Historical Metadata: a deep-dive

One of the most common expectations we come across in the metadata landscape is the ability to access historical views of metadata.

The benefits of historical metadata appear in various use cases:

- being able to look back in time at data lineage, for example, to answer questions about a report or decision made months ago
- understanding what data was available at a given point in time, what was known about it

Addressing such use cases can be thought of as a simple matter of providing point-in-time inquiry. Egeria provides interfaces to make such point-in-time inquiries, and a native repository (based on Crux) to efficiently serve them.

However, the reality of temporality introduces some nuances of which we should be aware – particularly in the context of the distributed metadata landscape that Egeria enables.

Bi-temporality in regards to history: "validity" vs "transaction" time

The first place to delve deep is perhaps the conceptual: distinguishing what we really mean by "history" and recognising that there are in fact multiple dimensions involved. There are various excellent examples and articles already out there which I will not hazard to repeat, but suffice to say that it is important to distinguish between these concepts:

- *valid* time (sometimes called *effective time* or *actual time*)
- *transaction* time (sometimes called *record time*)

The first represents when something *really happened*, in the real world. For example, the Declaration of Independence was signed on July 4, 1776.

The latter represents when something was *known* (e.g. captured by a system). For example, King George III only learned of the Declaration of Independence about a month later (around August 10, 1776); however, America's Independence Day has always been recognised as July 4, not August 10.

In an ideal universe, we might hope that the two would be the same – but the real world is rarely so simple.

What we often need to do is capture what we believe to be the truth about reality (the *valid time*), but accept that we may need to retroactively change that based on new information. That new information could arrive at any point (*transaction time*), but needs to be able to change what we previously believed to be a historical truth. Being able to identify *both* dimensions allows us to know what reality was in the past (via *valid time*), but *also the point at which we learned about that reality* (which could have been some days, weeks, months or even years after the fact) (through the *transaction time*).

In Egeria, this comes to life through the distributed nature of the metadata landscape, for example:

- A new entity E is created in metadata collection X
- E is synchronised to another metadata collection (Y) in the cohort
- Y may receive update events from the cohort about E before it receives the initial creation request (or the updates themselves in mixed order)
- Yet we still want Y to capture the correct chronological state of metadata about E in its history (the *valid time*), despite the time at which the events arriving (*transaction time*) being both at a different time and also potentially in a different order

Reflecting the past, as best we can...

To cater for these differences in times, and the potential for out-of-order events and information, in Egeria we focus entirely on *valid time*:

- When storing metadata, we explicitly set the *valid time* for metadata based on the latest instance-embedded timestamp information (update time, if available; creation time if not). We could receive a payload on 1 January 2021, but if its update time specifies the metadata was changed on 23 December 1885, we will set the metadata change as valid from 23 December 1885. We leave the bi-temporal backing store (e.g. Crux) to capture the time at which we make such changes (*transaction time*).
- When retrieving or searching for metadata using a point-in-time timestamp, we explicitly do that retrieval based only on *valid time*.

This has a number of implications, not all of which may be immediately obvious:

- We can handle events arriving out-of-order, while still being able to accurately reflect each event's knowledge about the past – without them clobbering other (out-of-order) events' knowledge about the past.
- Our point-in-time inquiry will always give us a view of "what we know *now* about what the past looked like." We currently do not expose the transaction time, and therefore are currently *not* able to answer a more specific question along the lines of "what we knew *at time A* about what the past looked like *at time B.*" (In the case of Crux, we are actually storing this distinction, though, so if such an interface is important to you please get in touch!)
- This does mean that if we ask the same question about the same point in the past, we *may* get different answers, *depending on when we ask the question.*
    - On 1 January 2020, we ask "what did entity E look like on 2 February 1990?" - we get the response that entity E did not exist.
    - On 1 January 2021, we receive information about entity E being created on 3 March 1980.
    - (We receive no other information about anything happening to entity E in the meantime...)
    - On 4 April 2021, we ask *the same question*: "what did entity E look like on 2 February 1990?" - we get the details about entity E as of 3 March 1980.
- Furthermore, for reference copies we must accept that we may *never* know the entire detailed history of the reference copy. Our responses to a query can only ever contain the historical view of a piece of metadata as the repository itself understands that piece of metadata.
    - On 1 January 2020, we receive a reference copy for entity E with a creation time of 12 December 2019 and an update time of 1 January 2020.
    - Any historical query or retrieval for entity E between now and 1 January 2020 will give us the information received above for the reference copy.

    - However, any query or retrieval for entity E prior to 1 January 2020 (for example, 31 December 2019) can only tell us that entity E did not exist.
    - We can see from the first bullet that actually entity E *did* exist (it was created 12 December 2019); however, since the repository has no information about *what* that entity looked like between creation and update, we cannot accurately answer any query about it prior to the updated information we received.
    - Our best course of action then, while not *strictly* accurate in the real-world sense, is to answer queries related to times prior to 1 January 2020 with the *repository's* knowledge of that point in time: it does not know anything about the entity at that point in time, and therefore can only respond that it did not exist.

Other considerations

If you are thinking of embarking on creating your own historical repository to integrate with Egeria, you may want to think through some of these other nuances and implications that we came across when developing the Crux connector:

Classifications

Classifications are sort-of instances (extending InstanceAuditHeader, but not InstanceHeader), so you may want to store them as a unique kind of instance just like entity and relationship (this is the approach taken by the JanusGraph connector, for example).

However, classifications are only accessible through an entity – they cannot be retrieved independently – so you may also want to consider whether it is more optimal to store them directly on the entity to which they are inherently attached (as we have done with the Crux connector).

In either case, remember that the lifecycle of a classification (it's creation and update times, and versions) is still handled independently from that of the entity – and this could have an impact on history:

- We receive a reference copy entity at version 2 with an update time of 5
- We later receive the same reference copy entity (still at version 2, update time of 5) which now has a classification, whose creation time is 3
- At time 1 and time 2 the repository knows nothing about the entity: any query or retrieval for these points in time will indicate that the entity did not exist
- At time 3 and 4, even though the classification existed, we do not know anything about the entity: the classification is therefore "orphaned" at these points in time, and any query or retrieval for the entity (even based on classification) should indicate that it did not exist
- Only in response to queries or retrievals from time 5 onwards should we respond with details about the entity: and this should include the details of the entity *and* the classification

Instance re-identification

Egeria provides an interface for re-identification (changing the GUID) of instances as well – both entities and relationships. To implement these with regards to retaining history, you will need to make use of the `reIdentifiedFromGUID` property that was added to InstanceHeader in release 2.9:

- The re-identify method should first create a copy of the existing instance.
- On the copy, the method should set this new property to the old (existing) GUID of the instance.
- The method should set the GUID of this copy of the instance to the new GUID.
- The method should find all relationships that point to the original GUID, and update these to point to the new GUID.
- The method should then delete the original instance, and create the new (copied) instance.

In this way, a consumer can then still navigate through the history:

- Retrieve an instance by GUID, and its history. This should cover the referential integrity to any relationships at those historical points in time that referred to this entity based on this (new) GUID.
- This will only go back as far as the instance existed or was re-identified.
- If the instance has the `reIdentifiedFromGUID` property set, this will give its previous identifier.
- The consumer can use this previous identifier to retrieve that historical instance by its (different) GUID, and all of its history. This should also cover the referential integrity to any relationships at those historical points in time that referred to this entity based on that (different) previous GUID.
- This process can then recursively repeat, to navigate through any number of re-identifications of the instance.