From toolkit to framework - the past and future evolution of PhEDEx

A Sanchez-Hernandez,[a] R Egeland[b] [1], C-H Huang[c], N Ratnikova[d], N Magini[e] and T Wildish[f]

[a] Centro Invest. Estudios Avanz IPN
[b] University of Minnesota, Twin Cities
[c] Fermi National Accelerator Laboratory
[d] KIT (Karlsruhe Institute of Technology) and ITEP (Institute for Theoretical and Experimental Physics, Moscow, Russia)
[e] CERN
[f] Princeton University

E-mail: `awildish@princeton.edu`

Abstract. PhEDEx is the data-movement solution for CMS at the LHC. Created in 2004, it is now one of the longest-lived components of the CMS dataflow/workflow world. As such, it has undergone significant evolution over time, and continues to evolve today, despite being a fully mature system. Originally a toolkit of agents and utilities dedicated to specific tasks, it is becoming a more open framework that can be used in several ways, both within and beyond its original problem domain. In this talk we describe how a combination of refactoring and adoption of new technologies that have become available over the years have made PhEDEx more flexible, maintainable, and scaleable.

1. Introduction

PhEDEx[1] (Physics Experiment Data Export) is the data-placement management system developed by CMS for managing the experiments' data on the grid. It uses grid-level transfer tools (FTS[2], SRM[3], etc) to transfer files according to the CMS data placement policy. It was originally implemented in 2004, and consists of a set of agents and a few ancilliary command-line tools, and a website for presenting information to users.

The agents are used for repetitive tasks, scheduling them at regular intervals, such as scheduling transfers according to priorities and link performance, processing the transfer queues for a site, and updating the bookkeeping based on recent activity. Each agent performs a single task, a set of agents co-operate to perform the complete functionality needed at a given site or for the central management of PhEDEx. Agents are Perl programs which contact an Oracle database to find out about their workload. They perform their work, and update the database with their results. Agents do not directly communicate with each other, instead they communicate through the state they maintain in the database.

The command-line tools are used for occasional or unscheduled activities, such as controlling the set of agents at a site, or injecting data into PhEDEx. Many of the command-line tools have

[1] now at Montana State University

been replaced with access through a web-based API via the PhEDEx data-service[4], though a few are still in use.

The website is a Perl CGI program which was written and maintained separately from the agents. It provides high-level views of the state of PhEDEx activities, with options for filtering to select only the items of interest.

2. The problem

Agents are optimised for the individual task that they perform. Like the command-line tools, they use direct database connections to perform their work. SQL statements were coded directly into the agent, and there was little sharing of code between agents beyond some low-level utility libraries. Writing and tuning an agent can require a considerable degree of expertise, given that site agents have to perform well with a variety of site configurations and scales, while the central agents need to have ever-increasing scalability to cope with the increasing volumes of data. At the time PhEDEx was originally put into production, there was no easy to use open-source framework for event-driven programming in Perl, so much of the internals of the workflow had to be coded by hand.

As a result of all this, PhEDEx was written as a collection of specialised tools. Developing a new agent was more complex than it might otherwise be the case, and was considered a task for an expert.

Some site-responsibles wrote their own tools for monitoring, but these were often specific to their own site and not easily shareable. So they were either destined to be re-written at other sites, thereby wasting manpower, or others would simply not be able to benefit from those tools.

The website was produced independently from the agents, with only the low-level utility libraries shared between them. Many of the SQL queries used by the website are duplicated from the agents or command-line tools, with the incumbent risk that changes over time cause them to drift apart, until the information presented to the user no longer accurately reflects reality.

As PhEDEx has grown in scale and scope, the number of users has increased. CMS now has over 100 nodes registered in PhEDEx, with as many site agents and site operators. People from physicists to computing management use PhEDEx on a daily basis to understand what is happening to CMS data. Even the experienced operators need better access to information, as the problems they tackle change over time because of the growing volume of data. Many of these people want to access data about PhEDEx from their own scripts and tools, so they can produce their own summaries of the things that interest them.

The overhead associated with the old way of producing agents and tools to perform tasks and provide information would be too much to support such a growing community. For new developers, many of whom are not working full-time on PhEDEx, the cost of getting up to speed on the software would delay their contributing to the project.

3. The solution

Already some years ago, PhEDEx provided a web-based data-service[4] for accessing information about the internals of PhEDEx. Data is available in a variety of formats, and detailed filtering is available to allow the user to select exactly what they want to see. This goes a long way to addressing the problem, but is not in itself a complete solution.

The path to a complete solution is to turn PhEDEx from a dedicated toolkit into a set of frameworks, so that people who want to solve a particular problem can easily produce their own tools for the job. These tools would then be more readily shared among sites, generalised for the whole community, faster to develop, and easier to maintain. This has been an evolutionary process, performed in many steps, and is still ongoing today.

