PVMANAGER: A JAVA LIBRARY FOR REAL-TIME DATA PROCESSING

G. Carcassi, K. Shroff# NSLSII, Upton, NY, USA

Abstract

Increasingly becoming the standard connection layer in Control System Studio [1], pvmanager [2] is a Java library that allows creating well behaved applications that process real time data, such as the one coming from a control system. It takes care of the caching, queuing, rate decoupling and throttling, connection sharing, data aggregation and all the other details needed to make an application robust. Its fluent API allows specifying the details for each pipeline declaratively in a compact way.

INTRODUCTION

Fig. 2 shows the general architecture of the NSLS2 control system environment, a common problem encountered by client applications for control systems is the decoupling of the events from the controls network and the UI thread. The need to aggregate the events in time (for rapidly changing pv/pvs) and for groups of pvs was necessary to address various performance issues in CS-Studio and to support multi-channel applications.

The goal of pvManager is to make writing clients for real-time data more straight-forward, by providing all the pieces that such a client require, such as data rate decoupling, via either queuing or caching, data aggregation/manipulation and notification dispatch on the appropriate final thread.

ARCHITECTURE

The initial intent of pvmanager was just to address the recurring issues of writing a well behaved client of a soft real-time system. The aim has now grown to provide a full end-to-end framework for gathering data from different sources, both publish/subscribe and command/response, aggregating it and performing computation on background threads [3].

The framework now consist of multiple modules:

- vtype: provides the definition in terms of Java interfaces of a standardized set of data. One is not limited to the use of these types (the basic type in pvmanager is Object) but standardization on them allows to unlock all the functionality already implemented
- datasources: provides support for accessing data from publish/subscribe systems
- services: provides support for accessing data from publish/subscribe systems
- formula: provides a pluggable Domain Specific Language for aggregation and computation

shroffk@bnl.gov

The core of pvmanager allows combining all these elements and creating readers or writers that are thread-safe with a managed rate of notification.

CS-Studio can now leverage all these elements, but, since they are well separated, they can be tested without the UI environment (unit tests are much easier to write) and can be used in other environments (such as plain Swing applications, command line, web servers, and so on).

DATASOURCES

Datasources are the abstract definition for publish/subscribe data, which is the typical mode for real-time systems, such as EPICS. Datasources work on channel, and are able to subscribe readers or writers to each channel. Current implementations include support for simulated signals, an in memory scratch space, filesystem, Channel Access (v3) and PVAccess (v4).

The system can be easily extended with other types. All one needs to do is implement a few abstract methods, and connect the callback of the desired system to the methods that trigger processing of connection and message notifications. Datasource automatically provide support for multiplexing (multiple readers on the same channel). The rate decoupling (limit the rate from the datasource to the UI subsystem) and rate throttling (decrease the rate if the UI can't keep up) that are needed for a well behaved client, are also automatically supported.

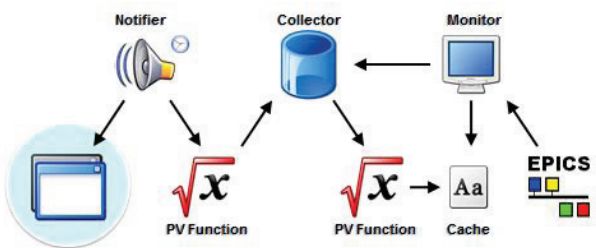

Figure 1: Architecture diagram. Shows the part of the client that runs at the rate dictated by the source of the data on the right and the part of the client that runs at the rate at a client dependent rate.

