High-Performance Annotation Tagging over Solr Full-text Indexes

Michele Artini, Claudio Atzori, Sandro La Bruzzo, Paolo Manghi, Marko Mikulicic, and Alessia Bardi

ABSTRACT

In this work, we focus on the problem of annotation tagging over information spaces of objects stored in a full-text index. In such a scenario, data curators assign tags to objects with the purpose of classification, while generic end users will perceive tags as searchable and browsable object properties. To carry out their activities, data curators need annotation tagging tools that allow them to bulk tag or untag large sets of objects in temporary work sessions where they can virtually and in real time experiment with the effect of their actions before making the changes visible to end users. The implementation of these tools over full-text indexes is a challenge because bulk object updates in this context are far from being real-time and in critical cases may slow down index performance. We devised TagTick, a tool that offers to data curators a fully functional annotation tagging environment over the full-text index Apache Solr, regarded as a de facto standard in this area. TagTick consists of a TagTick Virtualizer module, which extends the API of Solr to support real-time, virtual, bulk-tagging operations, and a TagTick User Interface module, which offers end-user functionalities for annotation tagging. The tool scales optimally with the number and size of bulk tag operations without compromising the index performance.

INTRODUCTION

Tags are generally conceived as nonhierarchical terms (or keywords) assigned to an information object (e.g., a digital image, a document, a metadata record) in order to enrich its description beyond the one provided by object properties. The enrichment is intended to improve the way end users (or machines) can search, browse, evaluate, and select the objects they are looking for. Examples are qualificative terms, i.e. terms associating the object to a class (e.g., biology, computer science, literature) or qualitative terms, i.e. terms associating the object to a given measure of value (e.g., rank in a range, opinion).[1] Approaches differ in the way tags are generated. In some cases users (or machines)[2] freely and collaboratively produce tags,[3] thereby generating so-called

Michele Artini (michele.artini@isti.cnr), **Claudio Atzori** (claudio.atzori@isti.cnr.it), **Sandro La Bruzzo** (msandro.labruzzo@isti.cnr), **Paolo Manghi** (paolo.manghi@iti.cnr.it), and **Mark Mikulicic** (mmark.mikulicic@isti.cnr.it) are researchers at Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo," Consiglio Nazionale delle Richerce, Pisa, Italy. **Alessia Bardi** (mallessia.bardi@for.unipit.it) is a researcher at the Dipartimento di Ingegneria dell'Informazione, Università di Pisa, Italy.

folksonomies. The natural heterogeneity of folksonomies calls for solutions to harmonise and make more effective their usage, such as tag clouds.[4] In other approaches users can pick tags from a given set of values (e.g., vocabulary, ontology, range) or else find hybrid solutions, where a degree of freedom is still permitted.[5,6] A further differentiation is introduced by *semantically enriched tags*, which are tags contextualized by a label or prefix that provides an interpretation for the tag.[7] For example, in the digital library world, the annotation of scientific article objects with subject tags could be done according to the tag values of the tag interpretations of ACM scientific disciplines and "Dewey Decimal Classification," whose term ontologies are different.[8]

The action of tagging is commonly intended as the practice of end users or machines of assigning or removing tags to the objects of an *information space.* An information space is a digital space a user community populates with information objects for the purpose of enabling content sharing and providing integrated access to different but related collections of information objects.[9] The effect of tagging information objects in an information space may be private, i.e., visible to the users who tagged the objects or to a group of users sharing the same right, or public, i.e., visible to all users.[10] Many well-known websites allow end users to tag web resources. For example Delicious[11] (http://delicious.com) allows users to tag web links with free and public keywords; Stack Overflow (http://stackoverflow.com), which lets users ask and answer questions about programming, allows tagging of question threads with free and public keywords; Gmail[12] (http://mail.gmail.com) allows users to tag emails—at the same time, tags are also transparently used to encode email folders. In the digital library context, the portal Europeana (http://www.europeana.eu) allows authenticated end users to tag metadata records with free keywords to create a private set of annotations.

In this work we shall focus on *annotation tagging*—that is, tagging used as a manual data curation technique to classify (i.e., attach semantics to) the objects of an information space. In such a scenario, tags are defined as controlled vocabularies whose purpose is classification.[13,14] Unlike semantic annotation scenarios, where semantic tags may be semiautomatically generated and assigned to objects,[15] in annotation tagging authorized data curators are equipped with search tools to identify the sets of objects they believe should belong or not belong to a given category (identified by a tag), and to eventually perform the tagging or untagging actions required to apply the intended classification. In general, such operations may assign or remove tags to and from an arbitrarily large subset of objects of the Information Space. It is therefore hard to predict the quality and consistency of the combined effect of a number of such actions. As a consequence, data curators must rely on *virtual tagging* functionalities which allow them to bulk (un)tag sets of objects in temporary work sessions, where they can in real-time preview and experiment (do/undo) the effects of their actions before making the changes visible to end users. Examples of scenarios that may require annotation tagging can be found in many fields of application. This is the case, for example, in several data infrastructures funded by the European Commission FP7 program, which share the common goal of populating very large information spaces by aggregating textual metadata records collected from several data sources. Examples are the data

infrastructures for *DRIVER*,[16] *Heritage of the People's Europe (HOPE)*,[17] *European Film Gateway (EFG* and *EFG1914)*,[18] *OpenAIRE*[19] (http://www.openaire.eu), and Europeana. In such contexts, the aggregated records are potentially heterogeneous, not sharing common classification schemes, and annotation tagging becomes a powerful mean to make the Information Space more effectively consumable by end users.

There at two significant challenges to be tackled in the realization of annotation tagging tools. First is the need to support bulk-tagging actions in almost real time so that data curators need not wait long for their actions to complete. Second, bulk-tagging actions need to be virtualized over the information space, so that data curators can verify the quality of their actions before committing them, and access to the information space is unaffected by such actions. Naturally, the feasibility and quality of annotation tagging tools strictly depends on the data management system adopted to index and search objects of the information space. In general, not to compromise information space availability, bulk-updates are based on offline, efficient strategies, which minimize the update's delay,[20] or virtualisation techniques, which perform the update in such a way that users have the impression this was completed.[21]

In this work, we target the specific problem of annotation tagging of information spaces whose objects are documents in a Solr full-text index (v3.6).[22] Solr is an open-source Apache project delivering a full-text index whose instances are capable of scaling up to millions of records, can benefit from horizontal clustering, replica handling, and production-quality performance for concurrent queries and bulk updates. The index is widely adopted in the literature and often in contexts where annotation tagging is required, such as the aforementioned aggregative data infrastructures. The implementation of virtual and bulk-tagging facilities over Solr information spaces is a challenge, since bulk updates of Solr objects are fast, but far from being real-time when large sets of objects are involved. In general, independently of the configuration, a re-indexing of millions of objects may take up to some hours, while for real-time previews even minutes would not be acceptable. Moreover, in critical cases, update actions may also slow down index performance and compromise access to the information space.

