TopicMap Merge Platform

Latest: 20170611
In progress...

Background

A merge platform for the TQElasticKnowledgeSystem

Present Flows

Here is a trace of flows for a new topic entering the database:

- `ITQDataProvider.putNode` persists the node to the topic map.
- If no persistence errors
    - The new topic is sent to `MergeInterceptor.acceptNodeForMerge`
    - MergeInterceptor then queues that topic to be processed in worker thread.
    - Worker Thread performs two processes
        - The topic is sent to SameLabelDetector.acceptProxy

        - There, the proxy is subjected to Same Label analysis
            - This code simply builds and maintains a file called `SameLabel.json` which keeps track of all labels and the locators to which they belong. Labels are turned to lower-case before processing.
                - External agents read that file and process it,looking for merges
    - If a “MergeListenerPropagate” flag is set in the topicmap-properties.xml file
        - MergeInterceptor will send the topic as a JSON string out a specific TCP stocket to any listeners
        - NOTE: the TCP socket is blocking if there are no listeners
        - TODO: change this to a KafkaBackside source

Planned Flows

<INTENT>To map out the entire Kafka-based agent ecosystem as relates specifically to merge operations and topic map maintenance</INTENT>

Merge Interaction Use Cases

Automatic

MergeAgent detection by way of any agent which detects a merge and fires merge behaviors.

Manual

This is the *social* approach to merging, where some *user interface affordances* allow users to suggest merges with reasons given.

That process also includes *unmerge* suggestions, where a user makes the case that the topics should not have been merged

Merge Use Cases

Same Label

This is a very hard scenario. Simply having the same label does not mean that two topics are about the same subject. Especially hard in the case of humans.

Disambiguation requires locating and identifying non-subject-identity properties, and/or relationships.

Many same-label topics will exist in the topic map; over time, new things will be learned that allow some of those to be merged.

Same URL

By convention, if a topic is *about* a particular URL, then it has a particular subject identity property which is the “url” key. No two topics can have the same subject identity properties, so any topic which contains the same “url” value as any others is an automatic merge.

Same URI/PSI

Many *semantic web* objects have URIs given to them. Some objects, in fact, have several URIs for the same subject -- typically granted by different agencies.

In topic maps, we use the PSI key for the same purpose. Typically, a topic crafted solely within a topic map ecosystem will have just one and only one PSI. But, as we begin to import topics from the semantic web ecosystem, such as climate and biomedical ontologies, and when we look topics up at DbPedia, we will create topics which have PSI collections.

By definition, a PSI is a subject identity property. Anytime two topics with the same PSI are detected, that’s an automatic merge..

Merge Architecture

Processes

Flows

- A proxy (topic) is sent to `IMergeThread`
- `MergeThread` is threaded: it sends proxies to `IMergeEngine`
- For each `IMergeAgent` known to `MergeEngine`
    - Ask if that agent is appropriate to the given proxy
    - If appropriate
        - The proxy is handed to that `IMergeAgent`
            - If a merge is found
                - Merge is performed

Design Issues

Merging is a *threaded* process. The `MergeAgent` cannot know if a merge succeeded after a given proxy is passed to a given `IMergeAgent` at the moment it is passed; `MergeAgent` relies on the results of that merge passed back as an `IMergeResultListener`. This means that `MergeAgent` must:

- Cache the proxy together with the index of the last `IMergeAgent` tested
- Wait for an `IMergeResultListener` event for that proxy
    - If merge occurred, remove the proxy from the cache
    - Otherwise, restart merge testing at the next `IMergeAgent`
        - If no further merge agents to test
            - No merge occurs
- It's easy to code for that.
- The issue lies in scalability of that; just how many proxies can you cache before running out of memory?
- Parallel merging is a related issue
    - Parallelize merging to use different JVMs
    - Must coordinate with each other
        - No proxy can arrive at multiple `MergeEngine` instances
- A solution is this
    - IMergeThread can talk to more than one IMergeEngine
        - Send proxies to them in round-robin fashion
    - This means that IMergeEngines must, instead of being locally constructed (same JVM), must exist out on the Kafka ecosystem
        - Distributed merging through Kafka streams

`IMergeAgent`

A specification for merge agents which perform specific kinds of merges. Examples are same-label detection, same-url detection, etc.

