Software Lifecycle Management for WLCG beyond EMI

Markus Schulz
markus.schulz@cern.ch
V-0.1

This is a summary of the discussions that followed the presentations to the WLCG-MB and GDB during the last months. The purpose of this text is to provide a starting point for discussions leading to an agreed strategy that can be refined and implemented by a technical working group.
The end of EMI affects mainly the middleware components developed in Europe. Several are used globally and a common approach to middleware lifecycle management, including OSG, is desirable. This includes also the participation in the work needed to verify services and clients.

Situation after EMI

EMI funding is not only used to develop and support grid middleware, but also to provide effort for coordination, QA, integration and packaging.
EMI integrates the majority of the middleware into the EPEL repository. With this move this middleware is packaged in a standard way following process and structure of a major Linux distribution. Most middleware components have reached a relative stable state with few fundamental changes expected in the near future. In addition EMI provides Source RPMs open the door for code contributions from third parties.
The EPEL FedoraProject provides a process to release packages and supports this process with a tool chain suited for distributed teams. The transition from test to stable repositories is transparent and can be influenced directly by our community. Furthermore build tools for creating EPEL compliant packages are readily available. Due to the packaging rules and licensing restriction not all material needed for WLCG middleware services and clients can be integrated with EPEL. Currently these packages are provided via the EMI and egi-UMD repositories.
dCache Java packages cannot be integrated with EPEL. Sites should take the server code directly from http://dcache.org/ as it is already common practice on most sites. The clients need to be integrated with the UI and WN meta packages. This can be done via the UMD/WLCG repository.

A concrete discussion on WLCG requirements for middleware repositories after emi has started with EGI:
https://indico.egi.eu/indico/materialDisplay.py?materialId=minutes&confId=1155

These repositories fulfil to a large extend the functionality required by this proposal.

For simplicity I will refer throughout the text to this additional needed repository as the UMD/WLCG repository since it isn't clear who will provide and manage these repositories in the future.

For WLCG EMI covers the following critical functions:

1. Coordination between Product Teams (PT)
2. Agreement on standards
3. Agreement on roadmaps
4. Concrete planning of major releases and duration of support
5. Link to related projects (globus etc.)
6. A central point to gather and discuss requirements
7. Tracking tools
8. Support
9. Middleware configuration
10. Middleware documentation
11. Middleware maintenance and development
12. Build, integration and testing services for SL and Debian (ETICS etc.)
13. Managed repositories
14. Alternative packaging (tarball releases)

In one form or another these functions have to continue.
ScienceSoft is in an early state and it is not clear which of these functions will be covered by it.

Most middleware components used by WLCG are currently supported by PTs (Product Teams) located at sites that are part of WLCG or are closely related to the HEP community. Most have agreed to provide effort to maintain the software, in most cases with reduced resources. The biggest gap left is in the areas that need coordination between PTs, users and sites.

EMI's role in the software life cycle ends at the release stage. Rollout and enforcing the move to supported versions and the decommissioning of obsolete versions is handled by EGI and NGIs. It is vital for WLCG to play an active role in this area.
EGI builds from the EMI releases UMD distributions with additional QA, documentation and a process for a staged rollout. In line with EGI's role this process is not VO specific. In addition EGI defines the set of middleware versions that are acceptable based on criteria such as security and support.
WLCG's experiments run extensive mission critical workloads that require an additional step of verification beyond the staged rollout. In addition WLCG has additional criteria for restricting the selection of versions. Simple examples are efficiency and functionality. On the long term a SE version that is secure and supported, but doesn't support federated data access, will not be acceptable for WLCG use.

Overall Approach

Given the reduced funding and the distributed nature of the community it is most efficient to base our future work on a software lifecycle process that relies as much as possible on widely used processes and tools for open software products.
With no funding for coordination the Product Teams have to operate as independently as possible.

Using the EPEL repository and process as an integration point is most natural. The tool chain is already widely used and well accepted by the developers. With the concept of certified package managers a certain level of maturity is ensured. (https://fedoraproject.org/wiki/Bodhi_Guide)

With a mission critical infrastructure some additional processes are needed to ensure that changes are not disruptive. Over the years we have learned that production readiness on scale can be only verified with real production load. This requires a carefully managed transition from the EPEL-test to EPEL stable repositories. In addition a method to rollback to previously working versions is needed.

The boundary between middleware and user code can't be drawn easily in our environment. As a result roadmaps have to be designed in close cooperation between the PTs and the experiments.

Version management, announcement and verification are needs that are not covered by EPEL. On our production infrastructure at any given moment several versions will be deployed concurrently and need to be detected reliably.

