BRINGING THE POWER OF DYNAMIC LANGUAGES TO HARDWARE CONTROL SYSTEMS[*]

J.M. Caicedo[†], R. Stoica, N. Neufeld, CERN, Geneva, Switzerland

Abstract

Hardware control systems are normally programmed using high-performance languages like C or C++ and increasingly also Java. All these languages are strongly typed and compiled which brings usually good performance but at the cost of a longer development and testing cycle and the need for more programming expertise.

Dynamic languages which were long thought to be too slow or not powerful enough for control purposes are, thanks to modern powerful computers and advanced implementation techniques, fast enough for many of these tasks.

We present examples from the LHCb Experiment Control System (ECS), which is based on a commercial SCADA software. We have successfully used Python to integrate hardware devices into the ECS. We present the necessary lightweight middle-ware we have developed, including examples for controlling hardware and software devices. We also discuss the development cycle, tools used and compare the effort to traditional solutions.

INTRODUCTION

Integrating new devices into hardware control systems is frequent task that is usually quite complicated. The devices support different protocols for accessing the interfaces; sometimes they are standard and widely available protocols, such as SNMP or web services, but often they provide a proprietary and specific mechanism, e.g. Telnet access, which is more difficult to automatize.

Writing the communication layer between the device and the control system requires experimentation and prototyping. Frequently it is necessary to actually test and debug the program to understand completely the behavior of the device. In our experience, this task represents the larger fraction of the total time of the project.

Programming languages and tools contribute to make the task easier (or harder). Traditionally, hardware control systems use high performance languages like C, C++ and Java; with these languages, changing a small part of the code may imply several minutes of wait before seeing the effect of the modification. In these cases is desirable to have tools to support a more interactive style of programming. Finding a good balance between a rapid development cycle and performance would allow us to integrate more easily new components into hardware control systems.

Furthermore, in a large control system many programmers contribute to the code base and they also change over time. In particular in science, control systems tend to constantly evolve so it is important to keep the applications simple and improve the maintainability.

We describe in this paper how we have successfully used dynamic programming languages to implement different parts of the ECS of the LHCb experiment at CERN and overcome these difficulties. We present performance benchmarks and we include a case of study of the implementation of a program for remotely controlling electric power sockets using Python.

ECS SOFTWARE

Control Systems are used for operating, configuring, and monitoring the devices comprised by the system. For example, the LHCb ECS is in charge of the equipment of all the areas of the online system, like data acquisition, experiment infrastructure and controls [1].

Naturally, the devices used are heterogeneous and integrating them into an uniform control system requires dealing with multiple variables, such as the differences among hardware architectures and communication protocols. In the case of CERN, the four LHC experiments have adopted the Joint Controls Project (JCOP), which defines an architecture for the control systems, along with a framework and the tools for developing new components. This architecture is based on a commercial SCADA software (PVSS II).

DYNAMIC LANGUAGES

The term *dynamic languages* is used frequently to describe a category of programming languages that support *interactive* or *exploratory programming*. However, since this term does not have a precise definition, we select the following as the main features of the languages that allow this type of programming style:

- Programs are interpreted and most of the verifications, such as type checks, take place at runtime instead of doing them before the execution, i.e. at compile time.
- Programs have access to internal information of the runtime environment and can change it as they run (e.g. add or modify definitions).
- Languages provide high level instructions and data structures and simplify programs by dealing automatically with complex tasks, such as memory management.

[*] This research project has been supported by a Marie Curie Early Stage Research Training Fellowship of the European Community's Sixth and Seventh Framework Programme under contract numbers MEST-CT-2005-020216-ELACCO.

[†] e-mail: juan.manuel.caicedo.carvajal@cern.ch

The main benefits of dynamic languages are easy of development, an increase in the productivity and flexibility. These languages provide an environment where interactive programming is possible, so development cycle can be shorter because of the ability of easily making modifications and receiving immediate feedback to all the operations, instead of having to wait for the edit - compile - run cycle. This style of programming can be more efficient because encourages incremental development and allows the programmers to test quickly new ideas for better understanding the problem.