`IMergeEngine`

Extends IMergeResultListener
<new>

`IMergeImplementation`

`IMergeResultListener`

`IMergeThread`

<new>

`ISubjectProxy`

`ISubjectProxyModel`

Classes

`BaseVirtualizer`

`DefaultVirtualizer`

`KafkaProducer`

This class is the gateway to a Kafka stream-based ecosystem. It's method:

```
kafkaProducer.sendMessage(String message, Integer partition)
```

accepts a JSON-formatted string of this form:

```
{
      "verb": <some verb>,
      "cargo": <the cargo to be sent>
}
```

`MergeBean`

A class which performs VirtualMerge operations.

`MergeEngine`

<new>
Implements the `IMergeEngine` interface.
An entry point for merge operations on a given topic object. This class will exercise a list of

`MergeInterceptor`

`MergeThread`

Implements `IMergeThread`
<new>

`SameLabelDetector`

`SameLabelMergeHandler`

`SubjectProxy`

`SubjectProxyModel`

`VirtualizerHandler`

SPARQL Endpoints

Merging is going to entail taxonomic resolution: two classes with the same label might be the same class; one approach is to check the taxonomic super class, which might recurse up the line. An approach to resolving this, particularly with topics which are domain ontologies elsewhere, is to use a Sparql Endpoint and run an OWL:SameAs query.

RDF4J

Current work aims to craft a massive local Sparql Endpoint with RDF4J[1]. The plan is to load it with many ontologies and knowledge bases, including DbPedia, WikiData, and varieties of ontologies.

RDF4J-Server

RDF4J-Workbench

tq-rdf-importer

A stand-alone project which reads directories populated with rdf ontology and knowledgebase files and submits them to repositories in RDF4J-Server

tq-rdf-client

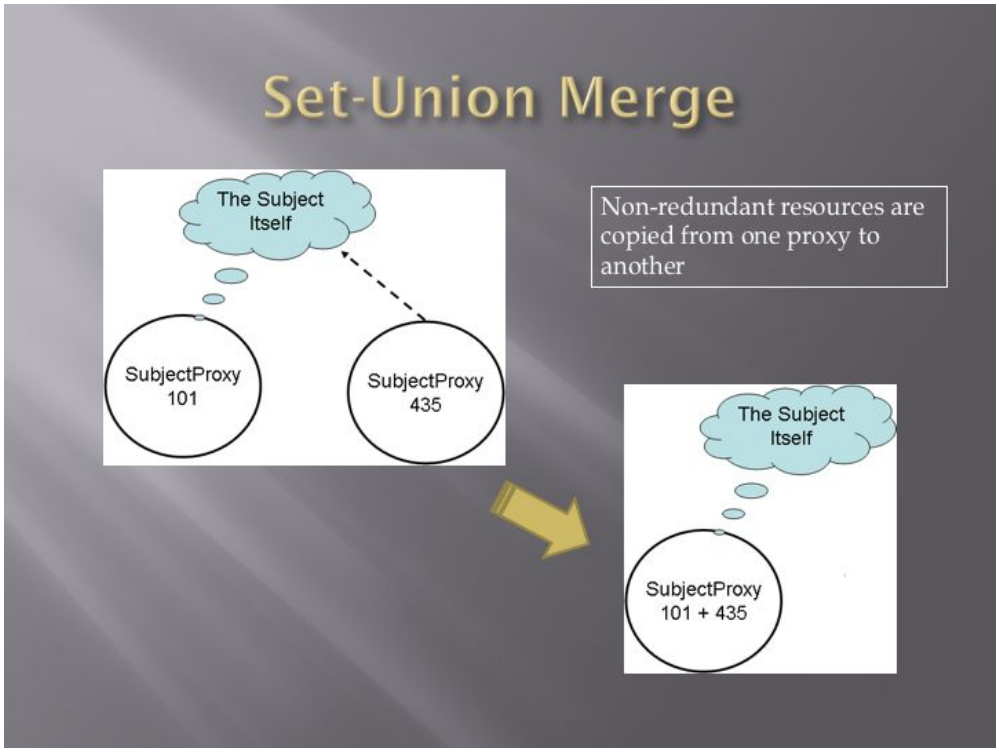
An embedded client which provides access to RDF4J-Server