For coordination between users and PTs and between PTs we should reuse existing WLCG infrastructures, such as GGUS, WLCG-T1 Service Coordination Meetings, GDB and MB as much as possible.

With the reduced effort the WLCG community has to make better use of distributed effort. Examples are the support for additional batch systems, alternative packaging and configuration management.
In the past sites and users have expressed their needs to the middleware projects and those tried to deliver the requested support. If the delay seemed too long, local solutions have been developed, but often these have not been shared.

The overall approach should be that whenever there is a special need this is addressed where the need arose and incentives for sharing should be increased. It is hard to imagine that a single T2 site is in need of a modification that no other T2 requires.

Straw-Man Process

Based on the overall concepts this is a model for a possible WLCG process. The future roles of ScienceSoft and EGI are not well enough defined to delegate activities in a definite form to them. This is not a problem, because most of the functionality needed is generic and can be assigned either to a WLCG partner or to a project. Where a strong overlap is expected elements are named accordingly.
This is should serve as an example for a process to provide WLCG with middleware and manage the deployment and versioning. This doesn't mean that all elements have to be run or controlled by WLCG.

Integration Point, Packaging and Repositories

If technical feasible all software has to be provided in an EPEL compliant format. Those packages that can't be included in EPEL have to be provided in the UMD/WLCG repository. Alternative packaging formats, such as used for CERNVMfs, re-locatable tarballs etc. have to be derived from EPEL-stable and UMD/WLCG repositories.
The integration point is the combination of the EPEL repositories and the UMD/WLCG production repository.
Managing repositories has proven to be laborious and error prone. As a consequence the number of different repositories should be as small as possible. For problem resolution, testing and rollback a repository containing past versions of the software is needed in addition. Currently EMI provides this service and it has to be decided how we keep past releases of packages available after the end of EMI.

With the UMD/WLCG repository at the highest priority and this repository being "protected", a method is given to enforce versions different from those in EPEL-stable. The ability to overwrite EPEL-stable is important for emergency patches and rollbacks.
As described later, EPEL-test can be used during the staged rollout.

The hierarchy of repositories:

1. UMD/WLCG: non EPEL material and specific versions
2. EPEL-stable

Additional repositories:

1. EPEL-test: for middleware in staged rollout, managed pilots and verification
2. PT-private: product teams use private repositories for tests and local pilots
3. UMD/WLCG-history: archive of all middleware and dependencies

Product Teams

They organize their work independently and use build and test tools of their choice. It is highly likely that most of them will use the Fedora project's tools (koji etc.). Organization of the teams' internal workflows and fine grained tracking is the PTs' responsibility. Major functional changes of a team's products have to be verified in a Pilot Service before the PT submits the package to the EPEL-test repository.

Depending on the required feedback by the experiments, Pilot Services are either coordinated by the PTs or the WLCG Operations Team.
During the development and Pilot-Service phase the PT maintains its own repository.

Coordination

High level tracking of software issues can be handled by GGUS. The same applies for the collection of requirements.

The prioritization of requirements and discussions between PTs , sites and experiments can be handled at dedicated PreGDBs with a frequency of approximately 4 months. If the need arises due to changing requirements or extended, non-conclusive email discussions, additional meetings can be scheduled.
After the discussion at the PreGDB the WLCG-MB endorses the prioritized list and tracks the progress.
WLCG middleware development and support will depend largely on independent, distributed funding and on voluntary contributions. As a result strict, reliable roadmaps for the implementation of requirements can't be expected. The emphasis has be on expressing relative priorities and maintaining the coherence between the different components.
The decommissioning of components, interfaces, APIs and the move to new OS versions requires longer term planning with the involvement of all parties affected.
In case informal communication doesn't lead to a formal, documented plan that is acceptable by WLCG the WLCG-MB should initiate workgroups to clarify the implied technical issues and create consensus over a manageable timeline.
The effort to track the implementation of these plans has to be minimized. A "push" based mechanism where the party responsible for an activity announces the progress of the work in a public space seems to be the simplest approach. This can be done via tables in Twiki pages. These pages are then periodically reviewed either by the WLCG Operations Team, or the WLCG-MB.
The WLCG-Baseline Version Twiki page can be used as a starting point to inform sites and users of the minimal required versions (https://twiki.cern.ch/twiki/bin/view/LCG/WLCGBaselineVersions).

This page can be expanded to provide additional information needed by the sites to plan updates. Especially information on an estimated timeline for components, including the date on which versions will become obsolete, is needed.
The information on the verification status of new versions needs to be communicated in an easy way. This could be either a table hosted by the WLCG Operations team and updated by the PTs, test sites and experiments or ScienceSoft might provide a similar functionality.
To minimize the workload on a central team it is essential that information is maintained and updated by those who do the work. The central teams roles should be limited to review progress and act if timelines are missed.
Illustration: Staged rollout, step by step.

Middleware Configuration

Many suggestions have been made in the TEG-OPS-WG5. The complexity of the problem originates from the differences in configuration needs of different middleware components and the different depths at which sites have to interact with the configuration of a service or client package. It is unlikely that one approach will cover the whole spectrum.
Small sites profit from simple tools, such as YAIM or RPM post installation scripts. For large sites integration with their fabric management tool is a must. Given the multitude of tools, Quattor, puppet, cfengine, YAIM etc., it is not reasonable to expect that PTs acquire the necessary knowledge to support all. Whatever a PT uses internally should be made available, but most importantly a well-documented generic configuration guide has to be provided. The current YAIM scripts can be used as a starting point.
Sites that support specific configuration tools should be encouraged to share their work. To limit the effort for single sharing of the work between sites is essential. This can be promoted by leaving room in the GDB and pre-GDB for site admins to present their work.

Validation in Production, Staged Rollout

The staged rollout process for clients and services requires the verification that new releases can be deployed and that the experiments use cases work with the new versions. Testing on test infrastructures and testing below production scale is necessary, but at the current quality level of the software a test with the experiments' workload at production scale is needed to avoid costly incidents.

This requires a significant effort from sites and experiments. Currently we are preparing an early form of this process for the validation of emi clients. For this a set of six sites have been organized that support different types of storage elements. To limit the coordination effort a Twiki page has been setup on which the sites announce their readiness and the experiments communicate the status of their tests. The WLCG Operations Team monitors the progress and will announce the next step towards deployment at scale.
This can serve as a model for a more structured involvement of WLCG sites and experiments in the staged rollout. It is important that the work is shared between different sites. For WLCG this sharing could be either organized by a rota system or a fixed link between sites and components. Both approaches require some coordination effort. For EGI this is done by EGI, for WLCG the WLCG Operations Team could handle the coordination between sites, PTs and experiments.

The staged rollout will start when packages are moved from the PTs repositories to the EPEL-test repository. A sufficiently long period for the transition to EPEL-stable should be set to avoid that packages are moved without proper validation. It is the PTs' responsibility to inform the involved sites and experiments.
Any of the stakeholders can set the karma level in the EPEL Fedora portal to stop the process. Issues are reported in parallel via GGUS.

Sites contributing to the staged rollout will install the new middleware packages from the EPEL-test repository against the EPEL-stable repository, unless more recent dependencies are needed.
This activity requires significant resources both for the technical work and the coordination.
A significant fraction can be handled by EGI and there is overlap with the role of an WLCG Operations Team.

Grid middleware and the experiment software have deep dependency trees. The constant flow of updates of base libraries requires in principle a permanent retesting of all our software.
This seems to be problematic. However, most changes are small and other communities find most problems while packages are in the test repository. Only rarely a serious defect in widely used packages passes undetected to stable.
During nightly builds a basic verification against EPEL-test improves the situation further.
For packages that are not much used outside our community, such as Globus, changes have to be tested explicitly via staged rollout. An agreed list of packages that WLCG needs to watch is needed. This list can be discussed at the middleware related pre-GDB meetings. Tracking changes of listed packages can be largely automated.
This automated monitoring of changes in the EPEL-test repository should cover all components that are likely to affect WLCG. To schedule the verification to a convenient time the Operation Team can extend the period a package should stay in EPEL-test or find an agreement with the developers.

Rollback

The concept of rolling back to the last recent working version isn't a concept well supported by the EPEL repository. The concept is to provide only the most recent version. Problems with new released packages are addressed by rolling forward to a fixed version.
Giving the deep dependency tree of middleware, the complexity of the experiments' environments and the variety of site setups, there is a non negligible probability that even a careful validation cannot guarantee that new packages will be free of critical issues. Rollbacks should be seen as a measure of last resort and not as common practice. The very same mechanism can be used if WLCG encounters problems with other EPEL packages.
To rollback the previous version of the packages are moved into the UMD/WLCG repository. This repository is protected and has the highest priority. Nodes that are updated after the rollback will receive the old packages. Nodes that upgraded before the rollback have to remove the faulty package or rollback locally via rpm commands before updating the node.
An approach easier for the site can be implemented via the epoch functionality of RPMs. However this concept of a hidden version number is confusing for

developers, site admins and users. In addition it is in conflict with the EPEL style guide.
The faster a rollback starts, the less work is required. Therefore the WLCG Operations Team should coordinate rollbacks.