By providing higher-level instructions and data types is possible to perform more operations using fewer lines of code, thus speeding up the development process. Additionally, dynamic languages are well suited for integrating systems, where is necessary to combine components of different applications. For example, [2] compares this type of applications implemented in different languages and shows how the ones that used dynamic languages were shorter and written with less effort. Finally, with dynamic languages programs can be more expressive and flexible because of the possibility of modifying at runtime the type system and the definitions or adding code during the program execution.

Python is a popular dynamic programming language that emphasizes in the readability of the code and aims to improve the efficiency of programmers by providing an easy to learn syntax and high level control and data structures [3]. Python mainly supports object oriented programming, but also other paradigms: functional and imperative programming. Its implementations include a comprehensive standard library and they can be extended with modules written in other languages such as C and C++.

The Python language is powerful and is already widely used in many sectors, including scientific computing. Furthermore, it has been the language of choice for many applications in the LHCb, including systems administration and data analysis tools, so we chose to also use Python in the LHCb control system.

PYTHON IN ECS

Python is integrated in a relative low level of the LHCb ECS: between the devices and the SCADA system that provides the user interface where the users can view and control the device state. The communication layer is provided by DIM [4]: a middle-ware for information publishing, data transfer and interprocess communication. We have developed a module for using this middleware from Python programs.

DIM – Distributed Information Management System

DIM provides a reliable and efficient layer for communication between processes in heterogeneous environments. It is based on the client-server architecture, where the servers provide data as the form of *services* and the clients access subscribes to them. In addition to services, DIM servers can also offer *commands*, which are routines that are executed on the server when a client invokes them supplying values as their arguments.

Clients can decide to request a service only once or at a regular intervals, but also can receive the data whenever they are updated by the servers. This mechanism for updates allows writing client applications that respond asynchronously to the data and also brings the possibility to have multiple clients subscribed to the same service and to receive the updates in parallel.

Another element of DIM is a *name server* which keeps track of the services and commands published by the servers. Clients ask to the name server which server provides the service they want to connect to and then they establish direct connection with it.

Reliable communication in DIM is achieved by automatic recovery from errors, such as network and server failures. In these cases, a client tries to re-establish the connection and waits until the services are available again.

DIM is available for multiple platforms and machine architectures. The system handles transparently the communication issues, such as differences of data representation between platforms.

PyDIM - Integrating DIM with Python

PyDIM is an extension for accessing DIM from Python. The extension is a wrapper around the DIM libraries for C and it provides a simple application programming interface (API) which is easy to learn for newcomers but also for users with experience in developing with DIM using C.

PyDIM aims to maintain the simplicity of Python and to keep all the features of DIM, such as reliable communications and portability. The extension deals transparently with memory allocation and type conversions of the sent and received data, so Python programs can keep using their native data types. Clients and servers implemented with PyDIM are entirely compatible with the ones implemented with other languages. Complete documentation of PyDIM, including code examples, is available in its website. [5]

Performance

We compare the performance of PyDIM in respect of DIM to measure the overhead of Python and the extension module. We performed three tests for measuring the latency, memory usage and throughput of DIM clients and servers written with Python and C/C++.

We performed the benchmarks on a single machine running the DIM client, server and name server so that we could minimize as much as possible the effect of external factors. The machine used was a Intel Xeon CPU server with 8 cores, 8 GB of memory and running the SLC 4 Linux operating system.

In all the tests the performance of the Python programs was lower. The overhead of the interpretation and the dy-

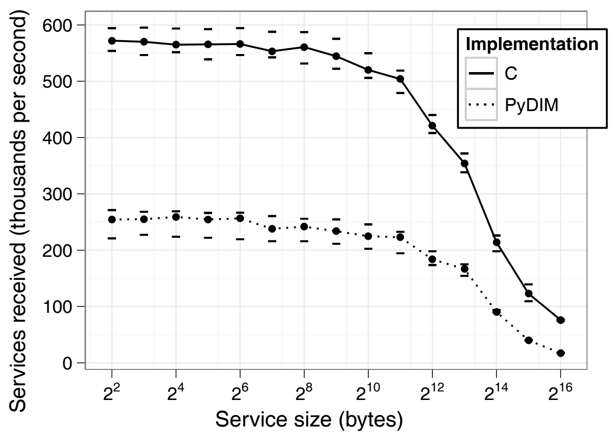

Figure 1: Services received per second for different data sizes

namic features of the language imply a cost in running time and in memory usage, even if the PyDIM module is written in C and calls directly the DIM library.

