Provenance model discussion

Ole Streicher, Anastasia Galkin

ole@aip.de

College Park, 2018-11-08

Content

- Parameter: definition, data model
- Specialized entity and usage classes
- Limit to core model?

Parameters in VOTABLE

- `FIELD` specifies metadata of a table column. Attributes: `ID`, `datatype`, `unit`, `utype`, ...
- `PARAM` specifies *one single* `value` with the metadata
- VOTABLE always carries values; heuristics with `ucd`
- no connection to its generation or usage
- attribute `utype` *may* point to a role in a data model
- Rather `Quantity` than provenance information

Parameters in UWS

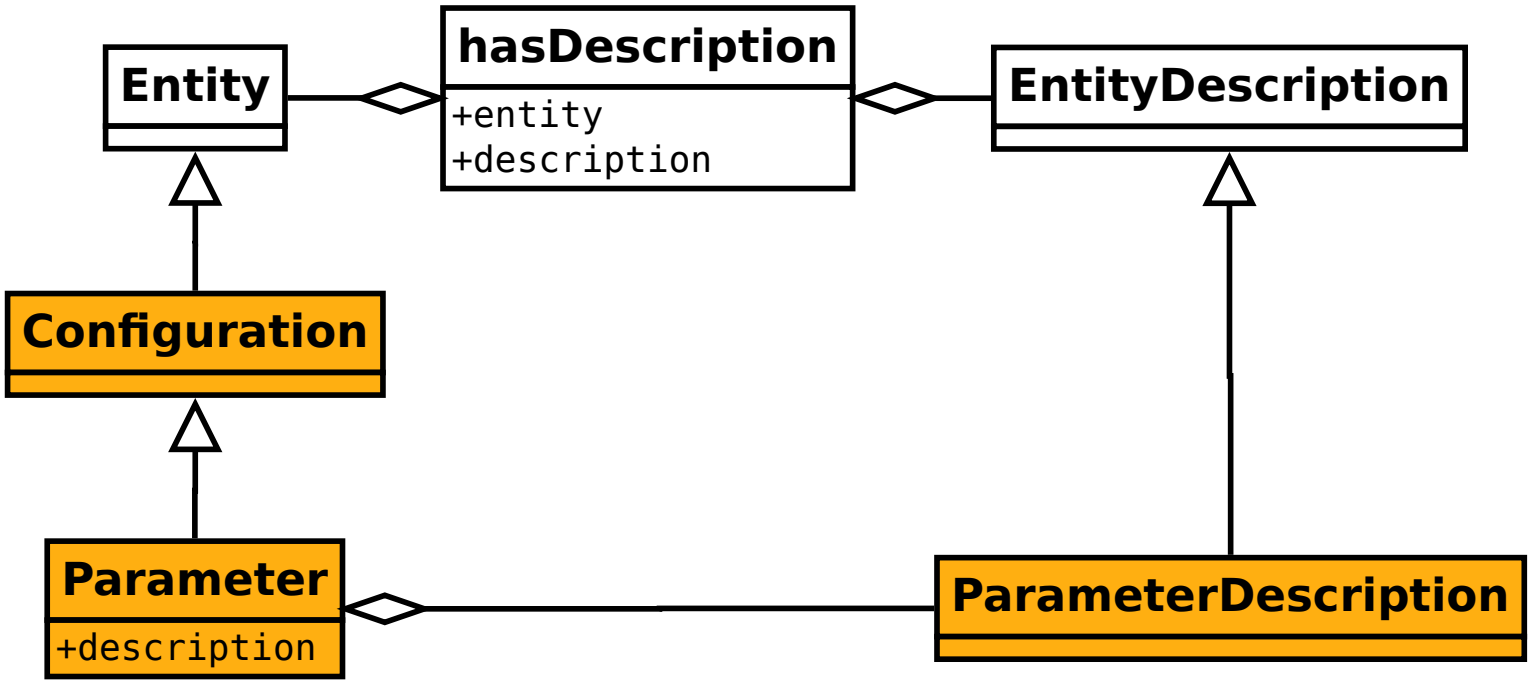
```
<uws:job version="1.1" ...> <...>
  <uws:parameters>
    <uws:parameter id="scaleFactor">1.8</uws:parame
    <uws:parameter id="image" byReference="true">
      http://myserver.org/uws/jobs/jobid123/param/i
    </uws:parameter>
  </uws:parameters>
  <uws:results>
    <uws:result xlink:href="http://myserver.org/uws
  </uws:results>
</uws:job>
```

- (science) data input, or configuration
- specify directly a value, or refer to a file