3.1. The core agent framework and AgentLite

The first step was to refactor the core agents to allow re-use of as much code and SQL as possible. Each agent was converted into a set of modules; one to provide functions to wrap the SQL statements, one for the core functionality (the specific task of that agent), and one to provide the interface to the core modules. As much common behaviour as possible was extracted from the agents, into the core modules, to ensure consistent behaviour and eliminate redundancy.

Providing functions to wrap the SQL statements, and putting them in their own package, makes it trivial to re-use an SQL statement originally written for one agent elsewhere, in others. They need only load the agents SQL package and call the function with the appropriate arguments. Reducing the number of variations on semantically equivalent SQL statements has the extra benefit of improving the performance of the Oracle statement cache by eliminating redundant variations of SQL.

Much of the truly generic SQL was factored out into a common package, shared by all agents. This allows us to perform further optimisations, such as caching results which are not expected to change often. The list of nodes known to PhEDEx is a good example, or the detailed mapping which translates a logical file name (LFN) into a physical file name (PFN) for a given storage. The agents can then simply call the common function as they want, and the core package provides caching to reduce the load on the database and increase the performance of the agent.

At the same time, we adopted an open-source framework for managing the agents. The Perl Object Environment (POE[5]) is an event-driven framework for co-operative multi-tasking. Being co-operative instead of pre-emptive has many advantages in our application. We do not need to worry about resource-locks or race conditions, they simply do not occur. We need only ensure that each event-handler yields the CPU in a timely manner to allow other tasks to be performed. Since none of our tasks are real-time critical, this suits us perfectly.

It also means that we can readily incorporate other threads of activity, such as monitoring and reporting, updating of statistics, checking for exit conditions, or anything else, as a completely separate code thread. Because the multi-tasking is co-operative, none of the threads need to know anything about the others, and do not need extra protection from each other. The agent-code is much simplified as a result.

More recently, we have begun a second re-factorisation of the core agent functionality, which we call 'AgentLite'. This allows us to split out the PhEDEx-specific parts common to all agents from the non PhEDEx-specific parts. This will allow us (and others, should they wish) to create agents that are not tied to PhEDEx behaviour, but can benefit from the body of code we have built up over the years.

3.2. The Namespace framework

The Namespace framework[6] is designed to provide an interface between PhEDEx-specific activities and storage element (SE) optimisations (figure 1). There are tasks that we wish to perform regularly, such as checking the consistency between files on an SE and files known to PhEDEx. The PhEDEx developers do not have easy access to all available storage technologies, nor do we have the expertise that site managers will have to be able to write optimised software for any given technology. However, we do know how to write the agents and tools that can perform the work that we want done, and in such as way as to perform optimally on the database.

Before the Namespace framework was developed, sites would typically write their own scripts, optimised for their own conditions, and therefore hard to re-use elsewhere. Either they wouldn't work for someone else without considerable re-writing, or it was hard to maintain a single version that worked well across multiple sites. The Namespace framework solves this problem by separating the SE access from the work being performed. PhEDEx experts can write a generic tool to perform a given task, and provide a plug-in module to the Namespace framework

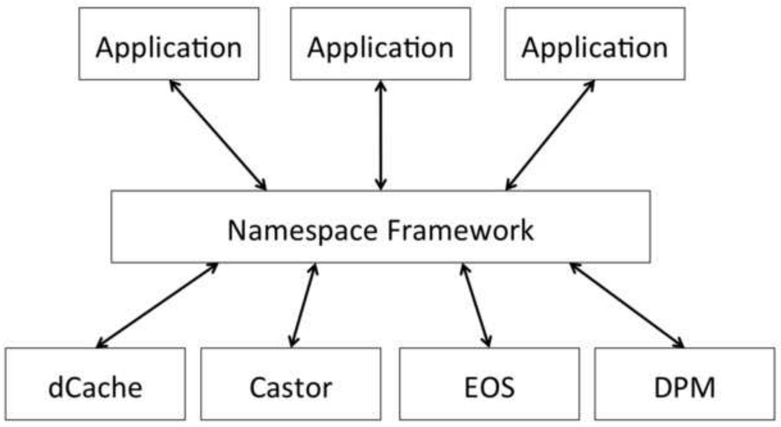

Figure 1. The Namespace framework separates application logic from specific details of storage element implementations.

to perform the SE-specific parts (access to information about files and directories etc).

For another site, with a different SE technology, it is sufficient for the responsible people to implement a different plug-in, optimised for their own circumstances. The same tool can then load the alternate plug-in and perform the same task, optimised for each of the two sites. Neither site has to maintain code for the other, they need only conform to the API specifications of the framework.

