Horizon 2020

ICT - INFORMATION AND COMMUNICATION TECHNOLOGIES

PROmoting **FI**nancial **A**wareness and **S**tabili**T**y

H2020 – 687895

Sharing Activities and Crowdsourcing Mechanisms

Work package No.	**WP2**	Work package Title	**Linked Data Life Cycle**
Task No.	**T2**.3	Task Title	Data Publication, Visualisation and Sharing
Milestone No.	24		
Organization name of lead contractor for this deliverable	**SWC**		
Editors	Albin Ahmeti (SWC)		
Contributors	Martin Schauer (SWC), Artem Revenko (SWC)		
Reviewers	Peter Hanecak (EEA), Victor Mireles (SWC)		
Status (F: final; D: draft)	F		
Nature	R – Report		
Dissemination Level	PU – Public		
Project start date and duration	**January 2016, 36 Months**		
Due Date of Deliverable: Actual Submission Date:	Dec 31, **2017** Dec 29, **2017**		

REVISION HISTORY

Version	Date	Modified by	Changes
0.1	25-10-2017	Artem Revenko	First version of ToC
0.2	7-11-2017	Artem Revenko	Introduction
0.3	13-11-2017	Albin Ahmeti	Approach
0.4	15-11-2017	Artem Revenko	Implementation
0.5	19-11-2017	Albin Ahmeti	Implementation, examples
0.6	14-12-2017	Albin Ahmeti	Revising and Finalizing
1.0	26-12-2017	Artem Revenko	Final Revision

Executive Summary

The deliverable presents the results of preparing the crowd-sourcing tools for the PROFIT platform. It reports about the following achieved results:

Section 1 clarifies the problem statement, lists imposed constraints, and explains the relation to PROFIT project. The main difference to the standard crowd-sourcing settings lies in the educational and inclusional aspect of the PROFIT platform: the existing knowledge is not shown to the user, therefore, the user is absolutely free to convey her own visual of the domain.

Section 2 outlines the followed approach. The framework of belief revision is employed for scoring and intersecting the existing knowledge graph and the user input. This framework enables to estimate the distance of user’s input to the existing knowledge and find the new suggestions.

Section 3 describes the implementation of the intended approach. The implemented functionalities, i.e. the capabilities of the tool are described. The workflow of the tool is described and illustrated with examples.

The described tool is implemented, delivered, and initial integration into the PROFIT platform is implemented. The required capabilities (Section 1.1) are fully covered by the implemented tool. The range of possible user’s suggestions is planned to be expanded in the next steps. The exact description of working with the tool is to be found in Section 3.3 “Usage Scenario”.

1. Introduction

1.1. Problem Statement

The PROFIT project is a collective awareness project that aims at social innovation, hence, PROFIT is inclusive in the sense of people involvement. This consideration influenced the design of the crowd-sourcing solution in that we do not make any assumption about the knowledge or social status of the user and offer the users to define their sub-domain of interest themselves. Indeed, the users are free to choose the concepts of their interest themselves and work with the relations that they are knowledgeable about.

Moreover, at the mid-term review the partners of the project have obtained some valuable feedback. Namely, in relation to the crowd-sourcing we considered that the implemented tool should not “force” any certain vision of the domain of discourse, even if this vision is believed to be correct and confirmed by the experts in the domain. In the implemented tool the users are free to provide their own structure of the domain without seeing the already existing structure. The users therefore have a possibility to provide input that would contradict the existing knowledge. Moreover, the compliance with the existing structure is estimated upon the submission of results and used to:

1. give user a feedback on the confirming/new/contradicting elements;
2. establish an acceptance criteria for the suggestions.

1.2. Objectives

The objectives of the crowd-sourcing tool:

1. Extend the existing knowledge graph;
2. Raise the awareness about the existing knowledge/structure of the domain;
3. Give users feedback about overlaps and contradictions that arise between their vision of the domain and the existing knowledge graph of the domain;
4. Initiate interaction and opinion exchange with
    a. Other users;
    b. Experts.

1.3. Relation to PROFIT project

- **WP1**. The incentive and gamification mechanisms encourage the participation of the users in the crowd-sourcing process. The users get points for:
    - providing their input (score: number of concepts in the input *2),
    - upvoting or downvoting the input of other users (score: 8 points),
    - if their suggestion is accepted (score: 15 points).
- **WP2**. The crowd-sourcing tool allows for extending the existing knowledge graph (as provided by WP2), hence, contributing to it. Moreover, the existing knowledge graph is used to assess the user input.

