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

A. Sanchez-Hernandez[1], R.Egeland[2], C-H.Huang[3], N.Magini[4], N.Ratnikova[5], T.Wildish[6]
[1]Centro Invest. Estudios Avanz IPN, [2]University of Minnesota, now at Montana State University,
[3]Fermi National Accelerator Laboratory, [4]CERN, [5]Karslruhe Institute of Technology and ITEP, [6]Princeton University

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. It has undergone significant evolution over time, and continues to evolve today, despite being a fully mature system.

Originally a dedicated toolkit of agents and utilities, it is now a more open framework that can be used in several ways, both within and beyond it's original problem domain. A combination of refactoring and adoption of new technologies have made PhEDEx more flexible, maintainable, and scaleable.

The original architecture of PhEDEx

PhEDEx[1,2] was written as a set of agents, each performing a highly specific task. Some agents run only at a single, central site, others run at each site, configured for the specific conditions at that site.

Agents do not communicate with each other directly, only through changes they make in the central Oracle database.

This architecture corresponds to **a toolkit of 'power tools'**. The tools are not general purpose, and can only be used for the single task they are designed for.

This keeps the agents cleanly separated, which makes it easier to code and maintain them. It also simplifies tuning a PhEDEx agent for performance with ever-growing volumes of data.

Problems with the original architecture

The number of PhEDEx users is increasing. Operators want more detailed views of PhEDEx, physicists want to know about their own data, and experiment managers want summaries and higher level overviews of the behaviour of the whole system.

Many people write their own tools, often with significant functional overlap but little common code. This is a poor use of manpower.

There was minimal code-sharing between agents, neither of the Perl code nor the SQL statements they use. Duplicate code leads to maintenance and behavioural inconsistencies. Database load can increase if several variants of the same SQL statement are used.

When PhEDEx was originally developed, there was no freely available event-driven framework which could be used to build the agents. This made the agents less flexible than they could have been. Multi-threading, for example, was all but impossible to achieve.

Developing a new agent is a specialised task. For new developers, the learning curve can be significant, which makes it hard to use people who can only contribute a fraction of their time.

The solution

We are re-engineering PhEDEx as **a collection of open frameworks**. People should be able to develop code that is supported by PhEDEx, so they can accomplish their task easily, efficiently, and in a way that can be shared by others.

PhEDEx now has several lightweight frameworks for developers. They are the **data service**, the **next-gen website**, the **AgentLite framework**, the **Namespace framework**, and the **LifeCycle agent**.

The data service

PhEDEx provides a web-based data-service[3] for access to information. Data is available in Perl **Data::Dumper, JSON** and **XML** formats. A large number of API calls provide fine-grained selection of data.

Some APIs are generic, such as those for exploring replicas of data at sites, others are more task-specific, such as the one that reports stuck transfer queues at sites, based on a complex algorithm.

New APIs are being added constantly to address new needs of the user community.

AgentLite and the core agent framework

PhEDEx agents were re-engineered some years ago to use **POE**[4], the **Perl Object Environment**. This is an open source event-driven framework for co-operative multi-tasking. As a result, agents, or specialised tasks like monitoring, can be coded independently as separate POE programs, then combined in the same process for ease of management.

At the same time, SQL statements were factored out into libraries of functions which could be shared. Use of caching in the library functions further reduces the load on the database while keeping agent code simple.

Now we are breaking the agent behaviour into separate modules, for database interaction, dropbox queue-management, and others.

Specialised agents, like the PhEDEx watchdog, can be written using the AgentLite framework, dropping the parts they do not need. It can also be used for non-PhEDEx work, or outside CMS.

References

1. CMS Data Transfer operations after the first years of LHC collisions, CHEP 2012
2. Data transfer infrastructure for CMS data taking, ACAT 2008
3. The PhEDEx data-service, CHEP 2010
4. http://poe.perl.org/
5. Data storage accounting and verification in LHC experiments (Id 224)
6. CRAB3: Establishing a new generation of services for distributed analysis at CMS (Id 206)
7. Data Bookkeeping Service 3 - A new event data catalog for CMS (Id 187)
8. From IPv4 to eternity - the HEPiX IPv6 working group (Id 147)

The Namespace framework

The Namespace framework provides an interface between **high level activities**, such as checking the existence of files on a storage element (SE), and **low level details** of optimised access to a particular SE.

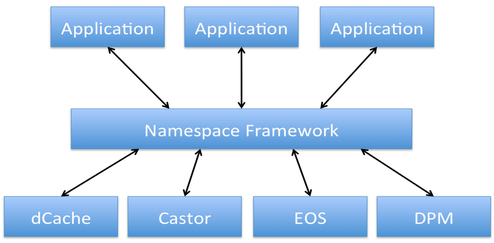
Tools written to use the Namespace framework do not need to know details of the SE. Operations staff, site managers, or PhEDEx developers can write tools that will run anywhere, unmodified.

Plugins to the framework tune the performance for each SE type. SE experts can write a plugin for the SE they know best, and all users of the Namespace framework will benefit from it.

One optimisation is to use a **dump of the SE metadata database** instead of direct access to the SE[5]. This trades speed for possibly stale information. The PhEDEx block-consistency checks are one example.

Or we can **fetch several file-attributes** simultaneously, and cache the result. If many attributes can be obtained at the same time at no extra cost, the values can be cached by the plugin.

Or we can retrieve information for **several files** at the same time. If we want information about one file, we will probably want the same information about the rest of the files in that same directory. This can lead to large gains in performance.



The LifeCycle Agent framework

The LifeCycle agent is a framework for **driving a system through a sequence of events**. It is used for **debugging**, for **validation** of new releases, and for **stress-testing** of hardware or software. The LifeCycle agent is not PhEDEx-specific. It is being used by CRAB[6] and DBS[7] within CMS, and by the HEPiX IPv6 group, for transfer tests[8].

We use it for **scalability testing**, running PhEDEx at artificially elevated rates, with fake file transfers. This is how we know PhEDEx scales to the data volumes and rates expected in the next 2-3 years.

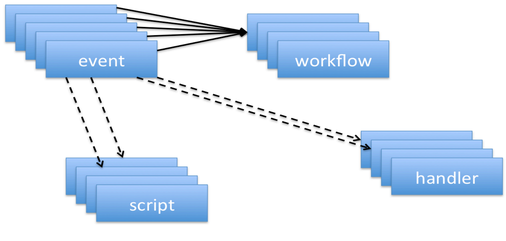
The LifeCycle agent combines a number of '**event handlers**' (Perl functions or external scripts) into '**workflows**'. Each workflow can be constructed to mimic some aspect of the work of the real system, and the agent can run a large number of workflows in parallel.

Workflows for PhEDEx involve injecting data into PhEDEx, subscribing it for transfer to multiple sites, and deleting it from one or more of them later. Any action that can be coded as an independent script can be turned into a handler and added to the workflow.

A handler needs to know the structure of the '**payload**' it is processing, and modifies that payload according to its results. The modified payload is then passed to the next event in the sequence. The same workflow can be started with different initial conditions (different initial payloads) which can cause a different resulting behaviour in the system.

A handler can create an array of output payloads, which will **fork the workflow**, so each payload is processed in parallel.

Handlers can add or delete events to **modify the workflow dynamically**, based on results of their processing.



Conclusion

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