# Activity instance schema use cases

1. Scheduling use case

Description:
A client sends a request to a scheduler (could be EMS in OGSA speak) using an activity template which describes the requirements of the submitted work unit.

The initially receiving (primary) scheduler takes the template and, if it is willing to handle it, creates an activity instance for it, storing the initial template and, if applicable, additional information. The latter should at least include a "provenance record" which denotes that the current scheduler has taken over responsibility for the execution of the given activity. Other candidate information may include scheduling attributes, dependencies to other activities, and the current state of the activity, possibly reusing the BES state model.

On activity delegation, the delegator acts like a client towards the potential delegatee, offering the job to another scheduler. Again, if the delegatee is willing to accept the job, it takes over responsibility and the provenance records and depending information (e.g. the expected BES) are updated. If necessary, the activity template may be modified, as long as the manipulation history is kept.

Throughout the whole process, state information is constantly updated in the activity record. After activity completion, the resource consumption is written to the activity record. The corresponding entries and dependent parts of the record could then be sealed (marked final) as to denote the completion of the activity.
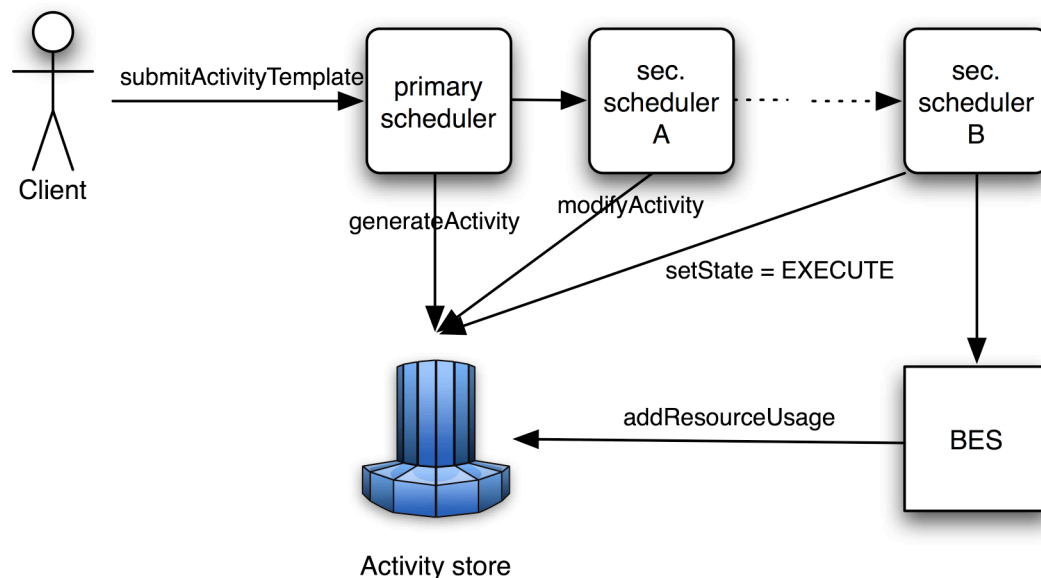
Actors:
- *Client*. Maybe a user accessing the *Primary Scheduler* directly or a component doing it on the user's behalf.
- *Primary Scheduler*. The scheduler that receives the Activity template and that generates the Activity.
- *Secondary Scheduler(s)*. An arbitrary number (but at least one) of schedulers, to which the Activity is delegated.
- *BES*. Executes the job related to the Activity.

- *Activity Store.* Some instance (potentially distributed) where the Activity is stored.

Endorsed by:
- Grid Scheduling Architecture Research Group
- D-Grid (Germany)

Specific requirements:
[R001]
Description of activity and its requirements including:
- scheduling QoS,
- dependencies to other activities → Use Case III (move).

[R002]
Current state of the activity (e.g., as defined by BES state model). Question: Copy Activity state to the Activity document or provide link to running Activity? A: Snapshot here.

[R003]
"Provenance record", for example, which scheduler has taken over responsibility for the activity.

[R004]
A history of records as required by [R003].

[R005]
Decisions made by a scheduler for the activity like e.g.:
- A record of the information (e.g., candidates) used for the decision.
- Since this recorded information can be huge there are probably different levels of support:
  - Provision of a "container" and an example of what kind of information may go in it, or
  - a pointer to a location where that information is available
- This is probably a low priority requirement.