In this paper, we present TagTick, a tool that implements facilities for annotation tagging over Solr with no remarkable degradation of performances with respect to the original index. TagTick consists of two main modules: the TagTick Virtualizer, which implements functionalities for real-time bulk (un)tagging in the context of work sessions for Solr, and the TagTick User Interface, which implements user interfaces for data curators to create, operate and commit work sessions, so as to produce newly tagged information spaces. TagTick software can be demoed and downloaded from http://nemis.isti.cnr.it/product/tagtick-authoritative-tagging-apache-solr.

ANNOTATION TAGGING

Annotation tagging is a process operated by data curators whose aim is improving the end user's search experience over an Information Space. Specifically, the activity consists in assigning searchable and browsable tags to objects in order to classify and logically structure the

Information Space into further (and possibly overlapping) meta-classes of objects. Moreover, when ontologies published on the Web are used, for example ontologies available as linked data such as the GeoNames ontology (http://www.geonames.org/ontology/documentation.html) or the DBPedia ontology (http://dbpedia.org/Ontology), then tags are means to link objects in the information space to external resources. In this section, we shall describe the functional requirements of annotation tagging in order to introduce assumptions and nomenclature to be used in the remainder of the paper.

Information Space: Objects, Classes, Tags, and Queries

We define an information space as a set of objects of different classes $C_1 \ldots C_k$. Each class C_i has a structure $(l_1 : V_1, \ldots, l_n : V_n)$, where l_j's are object property labels and V_j are the types of the property values. Types can be value domains, such as strings, integers, dates, or controlled vocabularies of terms.

In its general definition, annotation tagging has to do with semantically enriched tagging, where a tag consists of a pair *(i, t)*, made of a *tag* interpretation *i* and a *tag* value *t* from a term ontology *T*; as an example of interpretation consider the ACM subject classification scheme (e.g., *i = ACM*), where *T* is the set of ACM terms.

In this context, tagging is de-coupled from the Information Space and can be configured a-posteriori. Typically, given an Information Space, data curators set up the annotation tagging environment by: (*i*) defining the interpretation/ontology pairs to be used for classification, and (*ii*) assigning to each class *C* the interpretations to be used to tag its objects. As a result, class structures are enriched with a set of interpretations $(i_1{:}T_1 \ldots i_m{:}T_m)$, where i_j are tag interpretation labels and T_j the relative ontologies. Unless differently specified, an object may be assigned multiple tag values for the same tag interpretation, e.g. scientific publication objects may cover different scientific ACM disciplines.

Finally, the information space can reply to queries *q* formed according to the abstract syntax intable 1, where *Op* is a generic Boolean operator (dependent on the underlying data management system, e.g. "=," "<," ">") and $C \in \{C_1, \ldots, C_k\}$. Tag predicates *(i = t)* and class predicates *(class = C)* represent exact matches, which mean "the object is tagged with the tag *(i, t)*" and "the object belongs to class *C*."

q ::=(q And q)
| (q Or q)
| (l Op v)
| (i = t)
| (class = C)
| v
| ε

Table 1. Solr Query Language.

Virtual and Real-time Tagging

In annotation tagging data curators apply bulk (un)tagging actions with respect to a tag *(i, t)* over arbitrarily large sets of objects returned by queries q over the information space. Due to the potential impact that such operations may have over the information space, tools for annotation tagging should allow data curators to perform their actions in a protected environment called work session. In such an environment curators can test sequences of bulk (un)tagging actions and incrementally shape up an information space preview: they may view the history of such actions, undo some of them, add new actions, and pose queries to test the quality of their actions. To offer a usable annotation tagging tool, it is mandatory for such actions to be performed in (almost) real-time. For example, curators should not wait more than a few seconds to test the result of tagging 1 million objects, an action which they might undo immediately after. Moreover, such actions should not conflict (e.g., slow performance) with the activities of end users running queries on the information space. Finally, when data curators believe the preview has reached its maturity, they can commit the work session, i.e., materialise the preview in the information space, and make the changes visible to end users.

APACHE SOLR AND ANNOTATION TAGGING

As mentioned in the introduction, our focus is on annotation tagging for Apache Solr (v3.6). This section describes the main information space features and functionalities of the Solr full-text index search platform. In particular, it explains the issues arising when using its native APIs to implement bulk real-time tagging as described previously.

Solr Information Spaces: Objects, Classes, Tags, and Queries

Solr is one of the most popular full-text indexes. It is an Apache open source Java project that offers a scalable, high performance and cross-platform solution for efficient indexing and querying of information spaces made of millions of objects (*documents* in Solr jargon).[23] A Solr index stores a set of objects, each consisting in a flat list of possibly replicated and unordered fields associated to a value. Each object is referable by a unique identifier generated by the index at indexing time.

The information spaces described previously can be modelled straightforwardly in Solr. Each object in the index contains field-value pairs relative to the properties and tag interpretations of all classes they belong to. Moreover, we shall assume that all objects share one field named *class* whose values indicate the classes (e.g. $C_1, \ldots, C_k$) to which the object belongs. Such an assumption does not restrict the application domain, since classes are typically encoded in Solr by a dedicated field.

The Solr API provides methods to search objects by general keywords, field values, field ranges, fuzzy terms and other advanced search options, plus methods for the bulk addition and deletion of objects. In our study, we shall restrict to the search method *query(q, q_f)*, where q and q_f are CQL queries respectively referred as the "main query" and the "filter query". In particular, in order to match the query language requirements described previously, we shall assume that q and q_f are

expressed according to the CQL subset matching the query language in table 1.

getDocset :RS→DS	returns the docset relative to a result set
intersectDocsets :DS × DS→DS	returns the intersection of two docset
intersectSize :DS × DS→Integer	returns the size of the intersection of two docsets
unifyDocsets :DS × DS→DS	returns the union of two docsets
andNotDocsets :DS × DS→DS	given two docsets ds1 and ds2 returns the docset $\{d \mid d \in ds1 \wedge \neg d \in ds2\}$
searchOnDocsets :Q × DS→RS	executes a query *q* over a docset and returns the relative resultset

Table 2. Solr Docset Management Low-Level Interface.

To describe the semantics of *query(q, q_f)* it is important to make a distinction between the Solr notions of *result set* and *docset*. In Solr, the execution of a query returns a result set (i.e., *QueryResponse* in Solr jargon) that logically contains all objects matching the query. In practice, a result set is conceived to be returned at the time of execution to offer instant access to the query result, which is meantime computed and stored in memory into a low-level Solr data structure called *docset*. Docsets are internal Solr data structures, which contain lists of object identifiers and allow for efficient operations such as union and intersection of very large sets of objects to optimize query execution. Table 1 illustrates some of the methods used internally by Solr to handle docsets. Method names have been chosen to be self-explanatory and therefore do not match the ones used in the libraries of Solr.

$$[\![query(q,q_f)]\!]_{Solr} = \begin{cases} \{d/ID(d) \in [\![q]\!]_{DS}\} & if\ (q_f = null) \\ searchOnDocset(q, [\![q_f]\!]_{Cache(\phi)}) & if\ (q_f \neq null) \end{cases}$$

$$[\![q_f]\!]_{Cache(\phi)} = \begin{cases} ds & if(\phi(q_f) = ds) \\ [\![q_f]\!]_{Cache(\phi[q_f \leftarrow [\![q_f]\!]_{DS}])} & if(\phi(q_f) = \perp) \end{cases}$$

$$[\![(q_1\ And\ q_2)]\!]_{DS} = [\![q_1]\!]_{DS} \cap [\![q_2]\!]_{DS}$$

$$[\![(q_1\ Or\ q_2)]\!]_{DS} = [\![q_1]\!]_{DS} \cup [\![q_2]\!]_{DS}$$

$$[\![(l\ op\ v)]\!]_{DS} = \{ID(d) / d.l\ op\ v\}$$

$$[\![(i=t)]\!]_{DS} = \{ID(d) / d.i\ op\ t\}$$

Table 3. Semantic Functions.

Informally, *query(q, q_f)* returns the result set of objects matching the query *q* intersected with the objects matching the filter query q_f, i.e. its semantics is equivalent to the one of the command *query(q And q_f, null)*. In practice, the usage of a filter query q_f is intended to efficiently reduce the scope of *q* to the set of objects whose identifiers are in the docset of q_f. To this aim, Solr keeps in memory a filter cache $\phi: Q \rightarrow DS$. The first time a filter query q_f is received, Solr executes it and stores the relative docset *ds* in ϕ, where it can be accessed to optimize the execution of *query(q, q_f)*. Once the docset $\phi(q_f) = ds$ is available, *query(q, q_f)* invokes the low-level method *searchOnDocset(q, ds)* (see table 1). The method executes *q* to obtain its docset, efficiently intersects such docset with *ds*, and populates the result set relative to the query. Due to the efficiency of docset intersection and in-memory data structures, query execution time is closely limited to the one necessary to execute *q*. Table 3 shows the semantic functions $[\![.]\!]_{Solr} : Q \times Q \rightarrow RS$, $[\![.]\!]_{DS} : Q \rightarrow DS$, $[\![.]\!]_{Cache} : Q \times \wp(Q \times DS) \rightarrow DS$.

The first yields the result set of *query(q, q_f);* the second the docset relative to a query *q* (where *d* is an object); and the third resolves queries into docsets by means of a filter cache ϕ.

Limits to Virtual and Real-Time Tagging in Solr

Whilst Solr is a well-known and established solution for full-text indexing over very large information spaces, it poses challenges for higher-level applications willing to expose to users private, modifiable views of the same index. This is the case for annotation tagging tools, which must provide data curators with work sessions where they can update with tagging and untagging actions a logical view of the information space, while still providing end users with search facilities over the last committed Information Space. Since Solr API does not natively provide "view management" primitives, the only approach would be that of materializing tagging and untagging

actions in the index while making sure that such changes are not visible to end users. Prefixing tags with work session identifiers, cloning of tagged objects, or keeping index replicas may be valuable techniques, but all fail to deliver the real-time requirement described previously. This is due to the fact that when very large sets of objects are involved the re-indexing phase is generally far from being real-time. In general, independently of the configuration, processing such requests may take up to some hours for millions of objects, while for real-time previews even minutes would not be acceptable.

TAGTICK VIRTUALIZER: VIRTUAL REAL-TIME TAGGING FOR SOLR

This section presents the TagTick Virtualizer module, as the solution devised to overcome the inability of Apache Solr to support out-of-the-box real-time virtual views over Information Spaces. The Virtualizer API, shown in table 4, supports methods for creating, deleting and committing work sessions, and, in the context of a work session: (1) performing tagging/untagging actions and (2) querying the information space modified by such actions. In the following we will describe both functional semantics and implementation of the API, given in terms of a formal symbolic notation. The semantics defines the expected behaviour of the API and is provided in terms of the semantics of Solr. The implementation defines the realization of the API methods in terms of the low-level docset management library of Solr. The right side of figure 1 illustrates the layering of functionalities required to implement the TagTick Virtualizer module. As shown, the realization of the module required exposing the Solr low-level docset library through an API.

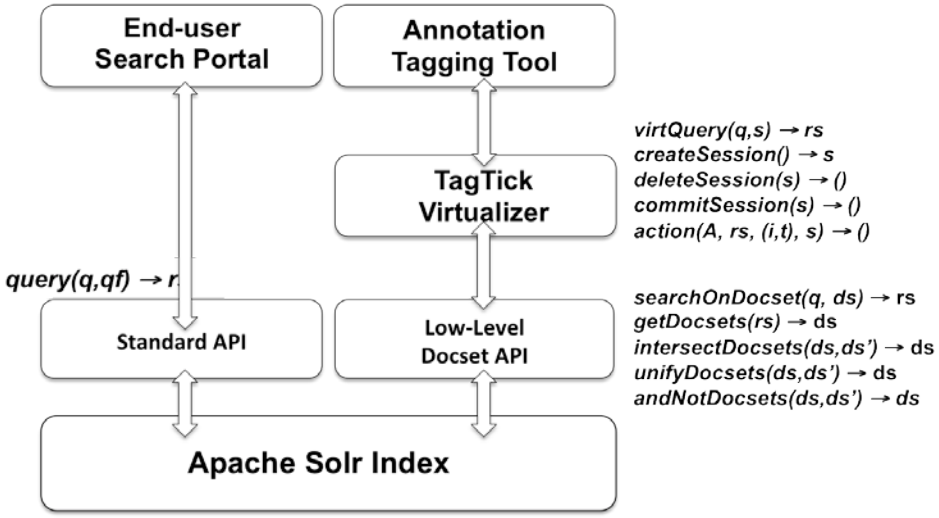

Figure 1. TagTick Virtualizer: The Architecture.

TagTick Virtualizer API: the intended semantics

The commands *createSession()* creates a new session *s*, intended as a sequence of (un)tagging

actions over an initial Information Space *I*. The command and *deleteSession(s)* removes the session *s* from the environment. We denote the virtual information space obtained by modifying *I* with the actions in *s* as *I(s)*; note that: $I(\epsilon) = I$.

createSession()	creates and returns a work session *s*
deleteSession(s)	deletes a work session *s*
commitSession(s)	commits a work session *s*
action(A, rs, (i, t), s)	applies the action *A* with *(i, t)*to all objects in *rs* in *s*
virtQuery(q, s)	executes *q* over the Information Space *I(s)*

Table 4. TagTick Virtualizer API: The Methods.

The command *action(A, rs, (i, t), s)*, depending on the value of *A* being *tag* or *untag*, applies the relative action for the tag *(i, t)* to all objects in *rs* and in the context of the session *s*. (Un)tagging actions occur in the context of a session *s*, hence update the scope of the Information Space *I(s)*. The construction of such *rs* takes place in the annotation tagging tool user interface and may require several queries before all objects to be bulk (un)tagged are collected. Annotation tagging tools may for example provide web-basket mechanisms to support curators in this process.

The command *commitSession(s)* makes the virtual Information Space *I(s)* persistent, i.e., materializes the bulk updates collected in session *s*. Once this operation is completed, the session *s* is deleted.

The command *virtQuery(q, s)* executes a virtual search whose semantics is that of the Solr's method *query(q, null)* executed over *I(s)*. More formally, let's extend the semantic function $[\![.]\!]_{Solr}$ to include the information space scope of the execution, that is: $[\![query(q,q_f)]\!]^{I}_{Solr}$ is the semantics of *query(q, q_f)* over a given Information Space *I*. Then, we can define:

$$[\![virtQuery(q, s)]\!]_{TV} = [\![query(q, null)]\!]^{I(s)}_{Solr}$$

TagTick Virtualizer API: The implementation

To provide its functionalities in real time, the TagTick Virtualizer avoids any form of update action into the index. The module emulates the application of bulk (un)tagging actions over the information space by exploiting Solr low-level library for docset management, whose methods are shown in table 2. The underlying intuition is based on two considerations: (1) the action *action(A, rs, (i, t), s)* can be encoded in memory as an association between the tag *(i, t)* and the objects in the docset *ds* relative to *rs* in the context of *s*; and (2) the subset of objects *ds* should be returned to the

query *(i = t)* if executed over *I* in the scope of *s* (i.e., as if *I* was updated with such an action). By following this approach, the module may rewrite and execute calls of the form *virtQuery(q And (i = t))* into calls *searchOnDocset(q, ds)*, thereby emulating the real-time execution of the query over the information space *I(s)*. More generally, any query of the form *q And* $q_{tag\ predicates}$, where $q_{tag\ predicates}$ is a query combining tag predicates relative to tags touched in the session, can be rewritten as *searchOnDocset(q, ds)*. In such cases, ds is obtained by combining the docsets relative to tag predicates by means of the low-level methods *intersectDocsets* and *unifyDocsets*.

The TagTick Virtualizer module implements the aforementioned session cache by means of an in-memory map $\rho = S \times I \times T \rightarrow DS$, which caches the tagging status of all active work sessions. To this aim, ρ maps triples *(s, i, t)* onto docsets *ds* that are defined as the set of objects tagged with the tag *(i, t)* in the context of *s* at the time of the request. The TagTick Virtualizer is stateless with regard to the specific tags and sessions identifiers it is called to handle; such information is typically held in applications using the module to take advantage of real-time, virtual tagging mechanisms.

Tagging and untagging actions

The method *action(A, rs, (i, t), s)* has the effect of changing the status ρ to reflect the action of tagging or untagging the objects in the result set *rs* with the tag *(i, t)* in the session *s*. Table 5 describes the effect of the command over the status ρ in terms of the semantic function

$$[\![.]\!]_M : C \times \wp(S \times I \times T) \rightarrow \wp(S \times I \times T)$$

that takes a command *C* and a status ρ and returns the status ρ affected by *C*.

In order to optimize the memory heap, ρ is populated following a lazy approach, according to which a new entry for the key *(s, i, t)* is created when the first tagging or untagging action with respect to the tag *(i, t)* is performed in the scope of *s*. When the user adds or removes a tag *(i, t)* for the first time in the session *s* (case $\rho(s, i, t) = \perp$), the value of the entry $\rho(s, i, t)$ is initialized to the docset relative to the query *i = t*:

$$ds = getDocset([\![query((i=t),null)]\!]^I_{Solr}$$

The function $init(\rho, s, i, t)$ returns such new ρ over which the tag or untag action is eventually executed. If the action involves a tag *(i, t)* for which an entry $\rho(s, i, t) = ds$ exists (case $\rho(s, i, t) \neq \perp$), the commands return the new ρ obtained by adding or removing the docset *getDocset(rs)* to or from *ds*. Such actions are performed in memory with minimal execution time.

$$[\![action(A, rs, (i, t), s)]\!]_{M(\rho)} =$$

$$\begin{cases} updateTag(\rho, rs, (i, t), s) & if(A=tag\ And\ \rho(s, i, t) \neq \perp) \\ updateUntag(\rho, rs, (i, t), s) & if(A=untag\ And\ \rho(s, i, t) \neq \perp) \\ [\![action(A, rs, (i, t), s)]\!]_{M(init(\rho, s, i, t))} & if(\rho(s, i, t) = \perp) \end{cases}$$

$$init(\rho, s, i, t) = \rho[\rho(s, i, t) \leftarrow getDocset([\![query(i=t, null)]\!]_{Solr}]$$

$$updateTag(\rho, rs, (i, t), s) = \rho[\rho(s, i, t) \leftarrow \rho(s, i, t) \cup getDocset(rs)]$$

$$updateUntag(\rho, rs, (i, t), s) = \rho[\rho(s, i, t) \leftarrow \rho(s, i, t) \setminus getDocset(rs)]$$

Table 5. Semantics of tag/untag commands.

Queries over a Virtual Information Space

As mentioned above, the command *virtQuery(q, s)* is implemented by executing the low-level method *searchOnDocset(q', ds)*. Informally, q' is the subpart of q whose predicates are not affected by actions in s, while ds is the subset of objects matching tag predicates affected by actions in s, to be calculated by means of the map ρ. To make this a real statement, two main issues must be addressed. The first one is syntactical: how to extract from q the sub-query q' and the subquery to be filtered by ρ to generate ds. The second issue is semantic: the misalignment between the objects in the original Information Space I, where *searchOnDocset* is executed, and the ones in $I(s)$, to be virtually queried over and returned by *virtQuery*.

Syntactic issue:

To obtain q' and ds from q, the TagTick Virtualizer module includes a Query Rewriter module that is in charge of rewriting q as a query:

$$q'\ And\ q_{tags\ in\ session} \qquad (1)$$

Both queries are compliant to the query grammar in table 1, but the second is a query that groups all tag predicates in q which are affected by s. The reason of this restriction is due to the fact that the method *searchOnDocset(q', ds)* performs an intersection between the docset ds and the docset obtained from the execution of $\bar{q}$. In principle, $q_{tags\ in\ session}$ may contain arbitrary combinations of tag predicates $(i = t)$ combined with *And* and *Or* operators. To get a better understanding, refer to the examples in table 6, where we assumed to have two tag interpretations A with terms $\{a_1, a_2\}$ and B with terms $\{b_1, b_2\}$ where $\rho(s, A, a_1)$ and $\rho(s, B, b_1)$ are defined in ρ; note that keyword searches, e.g., "napoleon," are not run over tag values. The first two queries can be executed, while the last one is invalid. Indeed there is no way to factor out the tag predicate $(A = a_1)$ so that it can be separated and joint with the rest of the query using an *And* operator.

Clearly, the ability of the Query Rewriter module to rewrite the query independently of its complexity may be crucial to increase the usability level of TagTick Virtualizer. In its current implementation, the TagTick Virtualizer assumes that *q* is provided to *virtQuery* as already satisfying the expected query structure (1). As we shall see in the next section, this assumption is very reasonable in the realization of our annotation tagging tool TagTick and, more generally, in the definition of tools for annotation tagging. Indeed, such tools typically allow data curators to run Google-like free-keyword queries to be refined by a set of tags selected from a list. Such queries fall in our assumption and also match the average requirements of this application domain.

$q =$ "napoleon" $And\ (A = a_1\ Or\ B =\ b_1)\ where$:
$q' =$ "napoleon"
$q_{tag\ in\ session} = (A = a_1\ Or\ B =\ b_1)$
$q = (A = a_2\ Or$ "napoleon") $And\ (A = a_1\ Or\ B =\ b_1)\ where$:
$q' = (A = a_2\ Or$ "napoleon")
$q_{tag\ in\ session} = (A = a_1\ Or\ B =\ b_1)$
$q = (A = a_1\ Or\ B =\ b_2)\ And$ napoleon

Table 6. Query rewriting.

Semantic issue:

The command *searchOnDocset(q̄, ds)* does not match the expected semantics of *virtQuery(q, s).* The reason is that *searchOnDocset* is executed over the original information space *I* and objects in the returned result set may not reflect the new tagging imposed by actions in *s*. For example, consider an untagging action for the tag *(i, t)* and the result set rs in *s*. Although the objects in *rs* would never be returned for a query *virtQuery((i = t),s)*, they could be returned for queries regarding other properties and in this case they would still display the tag *(i, t)*. To solve this problem, the function *patchResultset : RS → RS* in table 7 intercepts the result set returned by *searchOnDocset* and "patches" its objects, by properly removing or adding tags according to the actions in *s*. To this aim, the function exploits the low-level function *intersectSize*, which efficiently computes and returns the size of the intersection between two docsets. For each object *d* in a given result set *rs*, the function verifies if *d* belongs to the docsets *ρ(s, i, t)* relative to the tags touched by the session *s*: if this is the case (*intersectSize* returns 1), the object should be enriched with the tag (*add(d, (i, t))*), otherwise the tag should be removed from the object (*remove(d, (i, t))*).

$$patchResultset(rs, r, s) = \bigcup_{r(s,i,t)\neq\wedge\ d\hat{I}rs} \bigcup \{patchDocument(d,(i,t))\}$$

$$patchDocument(d,(i,t)) =$$

$$\begin{cases} add(d,(i,t)) & if\ intersectSize(\{d\}, getDocset(rs)) = 1 \\ remove(d,(i,t)) & if\ intersectSize(\{d\}, getDocset(rs)) = 0 \end{cases}$$

Table 7. Patching result sets.

The TagTick Virtualizer implements also patching of results for browse queries. A Solr browse query is a CQL query *q* followed by the list of object properties *l* for which a group-by operation (in the sense of relational databases) is demanded. The query returns two responses: the query result set *rs* and the group-by statistics $(l, v, k_{(l, v)})$ calculated over the result set and for the given properties, where $k_{(l, v)}$ is the number of objects featuring the value *v* for the property *l* in *rs*. As in the case of standard queries, the semantic issue affects browse queries when a group-by is applied over a tag interpretation *i* touched in the current work session. Indeed, the relative stats would be calculated over the information space *I* rather than the intended *I(s)*. To solve this issue, when a browse query demands for stats over a tag interpretation *i*, the relative triples $(i, t, k_{(i, t)})$ are patched as follows:

1. If $(i, t, k_{(i, t)})$ is such that $\rho(s, i, t) = \perp$, i.e. the tag was not affected by the session, then $k_{(i, t)}$ is left unchanged;
2. If $(i, t, k_{(i, t)})$ is such that $\rho(s, i, t) = ds$, then

$$k_{(i,t)} = intersectSize(ds, getDocset(rs))$$

The operation returns the number of objects currently tagged with (i, t) which are also present in the result set *rs*.

Query execution:
The implementation of *virtQuery* can therefore be defined as

$$[\![virtQuery(q,s)]\!]_{TV} = patchResultset(searchOnDocset(\bar{q}, ds), \rho, s)$$

where *q* is rewritten in terms of $\bar{q}$ and $q_{tags\ in\ session}$ by the Query Rewriter module, and *ds* is the docset obtained by applying the function

$$[\![.]\!]_{VT}: Q \times S \times \wp(S \times I \times T) \rightarrow DS$$

defined in Table 8 to $q_{tags\ in\ session}$. The function, given a query of tag predicates, a session identifier, and the status map ρ returns the docset of objects satisfying the query in the session's scope.

$$[\![q_1 \; Or \; q_2]\!]_{V(s,\rho)} = unifyDocsets([\![q_1]\!]_{V(s,\rho)}, [\![q_2]\!]_{V(s,\rho)})$$

$$[\![q_1 \; And \; q_2]\!]_{V(s,\rho)} = intersectDocsets([\![q_1]\!]_{V(s,\rho)}, [\![q_2]\!]_{V(s,\rho)})$$

$$[\![(i{=}t)]\!]_{V(s,\rho)} = \rho(s, i, t)$$

Table 8. Evaluation of $q_{tags \text{ in session}}$ in session *s*.

The definition of ρ, the Query Rewriter module, the semantics of the commands action and *virtQuery*, the definition of *searchOnDocset*, and the function $[\![.]\!]_V$ guarantee the validity of the following claim, crucial for the correctness of the TagTick Virtualizer:

Claim (Search correctness)

Given an information space *I*, a map ρ, and a session *s*, for any query *q* such that

1. $q = \bar{q} \; And \; q_{tags \; in \; session}$
2. $ds = [\![q_{tags \; in \; session}]\!]_V(s, \rho)$

we can claim that

$$[\![virtQuery(q, s)]\!]_{TV} = [\![query(q, null)]\!]_{Solr}^{I(s)}$$

hence the implementation of the command *virtQuery* matches its expected semantics.

Making a Virtual Information Space Persistent

The *commitSession(s)* command is responsible for updating the initial Information Space *I* to the changes applied in *s*, i.e. add and remove tags to objects in *I* according to the actions in *s*. To this aim, the module relies on the map ρ, which associates each tag *(i, t)* to the set of objects virtually tagged by *(i, t)* in *s*, and on the low-level function *andNotDocsets*. By properly matching the set of objects tagged by *(i, t)* in *I* and *I(s)* the function derives the sets of objects to tag and untag in *I*. Overall, the execution of *commitSession(s)* consists in:

1. Identifying the set of tags affected by tagging or untagging actions in the session *s*:

$$changedTags(s) = \{(i, t) | \rho(s, i, t) \neq \perp\}$$

2. For each $(i, t) \in changedTags(s)$
   a) fetching the result set relative to all objects in *I* with tag *(i = t)*:

$$rs = query((i = t), null);$$

   b) keeping in memory the relative docset *ds = getDocset(rs)*;
   c) calculating in memory the set of objects in *I* to be untagged by *(i = t)*:

$$toBeUntagged = andNotDocsets(ds, \rho(s, i, t))$$;

d) calculating in memory the set of objects in *I* to be tagged with *(i = t)*

$$toBeTagged = intersectDocset(\rho(s, i, t), ds)$$;

e) update the index to tag and untag all objects in the two sets; and

f) Remove session *s*.

The TagTick Virtualizer module is also responsible for the management of conflicts on commits and to avoid index inconsistencies. To this aim, only the first commit action is executed, and once the relative actions are materialized into the index, all other sessions are invalidated, i.e., deleted.

TagTick User Interface: Annotation Tagging for Solr

The TagTick User Interface module implements the functionalities presented in previously over a Solr index equipped with the TagTick Virtualizer module described in the section on Solr and annotation tagging (see figure 2). The user interface offers to authenticated data curators an annotation tagging environment where they can open work sessions, do and undo sequences of (un)tagging actions, and eventually commit the session into the current Solr information space. When data curators log out from the tool, the modules stores on disk their pending work sessions and the relative (un)tagging actions. Such sessions will be restored at the next access to the interface, to allow data curators continuing their work.

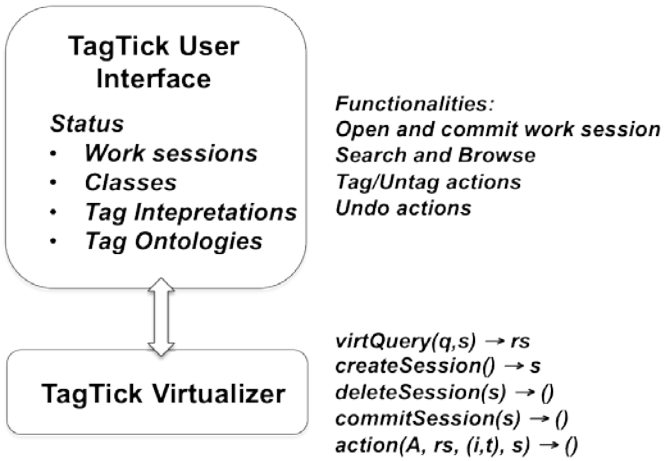

Figure 2. TagTick: User Interface.

The TagTick User Interface is a general-purpose module that can be configured to adapt to the classes and to the structure of objects residing in the index. To this aim, the modules acquires this information from XML configuration files where data curators can specify:

1. The names of the different classes, the values used to encode such classes in the index, and the index field used to contain such values;
2. The list of tag interpretations together with the relative ontologies: in the current implementation ontologies are flat sets of terms, which can be optionally populated by curators during the tagging step; and
3. The intended use of interpretations: the association between classes and interpretations.

Once instantiated, the TagTick User Interface allows users to search for objects of all classes by means of free keywords and to refine such searches by class and by the tags relative to such class. This combination of predicates, which matches the query structure $\bar{q} = q\ And\ q_{tags\ in\ session}$ expected by the TagTick Virtualizer, is then executed by the module and the results presented in the interface. Users can then add or remove tags to the objects—the interface makes sure that the right interpretations are used for the given class.

As an example, we shall consider the real-case instantiation of TagTick in the context of the HOPE project, whose aim is to deliver a data infrastructure capable of aggregating metadata records describing multimedia objects relative to labour history and located across several data sources.[24] Such objects are collected, cleaned, and enriched to form an information space stored into a Solr index. The index stores two main classes of objects: descriptive units and digital resources. Descriptive unit objects contain properties describing cultural heritage objects (e.g., a pin). Digital resource objects instead describe the digital material representing the cultural heritage objects (e.g., the pictures of a pin). TagTick is currently used in the project HOPE to classify the aggregated objects according to two tag interpretations: "historical themes," to tag descriptive units with an ontology of terms describing historical periods, and "export mode," to tag digital resources with an ontology which describes the different social sites (e.g., YouTube, Facebook, Flickr) from which the resource must be made available from. In particular, figure 3 illustrates the HOPE TagTick user interface. In the screenshot, a set of descriptive units obtained by a query is being added a new tag "Communism . . ." of the tag interpretation "historical themes." The TagTick User Interface offers the possibility to access the history of actions, in order to visualize their sequence, and possibly to undo their effects. Figure 4 shows the history of actions that led to the actual tag virtualization in the current work session. Curators can only rollback the last action they accomplished. This is because virtual tagging actions may be depending on each other; e.g., an action is based on a query that includes tag predicates whose tag has been affected by previous actions. Other approaches may infer the interdependencies between the queries behind the tagging actions and expose dependency-based undo options.

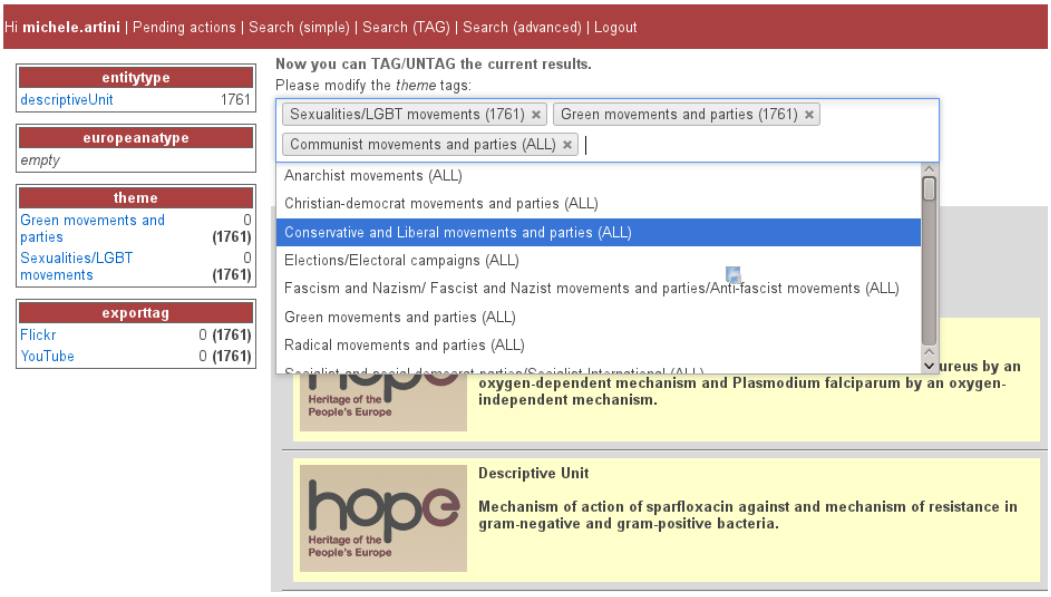

Figure 3. TagTick User Interface: Bulk Tagging Action.

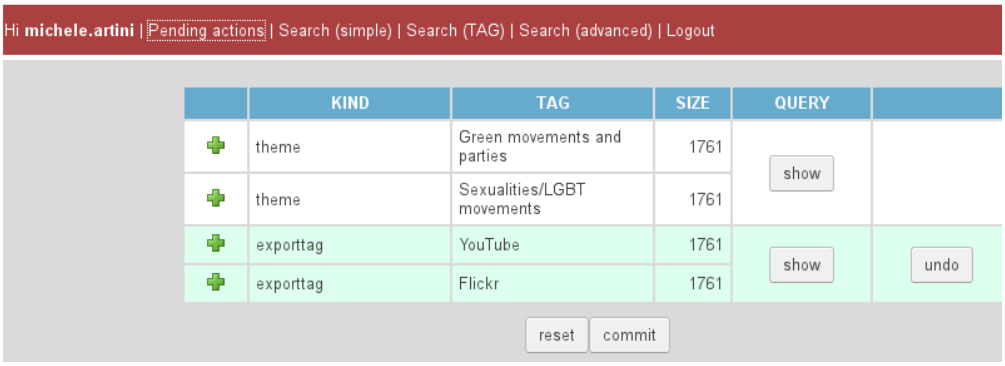

Figure 4. TagTick User Interface: Managing History of Actions.

STRESS TESTS

The motivations behind the realization of TagTick are to be found in annotation tagging requirements of bulk and real-time tagging. In general, the indexing speed of Solr highly depends on the underlying hardware, on the number of threads used for feeding, on the average size of the objects and their property values, and on the kind of text analysis to be adopted.[25] However, even assuming the most convenient scenario, bulk indexing in Solr is comparably slow with respect to other technologies, such as relational databases,[26] and far from being real-time. In this section, we present the result of stress tests conceived to provide concrete measures of query performance, i.e., the real-time effect, the scalability of the tool, and how many tagging actions can be handled in the same session. The idea of the tests is to re-create worst scenarios and give evidence of the ability of TagTick to cope and scale with response time and memory

consumption.

The experiments were run on a machine with processor Intel(R) Xeon(R) CPU E5630 @ 2.53GHz (4 cores), a total of memory 4 GB, and available disk of 100 GB (used at around 52 percent). The machine installs an Ubuntu 10.04.2 LTS operating system, with a Java Virtual Machine configured as -Xmx1800m -XX:MaxPermSize = 512m. In simpler terms, a medium-low server for a production index. The index was fed with 10 million objects randomly generated and with the following structure:

```
[ identifier: String,
  title: String,
  description: String,
  publisher: String,
  URL: String,
  creator: String,
  date: Date,
  country: String,
  subject: Terms]
```

The tag interpretation *subject* can be assigned values from an ontology *Terms* of scientific subjects, such as "Agricultural biotechnology," "Automation," "Biofuels," "Biotechnology," "Business aspects." The objects are initially generated without tags.

Each test defines a new session *s* with *K* tagging actions of the form

$$action(tag, virtQuery(identifier <> ID,null), t, s)$$

where *ID* is a random identifier and *t* is a random tag *(subject, term)*. In practice, the action adds the tag *t* to all objects in the index, thereby generating docsets of size 10 million. Once the *K* actions are executed, the test returns the following measures:

1. The size of the heap space required to store *K* tags in memory.
2. The minimal, average, and maximum time required to reply to two kinds of stress queries to the index (calculated out of 100 queries):
    a. The query *identifier <> ID* $AND_{(i,t)\in s}(i = t)$: the query returns the objects in the index which feature all tags touched by the session.
    b. The query *identifier <>ID* $OR_{(i,t)\in s}(i = t)$: the query returns the objects in the index which feature at least one of the tags assigned in the session.

    In both cases, since tagging actions where applied to all objects in the index, the result will contain the full index. However, in one case the response will be calculated by intersecting docsets, while in the other case by unifying them. Note that by selecting a random identifier value (*ID*), the test makes sure that low-level Solr optimization by caching is not fired, as this would compromise the validity of the test.

3. The minimal, average, and maximum time required to reply to browse queries which involve all tags used in the session (calculated out of 100 queries).
4. The time required to reconstruct the session in memory whenever the data curator logs into TagTick.

The results presented in figure 5 show that the average time for the execution of search and browse queries always remain under 2 seconds, which we can consider under the "real-time" threshold from the point of view of the users. User tests have been conducted in the context of the HOPE project, where curators were positively impressed by the tool. HOPE curators can today apply sequences of tagging operations over millions of aggregated records by means of a few clicks. Moreover, independently of the number of tagging operations, queries over the tagged records take about 2 seconds to complete.

The execution time has a major increase from 0 tags to 1 tag. This behavior is expected because when there is 1 tag in the session, the 10 million records must be "patched." From 1 tag onwards the execution time increases as well, but not at the same rate as in the previous case. This means that in the average case patching 10 million records with 100 tags does not cost much more than tagging them with 1 tag.

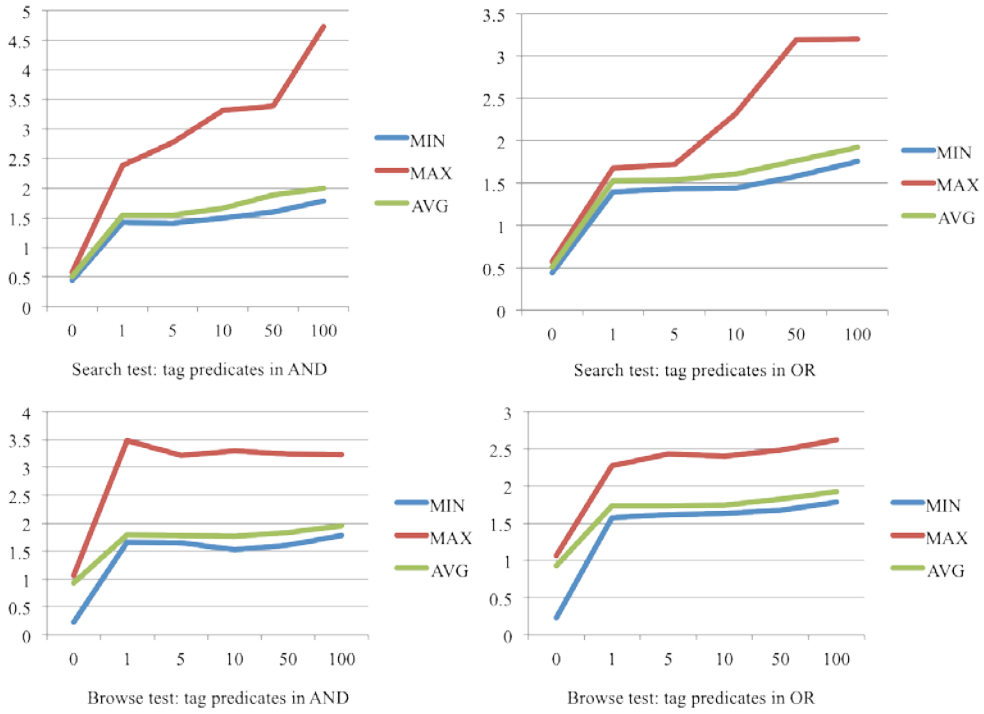

Figure 5. Stress Test for TagTick Search and Browse Functionality.

The results in figure 6 show that the amount of memory to be used does not exceed the limits expected on reasonable servers running a production system. The time required to reconstruct the sessions is generally long, starting from 20 seconds for 50 tags up to 1.5 minutes for 200 tags.

On the other hand, this is a one-time operation, required only when logging in to the tool.

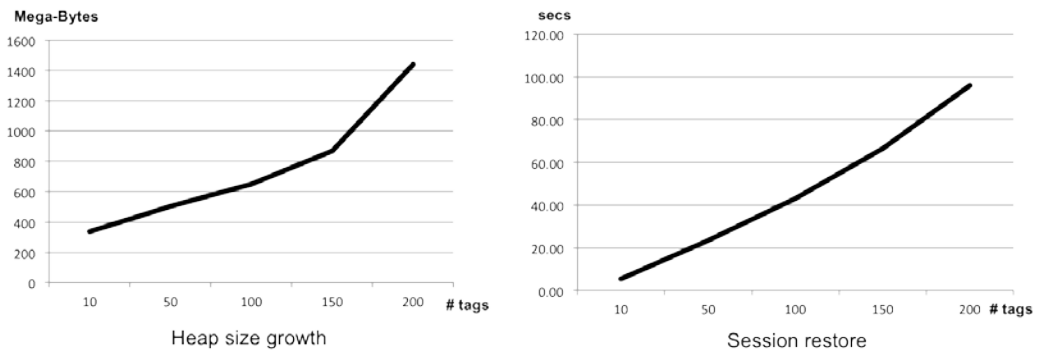

Figure 6. Stress Test for Heap Size Growth and Session Restore Time.

