Cylc Response to "Assessment Report on Autosubmit, Cylc and ecFlow"

Hilary Oliver, Andrew Clark, Ben Fitzpatrick, David Matthews, Oliver Sanders, Matthew Shin, 17 Jan 2016

The Cylc developers would like to respond to a recent comparison paper, *Assessment report on Autosubmit, Cylc and ecFlow (2016, Domingo Manubens-Gil et. al.)* and another that references it, *Seamless Management of Ensemble Climate Prediction Experiments on HPC Platforms (2016, Domingo Manubens-Gil et. al.)*. Two of us are listed as contributors to the first paper but it should be noted that this contribution was limited by time and workload constraints to correction of any major misunderstandings of Cylc.

The lead author of both papers is also the lead developer of Autosubmit so it is perhaps inevitable that the comparison plays to Autosubmit's strengths. However this was not made clear in the comparison paper, and we would like to address several points that we believe convey a misleading impression to readers.

1. The report is focused on a special type of workflow that happens to be Autosubmit's primary use case, a finite "multi-run climate ensemble experiment" in which each *member* of an ensemble is split into *chunks* (e.g. a 10 year integration may be split into 10 1-year chunks), and the whole structure is replicated over several *runs* (e.g. to run the same experiment for different start dates). The Autosubmit examples notably appear briefer and more straightforward than those of Cylc and ecFlow, but we believe this would not be the case for other types of workflow, because:
    - Autosubmit's built-in support for this workflow allows it to replicate tasks across "members", "chunks", and "runs" for the specific structure described above. But this is not a general parameter expansion mechanism; for other workflows every task would have to be defined separately.
    - The Cylc examples generate these tasks with explicit loops over the parameters. This is relatively verbose compared to Autosubmit (for this type of workflow) but it is completely general: Cylc can expand any number of arbitrary parameters at any point in a workflow.
    - Since cylc-6.11.0 *parameterized tasks* can achieve exactly the same simplification in Cylc, but without sacrificing generality. (The report's authors could not have anticipated this, however.)
2. In order to compare like with like, the Cylc *GloSea5* example should have been simplified:
    - It contains a lot of functionality that Autosubmit (and to a lesser extent ecFlow) does not have: clock triggers; conditional, failure, and suicide triggers for automatic recovery workflows; and date-time cycling. For many readers this will only give a misleading impression of complexity.
    - It derives from a real operational suite with very long descriptive task names and variable names, and as such the scheduling graph is not suited for display in a narrow document format. The resulting messy line-wrapping further contributes to a misleading impression of complexity.
    - (As per the final bullet-point under *1.* above, this workflow could be greatly simplified by rewriting it in terms of parameterized tasks instead of explicit loops).
3. It is briefly mentioned that Autosubmit does not have *task families*, but not why this is considered a critical feature in Cylc: in large workflows task families allow efficient mass triggering, efficient sharing of common task configuration without duplication or resort to global variables, efficient user interaction (e.g. to manually retrigger all members of a family at once); and tidier visualization and monitoring (summary state information is displayed for collapsed task families).
4. There is no discussion of scheduling configuration, a major point of difference between the tools that is apparent in the examples. Cylc's clean separation of scheduling and task runtime configuration allows users to work directly with workflow structure and it provides the basis for our advanced scheduling semantics (conditional, family, message, and task state triggers, etc.). In Autosubmit the workflow structure is implicit; it has to be reconstructed from the dependencies specified in each task definition.
5. Workflow cycling is another topic with significant implications that could have been expanded on:

- In Autosubmit we believe the entire workflow from start to finish is generated at start-up, with a different logical task for every job (the "chunks" referred to above are a proxy for dynamic cycling). Its workflows must therefore be finite in extent and not too large, and its users (like Autosubmit itself) are presented with the entire workflow at run time. This is a bigger barrier to open-ended real-time operational use than lack of real-time clock triggers.
- Cylc can generate these finite cycling-like static workflows for cycling jobs too, using parameterized tasks or Jinja2 loops. But Cylc also supports a flexible and powerful standards-based (ISO 8601) cycling notation for defining *continuous ongoing workflows of cycling tasks*. These get extended dynamically, potentially indefinitely, with no hard boundaries between the cycles.

6. The second paper states that a Python API for workflow definition (ecFlow), as opposed to a configuration file, is "easy, robust, and powerful". This point is debatable because most users are not primarily programmers, and a workflow definition of any type just *configures* the behaviour of a program (the workflow engine) which is precisely what configuration files are for. Config files are manifestly simpler and more robust than (most) programs: they are comprised of simple *key=value* pairs that can be validated against a spec, and have a consistent clear structure. Cylc's config files make it *possible* to write complex workflow definitions that can be understood at a glance by non-programmers. The main reason for programmability in this context is to allow efficient automatic generation of parts of a workflow definition, e.g. for ensembles of similar tasks, and Cylc users get this by embedding Jinja2 code anywhere in the config file. Jinja2 is Python-like, powerful, and extensible with custom Python filters. That said, we are still considering a Python workflow definition API for additional flexibility.
7. The second paper also asserts that with "a high number of cycle points active at the same time, the [Cylc] GUI could get slow and unresponsive". The Cylc GUI's efficiency, even on a low-spec laptop VM, is now such that the *tree* and *dot views* remain nicely responsive to the largest of current operational NWP suites (with several thousand cycling tasks). The (optional) *graph view* depends on the performance of the underlying *graphviz* layout engine and it struggles with very large ungrouped graphs. But performance aside it would not be the view of choice for very large suites anyway, and besides neither Autosubmit nor ecFlow have a live dependency graph view for comparison.
8. We don't regard lack of a built-in experiment database and version control integration as a limitation of Cylc any more than (say) lack of a program database and version control integration is a limitation of the Python programming language. Individual users and small and large sites may have very different requirements and preferences in this regard. Rose is used at many Cylc sites to provide this functionality. As the second paper notes Rose does increase the complexity of the software stack (somewhat), but it provides a powerful solution for workflow storage, discovery, and version control, and it supports inter-site collaborative development.
9. Cylc is a general purpose tool with a proven record in research and production environments. By contrast Autosubmit is quite specialized and is not geared for operational use, so the reports do not consider a host of other Cylc capabilities, some of which are not just useful in operations. Examples include internal queues, event handling, *broadcast* functionality, edit-runs, inter-suite triggering, and a powerful command line interface that can easily operate on many tasks at once.