[R006]
The submission of an activity to the secondary scheduler is potentially different than to the primary scheduler. Information about the activity "instance" (e.g., EPR) could be passed to the secondary scheduler instead of a template. This information could be part of the schema but does not have to. As this touches on implementation of an activity, which is out of scope, it has been agreed to postpone discussion for a later time (but to keep the requirement). → Low prio, to domain-specific.

[R007]
Records of resource consumption as part of the FINAL activity record.

[R008]
Ability to seal or mark final an activity record.
This also raised broader security or confidentiality issues. It has been agreed that there are a number of requirements including:
- Access control (on the whole activity or specific records).
- Policy definition (which nodes can be accessed by whom and when).

These requirements should be handled by existing security specifications, but the issues should be identified in a security considerations section.
[R009]
If a job is rejected should there be any information on who rejected it and why? And how should it be captured?

- At a basic (/simple) level this could be captured by state changes in the activity.
- There is no standard way to capture the reason for rejection; this could be free format in the general case.
- In certain narrow cases more information could be provided: For example in the scheduling use case, a maximum number of retries could be a reason.
- Schema or format for the reason of rejection is necessarily out of scope.

2. Job lifecycle tracking use case

Description:
The general purpose of this use case is to track a job (and its attributes) throughout the entire life cycle of a job. This is from the point in time where the job enters the system (activity creation; "Pending" in BES terms) to a point in time where the job is not active any more.

It is very useful to be able to see various bits of information about a particular job (an activity instance) at various phases within its life cycle. Different types of information are particularly relevant at different points in the job lifecycle, and the information can be used for various job-related activities such as job monitoring, system diagnosis, or capacity planning. The types of information that should be tracked for this use case are as follows:

- The point at which an activity has been submitted (and the unmodified parameters that were provided at submission).
- Any state changes that occur (according to the underlying state model). This would include information about the time of the change, the old state, and the new state.
- Information about the activity being forwarded from one scheduler to another (pertains to meta-scheduling and peer scheduling use cases). This part of the use case is similar to the delegation part of the "Scheduling use case".
- A report of resources consumed by the job both during the run time of the job, as well as a final "accounting" record that summarizes the job resource usage.

It is essential for this use case that the system provides access mechanisms which allow to access the different types of information mentioned above during job runtime by querying the BES (for example). In addition there is a demand for the possibility to examine a file that contains activity records.

Actors:
- *End users*. Those are monitoring the job progress and potentially the job's history.
- *Administrators*. They need diagnose means to analyse information about system exceptions (and take the necessary measures). The granularity of the information provided may vary with the specific application domain.
- *Scheduler/broker/etc*. Those entities (or humans) exploit the activity-related information to plan the job's execution, migration, re-scheduling, etc.

Endorsed by:
- Platform Computing

Specific requirements:
[R019]
Job attributes and state changes for entire lifecycle are required.
- From the point where the activity enters the system to the point where it is not active anymore.
- NOTE: It is not clear if the use case requires the information to stay around in the system after activity is no longer active (probably), and if so how long (implementation-dependent).

[R020]
Information to track:
- Initial submission requirements (e.g., JSDL); (Location: BES resource?)
- State changes, including a time-stamp
    - Handover of responsibilities (similar to delegation in the "scheduling use case" ; [R006]).
        - Recording of who made a decision and what that decision was
    - Resource usage during the runtime of the activity.
    - The final accounting record and perhaps a summary record.
        - NOTE: Summary records are not clearly a part of the Usage Record 1.0 specification; should probably be separate from the usage record entry.

[R021]
Access mechanisms, e.g.
- Query state, UR, (Chris, please detail …) while an activity is running, e.g. on a BES container.
- Potentially other service afterwards (maybe what 'file' refers to).

[R022]
Information when an exception happens (probably related to state changes) and what the exception was.