ISBN 978-3-95450-139-7

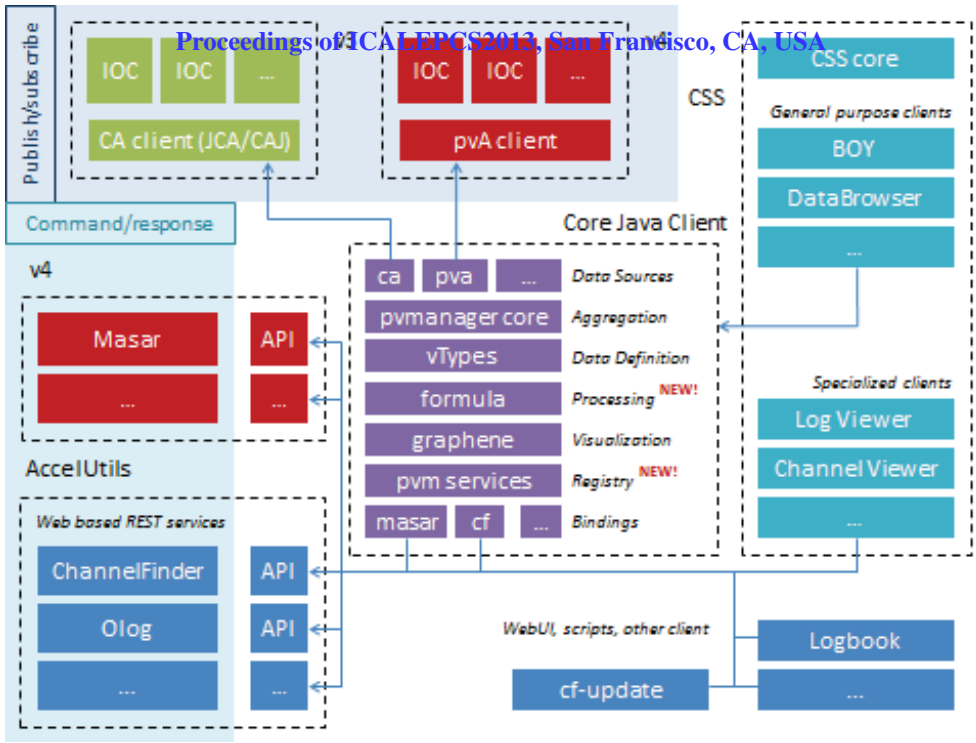

Figure 2: Overall Architecture of accelerator tools and services.

In Fig. 1 we can see the basic layout of the architecture and the basic building blocks. On the right side we see the DataSource (EPICS in this case) which is going to generate a series of events. In the middle we see the collector, which is the piece responsible to decouple the data rate. On one side it will receive events at the data source rate; on the other side it will receive requests for data at the desired rate. A different implementation of the Collector will be required in different cases (e.g. a cache, a queue or a cache for the last n seconds). On the left side of the diagram we see the client, receiving notifications on the requested thread at the rate desired. By desired rate we mean the maximum rate at which it makes sense for the client to receive notifications. For example, when displaying something on screen (e.g. a plot or an indicator) updates faster than 50 Hz make little sense. If displaying text that needs to be read, updates faster than 5 Hz may already not give the time to read. When writing to a database or saving to disks, one may want to batch the updates at 1 Hz, increasing the throughput.

Though this kind of rate decoupling is always required when writing a good client, it is very tedious and error prone to re-implement the whole stack for a new applications. Getting the synchronization right, checking for correct behaviour during high stress of the system, sizing the different parameters of the system, all these activities that a significant amount of time to get right. With pvManager we provide all the components already done and tested, so that one can put more time in the actual new feature of the client application instead of in yet another pipeline.

SERVICES

Command/response is another typical source of data. Examples of command/response include web services (REST or SOAP), CORBA services and databases. The abstraction for a service within the framework is an asynchronous call that takes a key/value map of argument, and returns a key/value map of results. This allows to generically handle very different types of data and services. The value, if possible, should map to VTypes so that one can reuse many of the functionality and clients built on top of those.

The framework also includes a ServiceRegistry, which works as a locator for the services. A typical use case in CS-Studio is that some plug-ins would register their service implementation to the registry, while UI elements would use the registry to fetch the service implementation given a name provided by the user.

General purpose implementations of services are also possible. We have a generic JDBC service implementation, which, given an xml with location of the server and queries, is able to dynamically generate a service implementation. A similar generic implementation is being written for PVAccess services.

FORMULA

Pvmanager provides a Domain Specific Language for data computation. This is automatically done on background threads, making it easy to leave the UI thread free. FormulaFunctions can be dynamically added in a formula registry, in much the same way that services are added. They are automatically picked up by the parser.

The language supports overriding, and the match to the correct signature is done at runtime. This also supports the case in which types within the expression are changing dynamically. Support for standard mathematical operation has already been added, as well as aggregation such as scalars into array and arrays into tables, and other utility functions (pick the highest alarm, “pointer-like”

ISBN 978-3-95450-139-7

function that given a string returns the value of the channel with that name).

CONCLUSION

Pvmanager has grown into a full framework to gather data in real time, aggregate it and perform computation. All this while taking care of the issues that such a complicated multi-threaded system would entail.

REFERENCES

[1] Control System Studio; http://controlsystemstudio.github.com
[2] http://pvmanager.sourceforge.net/
[3] G. Carcassi, Pvmanager & Graphene, EPICS spring meeting (2013)

ISBN 978-3-95450-139-7