The framework is designed to be minimalist on the SE-side, so site experts should not have to write large, complex amounts of code. Essentially, access to any property of a file, such as its size, it's access permissions, migration status and so on, is mapped from a standard name in the Namespace to a single function or set of functions in the SE-specific part. The SE-specific part can execute arbitrary commands to access the information, and can optimise access to the storage element in the process.

For example, when asking for the size of a file, if the size were obtained by the system *stat* command, the SE-specific part can get the size and access permissions together, in the same call, and cache both properties. Then if the client asks for the access permissions, there is no need to go to the storage element a second time. This can be a significant saving for larger sites, where we may need to check millions of files in a short space of time.

Another optimisation that can be performed is to cache per-directory information instead of per-file information. An SE that uses castor[7], for example, could get information about a file by using the *nsls* command. Running *nsls* for every file in a directory, one by one, is very slow compared to running it on the parent directory and gathering information about all files in the directory at the same time. CMS organises related files in the same directory, so this represents a considerable gain in efficiency, since we often want to process a set of related files at the same time.

One more optimisation that is useful is to use a 'storage dump' of the SE. This is a file containing a dump of the SE metadata database in a standardised format[8] which allows the file information to be retrieved much faster during subsequent consistency checks, at the expense of potentially stale information.

The Namespace framework is not PhEDEx-specific. It could easily be used by other tools that need to be run at several sites and that need to scale efficiently to large numbers of files.

3.3. The LifeCycle agent framework

The 'LifeCycle agent'[9](figure 2) provides a framework for debugging, for validation and for integration testing of PhEDEx and other components of the CMS data and workflow management system. The name derives from its ability to drive the entire set of PhEDEx components through their lifecycle in a controlled manner. It can inject data into PhEDEx at a given site, subscribe it for transfer to other sites, and make a deletion request for it after it arrives. By providing a fake transfer layer (emulating the behaviour of the *glite-transfer* commands) it can be used in a testbed to drive the real PhEDEx agents at several times their nominal workload. We routinely use this for scaling tests, for performance tests, and for debugging of new software.

The rest of the PhEDEx agents have no knowledge of the LifeCycle agent. As far as they are concerned, they simply see work appearing in their queue in the database, with nothing to distinguish it by where it comes from. We can run the entire PhEDEx machinery, with the real agents and whatever network topology we have instantiated in the PhEDEx database, and simply drive the activity with the LifeCycle agent.

Originally written specifically for PhEDEx, we have recently extended the LifeCycle agent to be a completely standalone component. This makes it attractive to other CMS dataflow projects, such as DAS[10], DBS3[11] and CRAB[12], which are all investigating the possibility of using it as part of their own testing and validation procedures.

The LifeCycle agent framework relies heavily on POE for its event-driven behaviour. The framework marshalls a 'payload' through a sequence of events defined in a configuration file, where the events are mapped onto event handlers which process the payload, modify it, and return the modified payload. Event handlers may be functions in a Perl module, or they may be external scripts, written in Perl, Python, or any other high-level scripting language.

For event handlers implemented in Perl modules, a simple API allows the event handler to be developed quickly. For handlers implemented as external scripts, the LifeCycle agent communicates with the script by a dropbox mechanism, passing a file containing a JSON dump of the payload, and expecting a similar JSON object in return. This allows the agent to be completely agnostic to the actual nature of the work that is being done, as well as to how that work is achieved.

The LifeCycle agent is also being used for non-PhEDEx-related activity. It is currently being used by the HEPiX IPv6 working group for simple network connectivity tests of ipv6 gridftp servers[13].

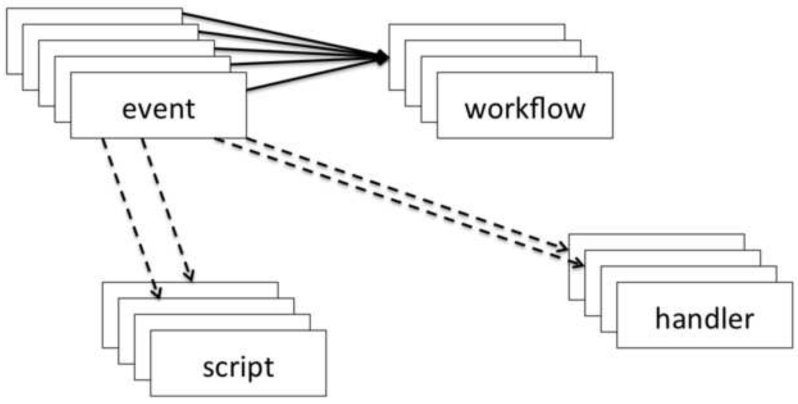

Figure 2. The LifeCycle agent takes a series of events, each of which corresponds to a Perl handler or an external script that executes an action. The events are organised into workflows, and the agent drives the sequence of events for multiple workflows in parallel.