3. Workflow fault detection and recovery
Description
Initially the Client submits a workflow to WMS layer which divides workflow description into the set of activities, and these activities will further distributed to the backend target systems (BES in this case). The scheduling function of the WMS layer will take care of the resource allocation. To avoid naming confusion between a workflow and an activity, consider a workflow as a Whole Activity (WA); an activity within a workflow is a Sub-Activity (SA). As soon as the client submits WA, the WMS creates an entry into the AS by storing each of the SAs initialization, storage, and computing resource information. While individual jobs are submitted to the EL, the WMS updates the AS with each of the SA's status, the address of the BES, data location, and record of the computing resource utilization. In job submission scenario, it is probable that SAs could be finished successfully or can be failed due to an unexpected software or hardware failures. For the successful execution of WA, it is important that the child SAs should be successfully executed and must ensure recovery in the events of failure. There could be many scenarios where failure could happen; one scenario of failure could be that, when WMS executing a particular SA detects a fault, this will be reported inside the AS and then it updates the SA history. If WMS intends to recover the failed job and resubmit it to the next available BES, then it can

leverage the AS's job history information to consider resubmit/reschedule. Each SA submitted is check pointed into AS for job recovery and resumption to the best available BES instance at the time of failure.

On the other hand the job history for each SA will allow WMS to better schedule the new incoming jobs to the available BES instances.
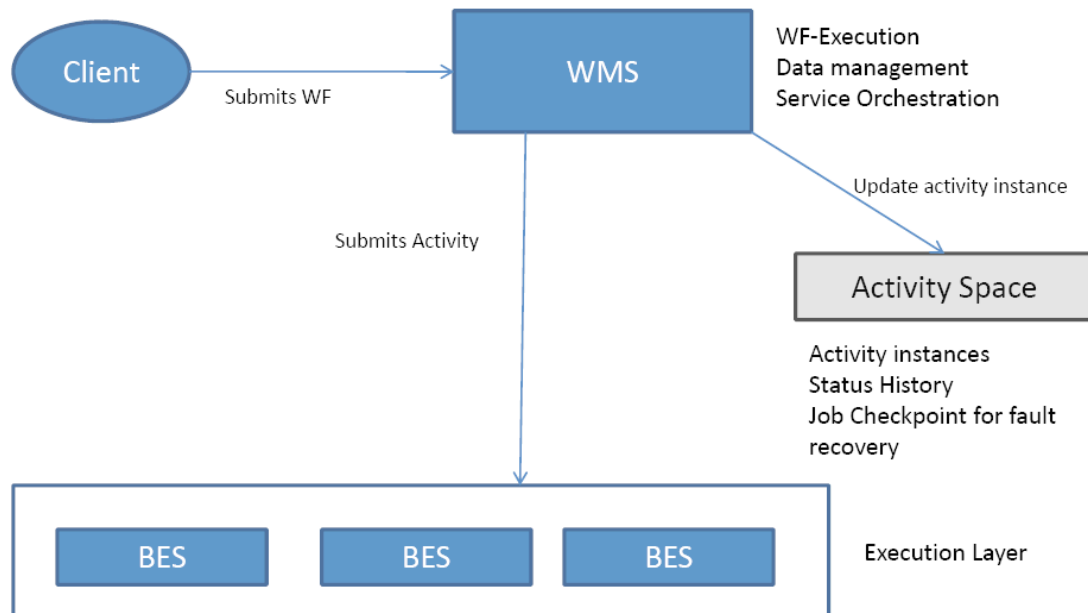
Actors
- *Client.* A workflow client.
- *Workflow Management System (WMS).* A component contains the services to manage workflows; orchestrate workflows into individual activity fragments; data management.
- *Execution Layer (EL).* The EL manages job execution on the target resources (a set of BES instances).
- *Activity Space (AS).* The AS stores activity templates and captures provenance information of the orchestrated activities from the time of submission until the job finishes; job check-pointing information to recover or resubmits the job at the time of failure; resource usage information for accounting and billing.

Endorsed by:
- Chemomentum Project (www.chemomentum.org/)

Specific requirements:
[R010]
Job check-pointing information. This could be part of the record or a pointer to another location.

[R011]
Detailed information about fault types and what caused the activity to fail.

[R012]
Resource usage information for accounting and billing.

[R013]
Status history (any transition). The transitions should have time-stamps attached.

[R014]
Address of the BES (EPR) executing activity.

[R015]
Data locations used by each activity. → (Shahbaz, more details, please).

[R016]
Status of each sub-activity and of the whole activity.

[R017]
Description of dependencies.

[R018]
History of records.

