written by Max Tomassi on 27 February 2014 in Atlas, engineering

the war of the models: bringing the peace

I recently wrote about the fun we were having while ingesting data from a new data source in Atlas, dealing with very large files and their encoding. However, when you are in front of a such a big and complex data source, the physical representation of the data is not the only problem: dealing with its structure can give you the same amount of fun, if not more!

your model into my model

As you probably know Atlas collects media metadata from many different sources, merges it, and then makes it available for its clients through RESTful APIs. Every data source has obviously a different data model, even if the domain is actually the same, or very similar. Atlas therefore has built its own model trying to abstract them and to be able to collect data from heterogeneous sources into a single place. Every time a new source comes, we need to find a way to map its own data model into the Atlas' one, and sometimes this is not truly straightforward.

your model doesn't fit mine

Sometimes obviously happens that there isn't an easy one to one mapping from the source model to Atlas' model, in that case you need to apply some kind of logic in order to make it happen. An example is the cardinality mismatch: let's say your model handles multiple types of descriptions for the same TV program, while my model only supports one description. Also, descriptions are optional in your model, I might receive a description for a specific type or not. Well, if we know exactly the list of description types we can define a priority and take the description we think its the most important. This works pretty well while we are ingesting the full data source with all the data available that allows us to take this kind of decisions. But what if we receive updates for single bits of the source model?

processing updates for a normalized source model

Assume that you're dealing with a relational source data model and that what you need to ingest is a dump of this data model. In the first instance you will ingest the entire dump in one go, receiving all the available data. Suppose that the list of descriptions for a program are stored into a dedicated table. Fine, you receive all the records, pick the most important one and store it in your model.

From now on, though, you will start receiving daily updates saying that for each table some records have been created, some have been updated (and you'll receive the current status of the record), some have been deleted. Now you'll start having some problems. For example, if one of the descriptions changes, you'll receive the snapshot of the updated record, but wait, my model has only one description, how can I know if the one that has been updated is the one that I actually have stored? Obviously you can't compare the text :)

possible ways to go

As we've seen processing and ingesting updates as soon as they arrive doesn't work well if source and target models don't match properly. In this case you need to take a decision. We think there are two possible ways to go:

- Modify your own model in order to be more similar to the source one and defining a one to one map. For example if you extend your model for handling multiple descriptions introducing the concept of description type, you now can cope with single updates easily
- Store the full data source in your environment and keep it update with the daily changes you receive. In this case you'll have an exact copy of the source database and you'll be able to replay the mapping logic whenever you want, having the full data set always available.

Obviously, the best way to go depends on the implementation cost.

how do you deal with complex data sources?

I tried to extract some of the thoughts we had while ingesting data from a complex source and I hope this can be helpful if you're dealing with a similar requirement. As usual, we'd love to also hear your experience, so feel free to leave a comment below or tweet us.