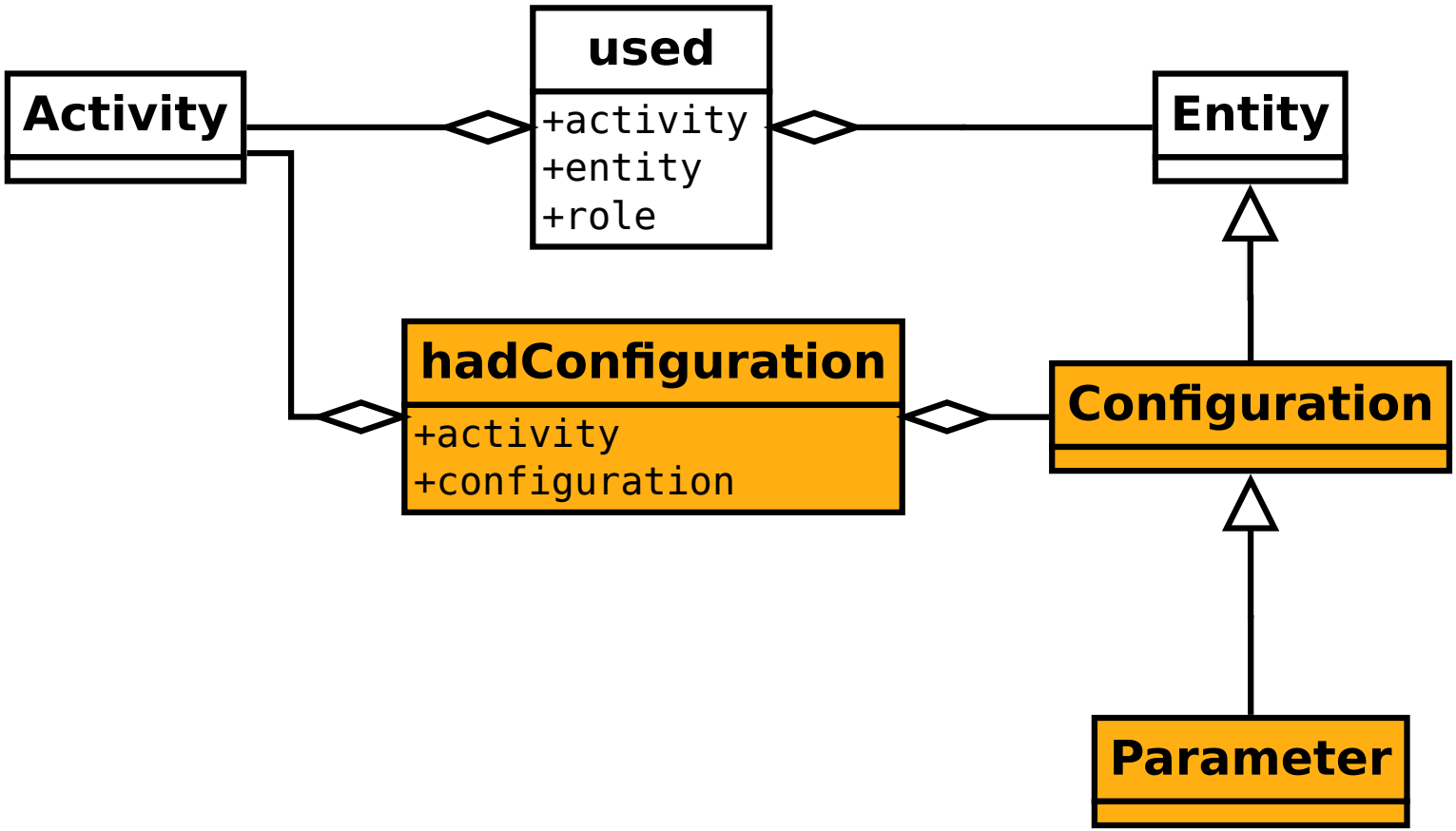
Handling in Provenance

- W3C Provenance `prov:entity`: "is a physical, digital, conceptual, or other kind of thing with some fixed aspects; entities may be real or imaginary"
- `prov:entity` may directly carry values, or refer to a `prov:location`
- basic subject in provenance (*may* have history, . . .)
- → `uws:parameter` are just provenance entities
- Usage (role) is defined in `prov:used`