CONCLUSIONS

In this paper, we presented TagTick, a tool devised to enable annotation tagging functionalities over Solr instances. The tool allows a data curator to safely apply and test bulk tagging and untagging actions over the index in almost real time and without compromising the activities of end users searching the index at the same time. This is possible thanks to the TagTick Virtualizer module, which implements a layer over Solr that enables real-time and virtual tagging by keeping in memory the inverted list of objects associated to a (un)tagging action. The layer is capable of parsing user queries to intercept the usage of tags kept in memory and, in this case, to manipulate the query response to deliver the set of objects expected after tagging.

Future developments may regard the ability to enable more complex query parsing to handle rewriting of a larger set of queries beyond Google-like queries currently handled by the tool. Another interesting challenge is tag propagation. Curators may be interested in having the action of (un)tagging an object to be propagated to objects that are somehow related with the object. Handling this problem requires the inclusion into the information space model of relationships between classes of objects and the extension of the TagTick Virtualizer module for the specification and management of propagation policies.

ACKNOWLEDGEMENTS

The work presented in this paper has been partially funded by the European Commission FP7 eContent*plus*-2009 Best Practice Networks project HOPE (Heritage of the People's Europe, http://www.peoplesheritage.eu), grant agreement 250549.

REFERENCES

1. Arkaitz Zubiaga, Christian Körner, and Markus Strohmaier, "Tags vs Shelves: From Social Tagging to Social Classification," in *Proceedings of the 22nd ACM conference on Hypertext and Hypermedia*, 93–102 (New York: ACM, 2011), http://dx.doi.org/10.1145/1995966.1995981.

2. Meng Wang et al., "Assistive Tagging: A Survey of Multimedia Tagging with Human-Computer Joint Exploration," *ACM Computer Survey* 44, no. 4 (September 2012): 25:1–24, http://dx.doi.org/10.1145/2333112.2333120.

3. Lin Chen et al., "Tag-Based Web Photo Retrieval Improved by Batch Mode Re-tagging," in *2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2010), 3440–46, http://dx.doi.org/10.1109/ CVPR.2010.5539988.

4. Emanuele Quintarelli, Andrea Resmini, and Luca Rosati, "Information Architecture: Facetag: Integrating Bottom-Up and Top-Down Classification in a Social Tagging System," *Bulletin of the American Society for Information Science & Technology* 33, no. 5 (2007): 10–15, http://dx.doi.org/10.1002/bult.2007.1720330506.

5. Stijn Christiaens, "Metadata Mechanisms: From Ontology to Folksonomy . . . and Back," in *Lecture Notes in Computer Science: On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops* (Berlin Heidelberg: Springer-Verlag, 2006).

6. M. Mahoui et al., "Collaborative Tagging of Art Digital Libraries: Who Should Be Tagging?" in *Theory and Practice of Digital Libraries*, ed. Panayiotis Zaphiris et al., 162–72, vol. 7489, *Lecture Notes in Computer Science* (Springer Berlin Heidelberg, 2012), http://dx.doi.org/10.1007/978-3-642-33290-6_18.

7. Alexandre Passant and Philippe Laublet, "Meaning Of A Tag: A Collaborative Approach to Bridge the Gap Between Tagging and Linked Data," in *Proceedings of the Linked Data on the Web (LDOW2008) Workshop at WWW2008*, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.142.6915.

8. Michael Khoo et al., "Towards Digital Repository Interoperability: The Document Indexing and Semantic Tagging Interface for Libraries (DISTIL)," in *Theory and Practice of Digital Libraries*, ed. Panayiotis Zaphiris et al., 439–44, vol. 7489, *Lecture Notes in Computer Science* (Springer Berlin Heidelberg, 2012), http://dx.doi.org/10.1007/978-3-642-33290-6_49.

9. Leonardo Candela, et al, "Setting the Foundations of Digital Libraries: The DELOS Manifesto." D-Lib Magazine 13, no. 3/4, March/April 2007, http://dx.doi.org/10.1045/march2007-castelli.

10. Jennifer Trant, "Studying Social Tagging and Folksonomy: A Review and Framework," *Journal of Digital Information* (January 2009), http://hdl.handle.net/10150/105375.

11. Cameron Marlow et al., "HT06, tagging Paper, Taxonomy, Flickr, Academic Article, To Read," in *Proceedings of the Seventeenth Conference on Hypertext and Hypermedia*, 31–40 (New York: ACM, 2006), http://dx.doi.org/10.1145/1149941.1149949.

12. Andrea Civan et al., "Better to Organize Personal Information by Folders or By Tags? The Devil is In the Details," *Proceedings of the American Society for Information Science and Technology* 45, no. 1 (2008): 1–13, http://dx.doi.org/10.1002/meet.2008.1450450214.

13. Marianne Lykke et al., "Tagging Behaviour with Support from Controlled Vocabulary," in *Facest of Knowledge Organization*, ed. Alan Gilchrist and Judi Vernau, 41–50 (Bingley, UK: Emerald Group, 2012)

14. Guus Schreiber et al., "Semantic Annotation and Search of Cultural-Heritage Collections: The MultimediaN E-Culture Demonstrator," *Web Semantics: Science, Services and Agents on the World Wide Web* 6, no. 4 (2008): 243–49, http://dx.doi.org/10.1016/j.websem.2008.08.001.

15. Diana Maynard and Mark A. Greenwood, "Large Scale Semantic Annotation, Indexing and Search at the National Archives," in *Proceedings of LREC* vol. 12 (2012).

16. Martin Feijen, "DRIVER: Building the Network for Accessing Digital Repositories Across Europe," *Ariadne* 53 (October 2007), http://www.ariadne.ac.uk/issue53/feijen-et-al/.

17. Heritage of the People's Europe (HOPE), http://www.peoplesheritage.eu/.

18. European Film Gateway Project, http://www.europeanfilmgateway.eu.

19. Paolo Manghi et al., "OpenAIREplus: The European Scholarly Communication Data Infrastructure," *D-Lib Magazine* 18, no. 9–10 (September 2012), http://dx.doi.org/10.1045/september2012-manghi.

20. Panagiotis Antonopoulos et al., "Efficient Updates for Web-Scale Indexes over the Cloud," in *2012 IEEE 28th International Conference on Data Engineering Workshops (ICDEW),* 135–42, April 2012, http://dx.doi.org/10.1109/ICDEW.2012.51.

21. Chun Chen et al., "TI: An Efficient Indexing Mechanism for Real-Time Search on Tweets," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, 649–60 (New York: ACM, 2011), http://dx.doi.org/10.1145/1989323.1989391.

22. Rafal Kuc, *Apache Solr 4 Cookbook* (Birmingham, UK: Packt, 2013).

23. David Smiley and Eric Pugh, *Apache Solr 3 Enterprise Search Server* (Birmingham, UK: Packt, 2011).

24. The HOPE Portal: The Social History Portal, http://www.socialhistoryportal.org/timeline-map-collections.

25. Assuming to operate a stand-alone instance of Solr, hence not relying on Solr sharding techniques with parallel feeding.
26. WhyUseSolr—Solr Wiki, http://wiki.apache.org/solr/WhyUseSolr.