4. <Give me a title, please.>
Description:
- A common format like JSDL and this activity schema helps interoperability.
- Administrators need this information for troubleshooting, decision making, capacity planning, etc.
- Customers also need to see this information to see how/if things work properly.
- This information can also be used to perform simulations on new job flows that are created from pre-existing jobs.
- Historical record helps in estimation of future job executions used for simulation and critical path monitoring.
- A repository is needed for this information - this can get quite complex federated?

Actors:

Endorsed by:

Specific requirements:
[R023]
A record of what to execute (e.g., JSDL).

[R024]
A record of what happens to activity through execution (generate a series of events).
- State changes are captured as individual records (events).

[R025]
Jobs may have problems during execution and they might get rescheduled (through third-party products) etc.
- This might require temporary (this is execution only) changes to the job definition (JSDL). Activity ID stays the same.
- It's also important to track all changes as events (changes in state).

[Add on]
Schema requirements:
- Fixed part where a specification exists (e.g., initial submission JSDL) plus extensibility for more private/proprietary information (NOTE: How far to go either way is a balancing act.
- From a software vendor's point of view we prefer a fixed specification, this provides the best opportunity for interoperability.

5. Financial service provision use case
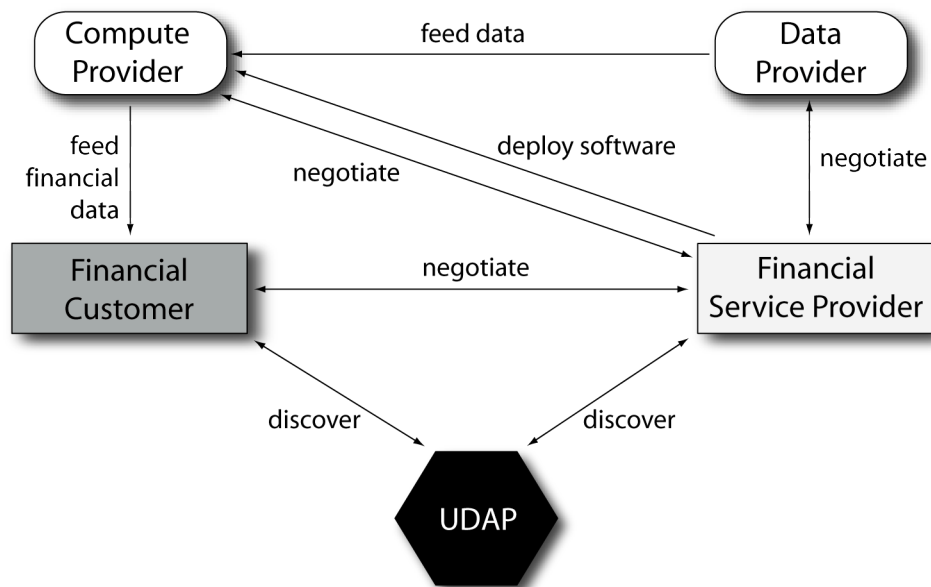
[Please note that this only describes the use case on a business level. Activity/SLA-specific details are not sketched.]

Description:
Traders working for a Financial Service Customer use a data feed provided by a Financial Service Provider. The feed consists of market data processed according to algorithms supplied by the customer. The provider in turn uses its own or third party Data and Compute Providers to provide this added-value service. The discovery mechanisms provided by the UDAP framework (Unified Dynamic Activity Package) allow the customer and provider to find useful services (financial, data or compute services respectively). All service interactions are governed by Service Level Agreements, except for the interactions involving the UDAP framework (which could be described as a set of "utility services"). Once an SLA has been agreed upon, an activity is created which includes inter alia the SLA (which does not change over its lifetime!). The activity is then used to monitor the compliance of the QoS with what has been agreed upon in the SLA. Breaching an SLA results in a negative ranking of the service provider (also tracked by the activity).

Actors:
- *Financial Service Customer*. Customer using financial services.
- *Financial Service Provider*. Provider offering financial services.
- *Data Provider*. Provider of financial data.
- *Compute Provider*. Provider of computing resources.
- *Unified Dynamic Activity Package (UDAP)*. Used for service discovery, activity creation (activity == SLA), and SLA tracking.

<u>Endorsed by</u>:
- NextGRID project

<u>Specific requirements</u>: