliasdefs	alias	create alias names
setup_admin.py	admindefs	admin	administrative stuff (path, etc.)

It may be helpful to enter a psql session to see interactively what is going on. Type at the command line: `psql -d <name of database>`
Then, at the psql prompt:
`\q` to exit
`\d` to list tables
`\d <name of table>` to list structure of a table
select * from <name of table> to list all entries in a table
The latter is very helpful to see the column names, which are needed to understand what the scripts are referring to.

VII. EXAMPLE

Suppose the experiment requires to drive three motors using the DCH motor controller from Dragonfly Devices, as well as reading images from a CCD camera, and controlling a temperature. The motor driver is connected to serial port /dev/ttyUSB0, the camera requires a call to a program that handles exposure, readout, etc., and the temperature controller is connected by RS485 bus to an ethernet-serial bridge. Assume the motors are called `htr`, `vtr`, and `foc`, and we use a somewhat structured naming convention for the pVs (this is not necessary, but it is helpful). For each motor, we want to have PVs to send it to an absolute position, to a relative position, read back the current position, and read back the limit-switch status (many more PVs may be desirable).

Then, for each motor, one would require at least PVs to send it to an absolute position