Managing and tracking middleware versions in the infrastructure

Defining the problem

In the past (pre emi-2) the middleware providers produced versioned components. Clearly defined sets of RPMs, for the middleware layer, were defined by versioned meta packages and a set of repositories. The minimal acceptable versions were communicated via the WLCBaselineVersions twiki page. This table covers ARC, glite, LCG specifics and emi. Sites should be at least at the listed versions. It doesn't recommend specific versions. It communicates the lowest denominator.
That sites have installed acceptable versions of the middleware is verified in an ad hoc way, some sites use Pakiti on the WN, SAM and direct usage are further ways to discover versions. Services publish their version into the information system.
For readiness challenges and security exercises versions have been verified explicitly and thoroughly. However, no systematic approach has been established for all components. Recently EGI started to track versions systematically.

The situation is more complicated for clients due to the different distribution channels. Clients can be installed as RPMs, re-locatable tarballs and via versions provided to the Application Area. During the build up of WLCG sites installed frequently inadequate versions of the clients. Experiments needed to provide client libs and tools via their software distribution mechanisms. As a result the locally installed clients are not necessarily those used by the experiment software, a situation complicating problem resolution.
In the past services that benefited directly from improvements to elements within the dependency tree could move forward independently. This, together with versioned meta packages, made it easy to understand the set of software on a node.

With the move of the integration point to EPEL the approach changed. This is not due to a policy decision by emi, but due to the fundamental concepts of EPEL. Versioned meta packages are no longer provided, nor are they meaningful. All dependencies of all components are satisfied by the latest version of the EPEL-stable repository. This is based on the assumption that all critical issues are found during the time packages are in the EPEL-test repository.
This includes foundation middleware such as Globus provided by IGE via EPEL. For an OS distribution this is a reasonable. For most use cases frequent updates to the latest versions are recommended. The assumption is that within a very short time window the world is at the same version.
Frequent updates of our complex services are not practical for all sites.
The problem that we face now is that we can't express for our middleware the lowest acceptable version and can't track what is installed on sites. For client

libraries we also suffer from interferences between the locally provided software and the libs that are brought to the WN with the experiment software. There is no clear boundary between the middleware specific components and the application specific dependencies. This leads to behaviour that is very difficult to understand and correct.

Potential solutions

The Product Teams release service components through the EPEL process without versioned meta packages. EGI/WLCG creates versioned meta packages for components. The lowest version number is increased whenever the PT releases an update. External dependencies are not specified in detail. This doesn't conflict with the ability of a site manager to consciously select a specific version.

Services publish the installed version automatically with the ResourceInfoProvider into the information system. To improve security this version number should be encoded with a public key. The required schema can be defined within GLUE-2 . This information will be collected and monitored by the information system.
To provide more information for the site admins and operations team the info provider also creates detailed version logs locally. These are not published, but can be locally processed and compared with the WLCG/NGI/EGI version policies to plan work and resolve problems.

For the WN a better separation between locally provided middleware and experiment provided components is helpful. The sites provide basic WNs. On these the elements that are needed for a working glexec and to bootstrap the pilot are installed. The versions are tracked either by pakiti or a version number. These can be obtained either from the versioned WLCG/EGI meta package or from the tarball. After the pilot takes control the experiment software changes the paths to the middleware clients that are moved with the experiment software to the site. The versions can be identified in the same way as before and should be included in error logs. This ensures that the experiment's environment is self-consistent. In this scenario the experiment is responsible to follow client releases in line with the published WLCG policy. Currently it is the sites' responsibility to provide clients compatible with their locally deployed SEs. With this change this becomes the experiments problem.
Sites can still provide for other VOs complete sets of middleware clients.

Examples:

Step by step descriptions of typical steps in the lifecycle

TO BE DONE

Required Resources

All efforts are estimates assuming a significant reduction in development activity after the end of emi and should be seen as a starting point for a discussion.

Most effort is distributed and has to be seen as part of the work of the PTs.

For the central coordination of the pilot services, validation of WNs and the staged rollout. (1FTE)

Effort by sites and experiments contributing to the staged rollout and validation is difficult to estimate and depends on service and release frequency. Six sites are the minimum to cover all SE types. Additional might be needed to cover all experiments. More sites can contribute on a rota basis. For this to work significant coordination would be needed.
(0.1 FTEs/site, 0.1 FTEs/experiment)

Production of versioned meta packages. This depends on the release frequency and can be largely automated. (0.25 FTE)

Central management of the WLCG version Twiki page, assuming that test sites, experiments and PTs update their information (0.1 FTE)

EGI/WLCG repository management. (0.25 FTEs)

Alternative packaging of clients in tarballs (0.1 FTEs)

Coordination and maintenance of the agreed roadmaps. Organizing meetings and documenting agreements. (0.2 FTEs)