[1] RDF4J: http://rdf4j.org/

Glossary

Merge

Bringing two topics together when, and only when they are *about* the same subject. A simple merge is a *set-union* combination of two topics into one.



MergeAssertion

A SubjectProxy which is an instance of a `MergeRelationType`, which means this proxy serves as a relation (a predicate) which connects a topic to a VirtualProxy. This relation will include a *biography* which is the reasons for the merge.

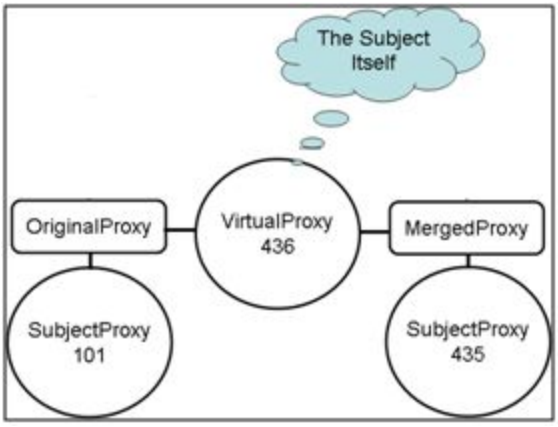
SubjectProxy

UnMergeAssertion

Like a MergeAssertion, except that it is an instance of `UnMergeRelationType`, complete with a *biography* which tells why the two proxies are no-longer merged. A reason for keeping merge connections around rather than deleting them is to maintain awareness of a topic's history. NOTE: it is possible that we might not use UnMergeAssertion in favor of updating the biography of a MergeAssertion. That remains to be seen.

Virtual Merge

Reference [2] summarizes the concept. Basically, a simple merge combines two objects into one. The downside of doing that is experienced if you need to *unmerge* -- break the single object back to its original objects. A virtual merge creates a VirtualProxy. That proxy serves the same function as a simple merge, but the original topics remain and are linked together with the VirtualProxy by way of a MergeAssertion.



VirtualProxy

TODO

- CONSIDER Distributed Merge
    - IMergeThread feeds the Kafka ecosystem with a merge message
        - Similar to the topic map feeding Kafka with a new topic to examine
        - One thought
            - No more IMergeThread
            - IMergeEngine lives on a Kafka consumer
                - When it is ready, it takes from the persistent Kafka stream for that topic and partition
            - Greatly simplifies the entire architecture
            - BUT
                - Greatly complexifies it for systems which are too small for a Kafka ecosystem
                    - Those can live with what we are implementing now (20170521)

Notes (Blog)

20170611 JP

- Installed RDF4J
    - Can load RDF4J server with owl, ttl, etc files by way of
        - RDF4J-Workbench (file size limits?)
        - Tq-rdf-importer
    - Owl:SameAs queries
        - Seem to work with a HttpURLClient
        - Not (yet) working with tq-rdf-client
- Continuing development of json-proxy-reader
    - Takes proxy definitions from large JSON files
    - Imports them into the topic map
    - Begins the process of *SameLabelMerge*
        - Taxonomic evaluation of Same-label proxies
            - SameLabelTaxonomicAgent
                - Performs taxonomic comparisons
            - OwlSameAsUtility
                - Performs Owl:SameAs queries to RDF4J-Server to support SameLabel evaluation

