**Work Area**

*WG4 – Operational Requirements on Middleware*

**Introduction and Overview**

There are 3 perspectives to operational requirements on middleware.

**Site perspective**

1. The sites have to **operate** middleware services.
	* The services should be as **simple** as possible, with few features that essentially are unused.
	* The services should be as **robust** as possible, i.e. admin interventions should rarely be needed.
2. They need to understand whether the services are **configured** correctly.
	* The **documentation** should explain the expected behavior resulting from the configuration.
	* Any parameters not explicitly configured should have reasonable **default** values where possible.
3. They need to understand whether their services are functioning correctly, e.g. via **monitoring**.
4. When a service is not functioning correctly, the **monitoring** should notify site people of the malfunction in a way that tells them efficiently:
	* what exactly is malfunctioning;
	* why it is malfunctioning;
	* which steps need to be taken to resolve the situation.
5. If a user **reports a problem** to the site, one of the **requirements on the middleware** is that it should be possible to quickly understand what the real problem is:
	* Good error messages and logging.

**User perspective**

1. Users (experiments) have to **use** the middleware clients and services.
2. It should be easy to understand how to **configure** a command or a job to make it do what is wanted, this should be well **documented**.
3. The system should be as **simple** as can be reasonably expected.
	* Standard usage should be straightforward.
4. Clients should be **robust**, e.g. fail over to another instance of a service when possible.
5. Commands should have reasonable **defaults**.
6. If something goes wrong with a user's job or requested operation, a **requirement on the middleware** is that it should be easy to understand what caused the problem:
	* Good error messages.

**Model perspective**

1. The **model** of clients and services should be as **robust** as possible.
	* Complexity should be reduced where possible.
	* Clear distinction between responsibilities of experiments vs. sites.
	* One site should not be able to cause another to be degraded (cf. FTS channel model).
	* Services should be able to throttle operations in times of congestion and clients need to deal with that through intelligent retry and fail-over logic.
2. **Correlations** between services should be easy to track.
	* Logging and monitoring should facilitate that.
3. The **state** of a service instance should be advertised consistently and **taken into account** by clients.
	* The service itself, the information system, the GOC DB, monitoring pages … should agree on the state.
	* Clients should make use of downtime information to help prevent unnecessary tickets.

**Technology and Tools**

* Documentation
	+ Good documentation will help both site admins and users understand how a particular service or client is supposed to behave with a given configuration, allowing for a reasonable setup to be designed from the start and subsequent logging messages and occasional errors to be dealt with more easily.
* Error messages
	+ On the server side they need to identify unambiguously which operation was attempted for which client connection at which time and why it failed.
	+ On the client side they need to indicate to a reasonable degree why and when the operation failed and if the problem may be transient.
	+ They need to avoid clutter that should only be visible at a higher verbosity or debug level.
* Logging
	+ Log files should have sensible rotation policies compliant with the 90-day retention policy required by WLCG.
	+ Messages logged at a given level need to identify unambiguously which operation was executed at which time and, when applicable, for which client connection and/or by which thread.
	+ They need to avoid clutter that should only be visible at a higher verbosity or debug level.
* Robustness strategies
	+ Services should be more prepared to deal with minor problems preferably by design, but in any case as much as possible out of the box. For example, the effect of a memory leak in a third party library could be drastically reduced by letting the service make use of short-lived processes rather than long-lived threads. If a regular graceful restart is needed for the time being, the service should already come with that functionality (e.g. a cron job).
	+ Complexity should be reduced where possible and carefully divided between sites, experiments and infrastructure providers. In particular, if sites are asked to do more, they will be more likely to fail in some respects.
	+ Services should be designed with long-term scalability in mind.
	+ Services and their clients should as much as possible be designed to allow for load-balanced deployment of redundant service instances.
	+ Services should as much as possible be designed to allow for High Availability solutions in their deployment.
	+ Clients should allow for configurable retry and fail-over strategies where they make sense and have reasonable defaults for the maximum number of attempts and the maximum time spent.
	+ Middleware should conform to standards and popular technologies as much as possible. This will in particular allow site admins to spend less time in setting up services and understanding their behavior and allow common tools to be used.

**Procedures and Policies Described**

### EGI requirements gathering process

### The open operational middleware requirements of EGI are summarized at:

### <https://wiki.egi.eu/wiki/Requirements_2010_2011>

### EGI periodically collects operational requirements that concern the deployed software, namely ARC, dCache, gLite, UNICORE and Globus. Requirements can be submitted through the EGI Request Tracker system:

### <https://rt.egi.eu/rt/index.html> (“Requirements” queue)

Read access to Requirements tickets is open. Requirements can be submitted by anybody: site administrators, Operations Centers etc. The submitter needs a valid SSO account. Anybody can get an SSO account on the EGI Single Sign-On system:

* <https://www.egi.eu/sso/>

Requirements can be characterized by adding extra attributes to the RT ticket, which qualify the identity of who is submitting (member of a user community, project member etc.) and the type of problem (middleware distribution affected, type of capability, etc.). Detailed instructions on how to open a RT requirement ticket are provided at:

* <https://wiki.egi.eu/wiki/Mw-requirements.html>

The same process can be followed to submit requirements that concern tools maintained by EGI, namely: accounting, accounting portal, GGUS, GOCDB, Operations portal and dashboard, SAM. The list of open tool requirements is available at:

* <https://rt.egi.eu/guest/Dashboards/1039/Operational%20Tools>

Middleware requirements are collected, discussed and prioritized at the Operations Management Board (OMB), which meets monthly:

* <https://www.egi.eu/indico/categoryDisplay.py?categId=19>

Operational tool requirements are collected, discussed and prioritized at the Operational Tool Advisory Group (OTAG):

* <https://www.egi.eu/indico/categoryDisplay.py?categId=4>

J. Shiers and M. Girone are both in the OMB and OTAG. Requirements that are approved are then either submitted to the Technology Provider such as EMI through GGUS (in case of enhancements) or discussed at the Technology Coordination Board - where EMI is represented - in case of major requirements with architectural implications or involving multiple products.

Sustainability can be improved by:

* Joint EGI and WLCG task forces in requirements gathering.
* Exploiting more the existing EGI resources for these matters.

**Areas of Improvement**

Note: detailed suggestions are provided in the **Technology and Tools** section.

|  |
| --- |
| Improvement Areas |
| Impact | Areas |
| 5 | Documentation |
| 7 | Logging |
| 7 | Error messages |
| 8 | Robustness strategies |

**Areas of potential efficiency gains**

Efficiency can be gained by addressing in particular the areas of improvement with high impact.

**Please identify the largest use of operational effort**

|  |  |
| --- | --- |
| Sites | Investigation of middleware client failures reported e.g. by users and of service misbehavior detected by sites themselves |
| Experiments | Investigation of middleware client failures |
| Infrastructure Providers | Assistance of sites and users with investigations of middleware failures |

**Missing Areas**

None identified.

**References**

<https://twiki.cern.ch/twiki/bin/view/LCG/WLCGTegOperationsWG4WG5>