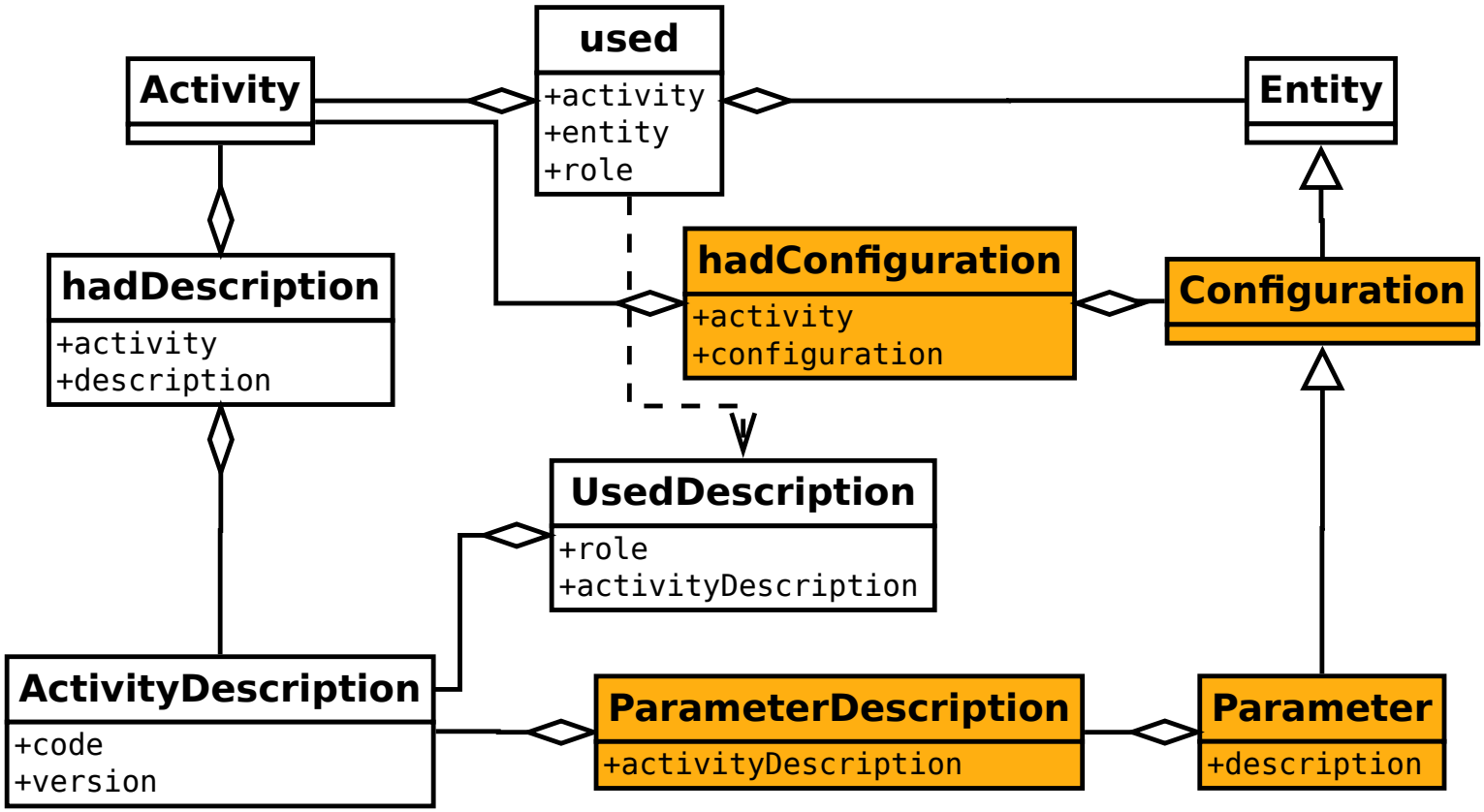
Entity/Parameter description



Entity/Parameter usage



Entity/Parameter usage description



Parameter limitations

- `Parameters` cannot be created independently of their usage
- `Parameters` cannot re-used by an `Activity` with a different `ActivityDescription`
- `Activity` cannot accept `Parameters` with different `ParameterDescriptions` for the same place

Choices for configuration values

- may be handled as a `Parameter`
- may be defined as `Parameter`, but used with a general `Entity` (*Which* `ParameterDescription` *is relevant then?*)
- may be handled like a general `Entity` (*proper class hierarchy missing then!*)

On implementation side, one has to choose between them! Clients and queries need to be aware of all three ways!

Simplified model for Parameters

- Do we need the **Parameter** term? As said, the corresponding term in provenance is **Entity**
- if yes: define `Parameter` as `Entity` that directly carry a value. No limitation to configuration.
- otherwise, just use general `Entity` instead
- Make `hadConfiguation` a true subclass of `used`

Special Entity and usage classes

- `MainEntity`, `Configuration`, `Context` and their subclasses
- describe *usage* of the entity; however usage is already handled by `prov:used` relation
- Removal of special classes removes redundancy
- Properties of the `Entity` should go to `EntityDescription`
- Classification (*what is configuration?*) is somehow arbitrary
- Usage roles go to ... `prov:role`

Limit to core model?

We could

- handle `Entity` with a value in the core
- extend the core with `prov:Plan` and handle `ActivityDescription`
- put the description of input and output into the `prov:Plan` description
- link to ObsCore DM (and other data models) instead of creating our own `EntityDescription`

and still solve most of our use cases.

- easy to understand
- simple to impement
- (relatively) simple to query

Limit to simplicistic model?

We could

- Restrict to main use case: how was my file generated?
- Just standardize what is carried out since years
- `Entity`, `Activity`, `used`, `wasGeneratedBy`
- simple to understand, impement, query
- 80/20 solution
- still extensible
- W3C