Repository Embedded Instances Design Notes

This page describes how *embedded instances* are to be managed by the repository. It does not say anything about other applications.

Contents

- Definition
- Behavior
    - Creation
    - Modification, Deletion
    - Dissemination
    - Harvest
    - Data Migration

The primary driver behind this design is the requirement by the data tools to have EIs edited with their parent in one atomic transaction.

The definition of an "instance" is effectively extended to cover its EIs as well as its direct property values.

Definition

An ***embedded instance*** (**EI**) is a resource instance which is hte value of an *object property* of another instance, its ***parent***. What sets it apart from other instances which are object values are these conditions:

1. Its **type** (possibly by inference) is in the class group

    ```
    http://eagle-i.org/ont/app/1.0/ClassGroup_embedded_class
    ```

2. ***Exactly* one parent**: There is *exactly one* instance which is the subject of statements for which the EI is the object. This is an informal restriction (really an *assumption*) imposed on all instances of embedded types by logic in the repository.
3. **No Orphaned or Shared EIs:** Any transaction which would result in an EI left without a parent, i.e. so it is ***not*** the object of any conforming statements ***or*** has multiple parents, is to be forbidden by logic in the repository.
4. ***No Broken Links to EIs***: If an EI is removed, an instance may not retain any statements of which it is the object. These must ***also*** be removed.

The model I'm imagining for EIs is that they are essentially *blank nodes with permanent names*. They behave like blank nodes in RDF, i.e. kind of anonymous, they only have URIs to make it easier to tell which one of a parent instance's EIs is to be changed in an `/update` operation. There is the additional restriction that any given EI is only connected at ONE place in the graph, which is technically not a restriction of blank nodes but is often the case in practice.&

Behavior

In general, EIs behave as if they are part of the parent. They are not subject to workflow and update actions by themselves, their contents may only be created or modified in the context of the parent.

EIs do not have any of their own administrative or provenance metadata. This means ordinary users cannot modify an EI, subject it to workflow transitions, etc.

Creation

An EI is created by adding a new URI with an appropriate `rdf:type` statement to a modification (including creation) of its parent. The type must belong to the embedded class group.

Modification, Deletion

Any modification of an EI ***must*** be done as a modification of its **parent**. The EI's properties, including type, may be changed; it may be deleted. These changes are recorded as a modification of the parent.

The changes to the parent and its EIs may be driven by one HTTP request to the `/update` service, and will be performed in a single transaction.

Dissemination

A dissemination request on the parent instance will include all of the statements about its EIs. The EIs will be filtered of hidden properties (e.g. admin data and contact hiding) by the same rules as the parent, and returned in the same serialized RDF graph.

Dissemination requests on EIs are not supported. It is not recommended, the results are undefined. In the first implementation in 1.1MS5, this only affects the repository's HTML interactive dissemination page, which is deprecated now anyway.

Harvest

EIs are not included in a `detail=identifier` harvest result.

When `detail=full`, the graph of each EI (filtered according to the same rules as the parent) is included in its parents description. The report can be expected to have the parent's statement whose object is the EI appear before any statements in which the EI is the subject.

Incremental reports: Since EIs do not have provenance metadata, **deletions of EIs are not reported** individually; only the change to parent gets reported.

Data Migration

Even before the 1.1MS5 release, several site repositories contain instances of types which belong to the Embedded Instance group and must change to the new behavior. The migration procedure updates the data already in the repository so it conforms to the new expectations. ***This is an essential step: Without migration, the repository will refuse to accept changes to some non-conforming instances.***

Migration will be implemented as part of the repository upgrade procedure in the `upgrade.sh` script. All external search indexes must be rebuilt after the migration. Since metadata is deleted, the incremental harvest will not return useful results.

Here are the steps to implement migration of EIs and their parent instances:

1. **Remove all repository metadata from EIs**. This includes provenance, for example, `dcterms:modified`, claims, workflow, etc. Each EI will inherit the metadata of its parent. The continued presence of metadata would cause false results from incremental `/harvest`.
2. **Find and eliminate orphans**. By "orphan", we mean any EI without a parent instance. They are summarily deleted. It seems cruel, but it is far kinder than leaving them to rot in the repository and cause questions and alarm later.

To find the orphans in your site repository, run this query against the **user-resources** view:

```
select distinct ?orphan where { ?orphan a ?eiType . ?eiType <http://eagle-i.org/ont/app/1.0/inClassGroup>
<http://eagle-i.org/ont/app/1.0/ClassGroup_embedded_class> graph :NG_Metadata {?orphan ?mdverb ?mdobj} optional
{ ?parent ?v ?orphan }

filter(!BOUND(?parent))}
```

1. **Make separate instances of any shared EIs**. There are many cases of an EI shared among 2 or more parents in the current data. This is no longer allowed. The solution is to delete the EI statement from all but one of the parents, and then replace it in the other ones with newly-minted copies of the EI. The copies are simply copies of all predicates and values of the original EI but on a new subject for each copy.