20170602 JP

- We now have two biotech ontologies imported
    - Large JSONObject of hits
        - >1 locator per label
- These are biomed
    - Implication being some are simply trivial not-mergable
    - Implication being some are same topic, different namespace
- Same Topic, Different Namespace
    - Implies must ripple up transitive closure to some stopping point
        - StoppingPoint
            - Some common class
                - SubOf ImportedOntologyType
    - Ultimately, must merge top down in the transitive closure
- POSSIBLE HELP:
    - Query DBPedia
        - Gather all namespaces for the class
        - Create a VirtualProxy from that
        - Merge against that
    - Setting up Fuseki as a SPARQL end point with DBPedia, etc
        - WILL borrow SPARQL code from YodaQA

20170521 JP

- MajorToM topic map merge engine discovered here:
    - https://code.google.com/archive/p/majortom/
    - Java Apache2
    - Paper was at TMRA2010

20170520 JP

- Starting to dive deeply into the merge platform in TQ-Elastic-KS
    - Noted that agent propagation of a proxy for external operations is turned off since the TCP socket blocks if nobody is listening
    - TIME to bring in KafkaBackside
        - Right now, it is not a Maven project
            - TODO
                - Mavenize KafkaBackside
                    - DONE
                    - Alec to add Docker
                - Move it from the knowledgegarden repo to the topicquests repo
                    - SOON
        - TQElasticKS pom.xml now reflects KafkaBackside
            - Added the class KafkaProducer
            - Now beginning to modify the merge flow to not use a TCP socket but instead use that new class
            - MergeInterceptor modified to sent the JSONObject of the new proxy with the verb “new” to the Kafka ecosystem
    - Borrowed the IMergeAgent class from the SolrSherlock codebase
        - It’s a root entry point which accepts a proxy in the forms
            - Locator string (to go fetch)
            - JSONObject
            - ISubjectProxy
        - It’s task is to then fire up a series of tests which examine the new proxy along many dimensions, all of which seek to compare subject identities with other topics in the topic map
        - If subject identities match, then perform a merge operation

References

[1] Park, Jack (2013). “Topic maps need merge engines”. Online at http://debategraph.org/Details.aspx?nid=247114

[2] Park, Jack (2010). “Topic merge scenarios for knowledge federation”. In: Maicher, Lutz and Garshol, Lars Marius eds. *Information Wants to be a Topic Map: Revised Selected Papers.* Leipzig: Universit¨at Leipzig, pp. 143–154. Online at https://www.researchgate.net/profile/Jack_Park/publication/48989667_Topic_merge_scenarios_for_knowledge_federation/links/0fcfd5117d010bff88000000.pdf & http://oro.open.ac.uk/23944/1/Park-TMRA2010.pdf

- Slides: https://www.slideshare.net/tmra/tmra2010-park

[3] Park, Jack (2013). “Topic Map Merge Platform”. Online at http://blackheathphilosophy.org/Details.aspx?nid=257147

[4] Bleier, Arnim, Patrick Jahnichen, Uta Schulze, and Lutz Maicher (2010). “The Praxis of Social Knowledge Federation”. In Dino Karabeg and Jack Park (Eds). *Knowledge Federation 2010 Self-Organizing Collective Mind: Second International Workshop on Knowledge Federation*, Dubrovnik, Croatia, October 3-6, 2010.
[5] Hoyer, Marcel (2011). “Scalability of Topic Map Systems: Topic Map Services in a Distributed Environment”. Master's Thesis. Online at http://lips.informatik.uni-leipzig.de/files/index.pdf
[6] Schulz, Uta (2010). “Hatana, a virtual merging engine”. Slides online at https://www.slideshare.net/tmra/hatana-virtual-topic-map-merging
[7]