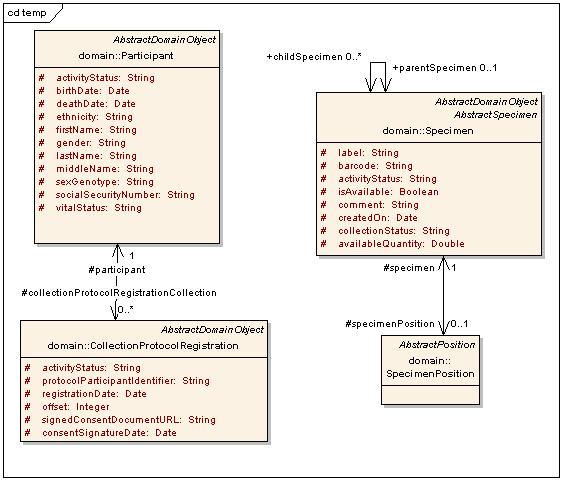
CQL 2 Query Related Issues

Please refer attached "model.jpg" for these queries. 

Note that first two queries are related to Specimen's self-association with the roleName "childSpecimen".

"Elaboration" specifies the query in an SQL-like language.

1. Specimens with more than 2 aliquots

elaboration: select \* from Specimen where (select count(\*) from

childSpecimen) > 2

From CQL2 schema perspective, the constraint (count > 2) is logically a part of the class CQLAssociatedObject.

This kind of functionality unfortunately makes the query language highly ambiguous and difficult to implement. Say you place additional query constraints after the count… are we counting objects first and then restricting further, or performing the restriction and counting the result? Nested counts get even uglier.

2. Specimens from which a DNA specimen was derived (recursive query)

Elaboration: select \* from Specimen where (exists a "derived"

specimen of type 'DNA')

Here, "derived" means we need to traverse the childSpecimen association to indefinite depth recursively (till we encounter a Specimen with type='DNA').

I’m not aware of any query languages which support this natively. One could “fake” it by (probably programmatically) building a CQL group using OR clauses and nesting associations deep enough to cover whatever level of recursion you might desire.

3. Temporal queries (cf

<http://www.cagrid.org/wiki/Data_Services:CQL2:Use_Cases#Temporal_Queries>):

Copied here for reference:

**Temporal Queries**

**From:** TBPT

**Overview:** Expression of query attributes in terms of ‘age’.

**Example:**

* A researcher would like to find all Samples in a data service that are 30 days old or newer.

**Currently:** While one could formulate such a query presently by doing something like “current day – 30”, the query cannot be saved for later re-use.

**Thoughts:** Such cases might be a modeling issue. Things like 'age' generally should not be stored in a database, since they change as time progresses. From a query perspective, we would need a way to make a query “relative to” some other value. This gets into the area of joins, which CQL doesn’t really do. In this case, the value is relative to today’s date, so it’s a known value and not really a join but a value replacement on the server side.

**Investigation:**

* Evaluate some TBPT models to see if anybody actually stores “age” values.

**Solutions:** Some tools around CQL [2] queries could be developed which finds and replaces values in attributes with values appropriate to the "age" desired.

a). Give me participants whose age > 30 years

elaboration: select \* from participant where <dateLiteral> - birthDate > 30Yrs.

b). Give me participants whose age at registration was over 30 years

elaboration:

select \*

from Participant p

join CollectionProtocolRegistration cpr

where cpr.registrationDate - p.birthDate > 30Yrs

I think both of these queries delve into the area of joins, which CQL has shied away from in the interest of being a “minimum barrier to entry” for getting data on the grid, regardless of the data source it comes out of. Query ‘a’ is certainly the ‘easier’ of the two, but it’s really just a special case of query ‘b’. Date and time processing is a non-trivial thing to implement.

4. Participants who are not registered for a collection protocol

elaboration:

select \* from participant

where identifier not in (select participantId from

CollectionProtocolRegistration)

Generally speaking, there are queries where we need to check for non-existence of an associated object. Another caTissue example is "Give me virtual specimens" (check for non-existence of associated SpecimenPosition).

Note that this is, in a way, a subset of use-case 1 above; the only difference is that the constraint is now (count = 0).

I don’t think it makes much sense here to implement the special case (count = 0) and not the general case. Questions like “how many aloquats are there for this sample?” might be better implemented in an analytical service where you could apply more fine grained rules.

This CAN be thought of as a special case of use-case 1. But that isn’t necessary. As you see, the SQL’s for the two can be built differently. Also, this case aims at an entirely different type of query from end-user perspective. (Also, queries of this type are apparently required more often).

From CQL schema perspective, this feature can be specified by having an optional boolean *not-exists* attribute as part of the *AssociatedObject* tag. This case is probably less ambiguous (if at all) than case 1 above. This is because “check no objects exist and then restrict further” doesn’t make sense. So it has to be “restrict first and check no objects exist