The latency was measured as the time between the moment where a client requests a service and receives the value from the server. For Python programs the mean was 1 980 µs and 996 µs for C programs (the standard deviations were 360 µs and 195 µs, respectively). This difference is due mainly to the conversion of the data received to Python objects which is performed by the wrapper library. As for memory, we measured the amount used only by each process (excluding the shared memory). This test was performed both for clients and servers using 25 services. The maximum memory used by the client was 372 kB with the C program and 2 256 kB with Python. On the other hand, the C server used 472 kB, whereas the maximum memory usage of the Python program was 2 416 kB.

We measure the throughput as the number of updates of a service that a client can receive during a certain time. This test was performed several times increasing the size of the value published in the service in order to measure the differences when the amount of data changes. The Python program had a lower throughput, but the variation of the size of the message had a similar effect in both implementations. Figure 1 shows the results of this test.

Case Study: Powersocket

We developed a simple program for integrating remote power management devices into the LHCb ECS. These devices are used for switching on and off a power outlet using SNMP or a Telnet interface.

The program is divided in two parts: one for controlling the device using the interface provided by the manufacturer, and other for accessing and manipulating this information via DIM. The two components can run and be tested independently, such that maintenance and debugging of the system can be made more easily.

Using Python was an advantage, not only because it allowed a rapid development cycle, but also because the availability of libraries speed up the process. For example, an early prototype used a library for automating the interaction using the Telnet interface of the device.

The code and the documentation of this project is available online. [6]

CONCLUSIONS

Dynamic programming languages can provide a shorter development cycle for integrating devices into hardware control systems. The time for writing the program can be reduced by using *interactive programming*, and the resulting code is usually short and simple to read and maintain. The advantage of dynamic programming language is also recognized by other control systems. For example, EPICS [7] and TANGO [8] provide bindings for accessing the system API from Python programs.

PyDIM allow us to write components of a control system using Python, so we can exploit the features of DIM: efficient and reliable inter process communication. The use of a uniform communication framework like DIM is necessary for a heterogeneous environment like the LHCb ECS.

The performance cost that comes with the dynamic features of the language is compensated by the powerful hardware used nowadays, the new implementation techniques or by using a mixed language approach, when the computationally intensive tasks are written in a language suited for systems programming, like C, and then are included as modules the can be used from the rest of the program written in a dynamic language.

REFERENCES

[1] C. Gaspar, B.J. Franek, R. Jacobsson, B. Jost, S. Morlini, N. Neufeld, P. Vannerem, "An integrated experiment control system, architecture, and benefits: The LHCb approach", *IEEE Trans. Nucl. Sci.*, 51:513-520, 2004

[2] J.K. Ousterhout, "Scripting: Higher-Level Programming for the 21st Century", *Computer*, 31(3):23-30, 1998

[3] Python Programming Language, `http://python.org/`

[4] C. Gaspar, P. Charpentier, M. Dönszelmann, "DIM, a Portable, Light Weight Package for Information Publishing, Data Transfer and Inter-process Communication", *Comput. Phys. Commun.*, 140(1-2):102-9, 2001, `http://dim.web.cern.ch/dim/papers/chep/dim.pdf`

[5] `http://lbdoc.cern.ch/pydim/`

[6] `http://lbdoc.cern.ch/powersocket`

[7] EPICS – Experimental Physics and Industrial Control System, `http://www.aps.anl.gov/epics/`

[8] TANGO Controls System, `http://www.tango-controls.org/`