Revision handling

Joop Ringelberg 17-03-20 Version: 1

Introduction

Perspectives stores its data in Couchdb: PerspectContext and PerspectRol are the primary representation. However, we also store DomeinFiles and these in turn have some contexts and roles. Each of these 'entities'[1] are stored as JSON documents and Couchdb provides them with a Revision[2] in the form of an extra parameter (usually `_rev`).

When we want to overwrite a document, we have to provide its current revision string as a query parameter[3]. For sake of efficiency, we keep the revision in entities themselves. Obviously we have to keep that value up to date. This turns out to be complicated, given the asynchronous interaction with Couchdb.

Note: this text is **not** about *published versions* of a model.

Overview of the flow of entities and documents

Each type of entity can be cached and, its serialised version is stored as a document in Couchdb. In this sense, cache and Couchdb are both a *destination* and a *source*. However, we have more sources and destinations.

Sources

Besides cache and Couchdb, an entity can come from:

1. A creation statement, executed by the end user.
2. Parsing an Arc model (a DomeinFile is the serialisation of a model).
3. Parsing a CRL file (it is a syntactical representation of Context- and Role instances).
4. Starting to use a model.
5. The query that retrieves the external role of model descriptions from a repository.
6. A UniverseContextDelta or a UniverseRoleDelta, originating from creation statements executed by *other* users.

[1] We use 'entity' as the group name for the data instances that represent Perspectives runtime information. We use 'document' as the group name for the JSON-serialized version of an entity, in the context of Couchdb.
[2] Couchdb uses Revision. We distinghuish this from *version*. An entity may change over time and we might call these successive reincarnations *versions*.
[3] We can also provide it as the member _rev in the document itself. Even though we recognize that member in our definitions, we cannot use it in communicating with Couchdb. This is because we rely on Generic classes for serialising and this means an extra object layer around the resource itself. Hence, our modelled _rev member is buried and invisible for Couchdb.

7. A query on Couchdb (e.g. all instances of an Enumerated Role type).

Of these, only the last one gives us revisioned entities. Each other case has to be handled carefully.

Destinations

Contexts and Roles may be serialised as either Deltas, or JSON in a format that is suitable for exchange between PDR installations that are not connected (it is also used to create instances client side). None of these have revision information in them. Revisions will be re-established in the receiving PDR.

Why do PDR instances not share version information?

Many end users can share a context. So why don't their PDRs share revisions? The reason is that each user *may* have a different perspective on that context (or, for that matter, on roles, too). Obviously if the entities themselves are different, their serialisations must differ and hence we're talking about different documents altogether! Different documents have different revisions. This might be confusing, so let's summarize all kind of variants we've seen:

1. A *revision* applies to a Couchdb document (the serialised entity).
2. A *perspective* gives a user a unique variant of a context, depending on his role in it.
3. As a context (or role) evolves over time, we can say a user's perspective on it has different successive *versions* (and their serialisations have different revisions).

So, because the perspective on an entity has different successive versions, its serialisation will have different revisions. And because perspectives are unique, each user has a unique serialisation and hence revision for a context that all participants identify as the same!

Maintaining consistent revisions

First, we explain the revision handling between Couchdb and cache. Then we turn to the other sources of entities, explaining what will happen when we move an entity from such a source to cache (we never move an entity to Couchdb unless it is cached).

From cache to couch and back

We aim to keep the revision of an entity in cache equal to its serialisation in Couchdb. In moving entities in and out of Couchdb, we accomplish this by

- Storing the revision that comes from Couchdb in the entity in cache (all these entities must have a `Revision` instance; `Revision` is a class defined in the Couchdb package).

- Sending the revision in cache – if it exists! - as a query parameter to Couchdb.
- Putting the new revision string that comes from Couchdb after storing the serialisation, back in the entity in cache.

Creation

When an entity is created and stored in cache, it obviously has no revision. It has never been sent to Couchdb! This situation is easy and requires no special handling. The function `saveEntiteit` (or `saveEntiteit_`, a variant that takes both an id and an entity) handles this case by not sending a query parameter with revision information.

Parsing an Arc model

An Arc model is serialised as a DomeinFile. A DomeinFile has a `Revision` instance. A DomeinFile will be stored in Couchdb. Typically, parsing and storing an Arc model is an activity for the modelling user; end users just download DomeinFiles and start using them.

Obviously, a freshly parsed Arc model has no revision. However, the modeller may have parsed that file before and so there may be an equally identified DomeinFile in Couchdb with a revision!

Before moving such a parsed DomeinFile from cache to Couchdb, we have to update its revision by looking in Couchdb whether it exists and, if so, taking its revision. This is handled in the function `Perspectives.DomeinCache.saveCachedDomeinFile`.

Parsing a CRL file

Like parsing an Arc file, parsing a CRL file is a modellers' activity. Or, rather, it is a programmers activity. Implicitly, when parsing an Arc file, a modeller may parse a file with the same name (but with the extension .crl instead of .arc) with it, adding entities to the cache (Context- and Role instances).

Such entities may have been stored in Couchdb before, e.g. because the programmer had parsed the file before. Like with parsing an Arc model, we have to update the revisions of these entities in cache before saving them in Couchdb. This is handled in the function `Perspectives.LoadCRL.loadAndSaveCrlFile`.

Starting to use a model

A user may decide to start using a model; this involves downloading a DomeinFile from a repository. Later on, he can decide to no longer use that model. And then even later on, he may again start using the model. This entails that, if we again download the DomeinFile from the repository, we have to update its revision in cache before saving it (again) to Couchdb.

Moreover, a DomeinFile contains a number of Context- and Role instances. These have no revision (or, rather, their revisions need to be ignored). However, as a DomeinFile may be taken into use multiple successive times while these instances have not been removed. Also, the entities taken from the DomeinFile on a previous occasion may have been changed in the meantime. So we prefer the version in Couchdb over the version in the DomeinFile. This is handled in the function `Perspectives.Extern.Couchdb.addModelToLocalStore`.

Notice: we will need to do better if we want to update a model!

External model descriptions

In order to show the user the models that are available on a repository, we query it to send us a list of the external roles of the descriptions of these models. These role instances are cached. Now some of the available models may have been taken into use before and this means that the corresponding external role of its description is already in Couchb. If this is the case, we prefer the version that is in Couchdb over the one that comes from the repository. This is handled in the function `getExternalRoles` in package `Perspectives.Extern.Couchdb`.

Receiving a UniverseContextDelta or a UniverseRoleDelta

It probably may happen that we receive a UniverseContextDelta for a Context instance that is already in Couchdb. As of the time of writing of this document, this case has not yet been implemented.