3.4. The data service framework and the website
The PhEDEx data service is perhaps the most visible framework in PhEDEx, and has been covered in detail elsewhere[4]. Originally used simply to provide access to data, it is now used more and more to interact with the database itself, registering requests for transfer or deletion of data and changing properties of data subscriptions. This makes it the method of choice for clients wanting to automate activity in PhEDEx, such as the Tier-0[14] or the Monte Carlo production machinery (WMAgent[15]) for which the PhEDEx clients are written in Python. They can now easily use the web-based API to register and move new data, rather than having to install a sub-package of PhEDEx software and maintain database connection parameters for a set of command-line tools.

Other people have developed graphical 'widgets', custom web pages, or monitoring scripts based on information retrieved by the PhEDEx data service. The data service is an essential source of information for many parts of the CMS dataflow and workflow management system.

The website, too, is being redeveloped[16]. Originally a static server-side program written in Perl, it is being incrementally replaced by a javascript application based on the YAHOO User Interface (YUI[17]) framework. This makes for a better user experience, as feedback on missing or incorrect information can be given immediately, and there are less delays waiting for the server to respond. It also means that the visual elements (data-tables, tree representations, and eventually charts) can be re-used elsewhere, embedded in other pages for the purposes of aggregating monitoring information from many sources.

It also means that, instead of having separate code on the server to fetch and process information, the information is retrieved by AJAX calls to the data service. We eliminate a lot of duplicate code from the web server and, at the same time, gain consistency between what the user sees on the website and what they see if they use the data service to deduce information for themselves. In some cases, information was available on the website but not via the data service. Re-implementing the website has meant creating new data service APIs to capture that information, which automatically is then available to the user.

4. Conclusions
Re-factoring the original code to turn PhEDEx from a toolkit into a framework has had many benefits.

- developers can now easily re-use code from any of the agents in any other part of the project
- new developers can get up to speed faster, since there is less code to worry about for a given task
- we now have consistent semantics between the website, the data service, and the agents, because they share the same SQL statements
- the data service is now the single point of reference for all monitoring information, both for users of PhEDEx and for the website itself
- PhEDEx users can create custom monitoring or workflow-manipulation tools easily
- we have the possibility to use PhEDEx tools as the basis of a CMS-wide integration and validation test suite

This has been a work in progress for several years, as we opportunistically refactor parts of PhEDEx. We expect this work to continue, with the move from a dedicated toolkit to a set of open frameworks being a guiding principle of the way PhEDEx is developed in the future.

References
[1] Egeland R, Wildish T and Metson S 2008 Data transfer infrastructure for CMS data taking *XII Advanced Computing and Analysis Techniques in Physics Research* (Erice, Italy: Proceedings of Science)

[2] Molnar Z, Baud J-P, Salichos M and Keeble O Next generation WLCG File Transfer Service (FTS), submitted to CHEP 2012

[3] Storage Resource Management (SRM) Working Group, https://sdm.lbl.gov/srm-wg/index.html

[4] Egeland R, Wildish T and Huang C-H. PhEDEx Data Service. *Journal of Physics: Conference Series,* 219(062010), 2010.

[5] POE, the Perl Object Environment, http://poe.perl.org/

[6] The PhEDEx Namespace framework, http://cern.ch/go/J87G

[7] Hefferman J, Lo Presti G, Mascetti L, Peters A, Ponce S, Reguero I, Iven J and Lamanna M Overview of storage operations at CERN, submitted to CHEP 2012

[8] Lanciotti E, Magini N, Wildish T, Sanchez Hernandez A, Zhang X, Huang C-H, Ratnikova N and Serfon C Data storage accounting and verification in LHC experiments, submitted to CHEP 2012

[9] The PhEDEx LifeCycle framework, http://cern.ch/go/H7gB

[10] Ball G, Kuznetsov V, Evans D and Metson S, Data Aggregation System - a system for information retrieval on demand over relational and non-relational distributed data sources *J. Phys.: Conf. Ser. 331 042029, 2011, CHEP 2011.* (doi:10.1088/1742-6596/331/4/042029)

[11] Giffels M and Guo Y Data Bookkeeping Service 3 - A new event data catalog for CMS, submitted to CHEP 2012

[12] Spiga D, Cinquilli M, Vaandering E, Riahi H and Mascheroni M CRAB3: Establishing a new generation of services for distributed analysis at CMS, submitted to CHEP 2012

[13] Kelsey D et.al. From IPv4 to eternity - the HEPiX IPv6 working group, submitted to CHEP 2012

[14] Hufnagel D CMS Tier-0: Preparing for the future, submitted to CHEP 2012

[15] Fajardo Hernandez E et.al. A new era for central processing and production in CMS, submitted to CHEP 2012

[16] Wildish T, Magini N, Huang C-H and Rossman P The PhEDEx next-gen website, submitted to CHEP 2012

[17] YUI, the YAHOO User Interface framework, http://developer.yahoo.com/yui/