- **WP3**. The crowd-sourcing tool also fulfils an educational objective: raising awareness about the existing knowledge in the domain and providing immediate feedback about confirming/contradicting elements.

1.4. Scope of the Document

The document describes the approach and the implementation of the crowd-sourcing mechanism developed for the PROFIT project. The description covers the following aspects:

- theoretical background;
- description of the method for computing the results;
- implemented functionalities;
- next possible steps;
- usage scenario;
- examples.

Development Procedure

All the tools were designed in an agile way. The initial design was presented to the consortium and the feedback was collected. The design was reworked according to the collected feedback and the first demonstrator of the tools was prepared. The demonstrator was shown to the consortium and the feedback was again collected and incorporated into the development of the final implementations. This process allows us to state that the tools were developed in a tight cooperation with the whole consortium and the tools cover the needs of all the partners.

2. Approach

2.1. Background

The proposed crowd-sourcing platform should enable the users to suggest new knowledge models in different domains, by expressing themselves in “free-form” mode. It should also enable users to vote on such models, i.e., a (crowd-source) voting by the peers. These models should adhere to some quality criteria, for instance they should not contain facts which are contradictive.

The platform should be able to compute the differences between existing thesauri and user proposed models.

For that purpose, the framework of *belief revision*[1] is chosen: the problem is translated into the belief revision where the thesaurus is ‘mapped’ to the world, and the model created by the user is ‘mapped’ to the update. In belief revision, one relies into the notion of distance to retrieve all models which are close to the world and that satisfy certain *conditions*. Nevertheless, computing the distance is one of the challenging problems in the area of belief revision. In the context of belief revision these conditions are called *postulates*, but in our context these are not really needed and we are not going to dive deep and discuss them. We just mention that the idea of these postulates would be to make the update operator as *rational* as possible.

Let us define the statements present in the existing thesauri as the 'world' W, and the proposed changes to thesauri by users as 'updates' U.

An *inconsistency* is a violation of thesaurus axioms, in our case SKOS axioms[2]. In general, we distinguish between two type of inconsistencies:

- *intrinsic inconsistency*, an inconsistency in the update itself;
- *(general) inconsistency*, an inconsistency that is only present in the union of W and U and does not appear neither in W alone nor in U alone.

Incorporating the update into the world is always challenging. This is due to the fact that some facts in the world contradict the facts coming from update. As a result, one has to choose which one of the facts from the world versus update are to persist and which not. In belief revision one can choose one of the following semantics[3]:

- **brave**, the new update overrides the facts in the world;
- **cautious**, the new update is simply dropped and facts in the world persist;
- **faint-hearted**, only the consistent subset part of the update is incorporated in the world.

Depending of the application requirements one can choose any of these semantics (update operators). This is the scenario when you rely on one of the above semantics to carry out the

[1] Gärdenfors, P. (Ed.). (2003). *Belief revision* (Vol. 29). Cambridge University Press.
[2] https://www.w3.org/TR/skos-reference/#L2055
[3] We deviate a bit here from the classical belief revision, and adopt brave, cautious, faint-hearted notions as implemented in SPARQL [ACPS16].

update in an automatic way, and not introduce humans in the middle to intervene with the result of the update.

For our application we define the distance as $dist(W, U) = |(W \setminus U) \cup (U \setminus W)|$. In Figure 1, distances are shown:

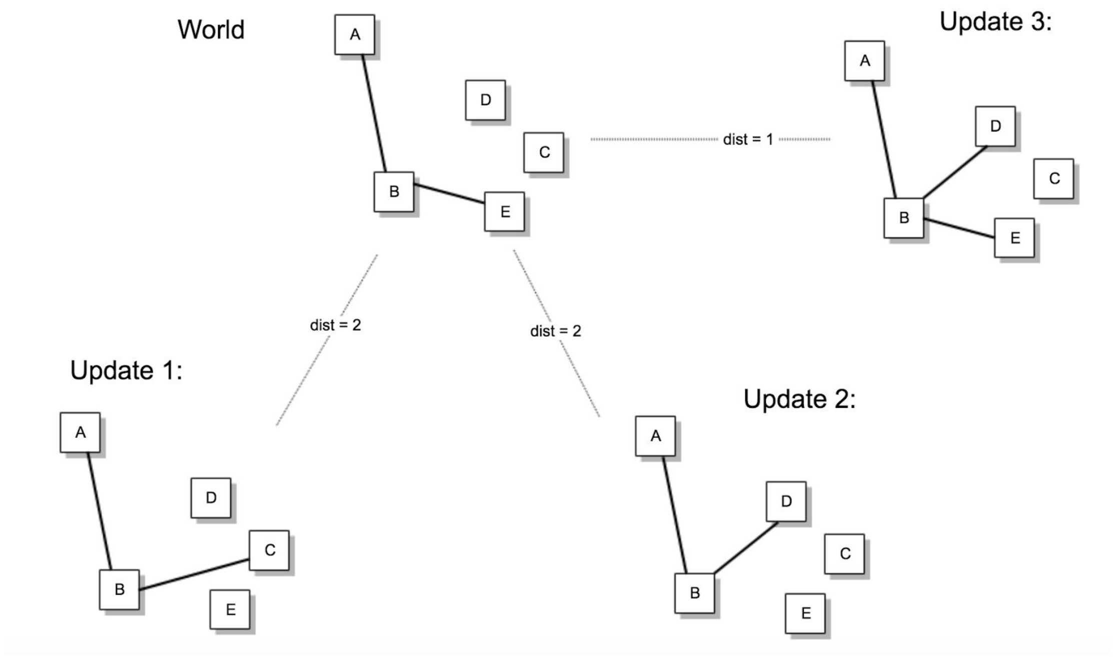

Figure 1: Cardinality-based distance between updates and the world. Nodes represent concepts, and edges represent relations between them. In our case, skos relations.

One can see that the model indicated by Update 3, is *closer* to the thesaurus indicated by the World, because the distance is “1”. For models Update 1 and Update 2 the distance is higher - respectively of distance “2”.

In belief revision, before computing distance, a *closure*[4] is performed on both the world and update, i.e., both are modeled as *belief sets*. Closure computes recursively all implicit triples that are derivable (inferred) from the explicit triples and are stored in a explicit way. This is necessary for correct computation of the distance. As a simple example,

$$if\ A \rightarrow B,\ B \rightarrow C\ is\ in\ belief\ set,\ then\ A \rightarrow C\ is\ computed\ by\ the\ closure\ and\ is\ put\ in\ the\ belief\ set.$$

In a similar fashion, we can define deletes and inserts that are going to be used to see the actual difference between the designated model and the thesaurus. As stated earlier, let W and U indicate World and Update respectively. Deletes DEL are defined as W \ U, i.e., all facts consisted in the World which do not exist in the Update. Inserts INS are defined as U \ W, i.e., all facts consisted in the Update which do not exist in the World. Deletes and Inserts will be useful to show to the user what is the difference in terms of facts, and not only just return the distance. The distance would instead be used in the voting step as an input parameter.

[4] In deductive databases it is called *reaching a fix-point*; in Semantic Web - *materialisation*.

Both thesaurus and models suggested by the user are typically represented as a *set of triples (a graph)* in the field of Semantic Web. For instance, a thesaurus is usually modeled using Web-based SKOS data model and can be serialised in SKOS/RDF in one of the serialisation formats such as RDF/XML, Turtle, N-triples etc. As in our setting we are given triples instead of facts, we need to use a query language to compute the distance between triples, for that reason we use SPARQL. Deletes and inserts can be easily computed in SPARQL (see 2.2 Method). SPARQL has the required expressivity to compute differences in graphs, and it is as expressive as SQL [AG08].

Using the SKOS data model one can develop taxonomies using one of the predefined classes such as:

- *skos:ConceptScheme*, denoting "concept scheme" - the highest member of the taxonomy
- *skos:Concept*, denoting the "top concepts" and "concepts"

Using SKOS one can further model and enrich taxonomies using one of predefined properties such as:

- *skos:topConceptOf* or *skos:hasTopConcept*, further differentiating "top concepts" versus "concepts"
- *skos:broader* or *skos:narrower*, stating the hierarchical relationships between the concepts
- *skos:related*, stating "related" relationships between the concepts

We emphasize that only *skos:broader* and *skos:narrower* are transitive and constitute a hierarchy.

While modeling taxonomies with SKOS one has to take care of some *modelling restrictions*, i.e., be compliant with respect to some *quality assurance checks;* those checks are typically modeled in terms of SPARQL queries and comprehensively are implemented in qSKOS[5] -- also supported in PoolParty. qSKOS is a tool that allows to report on quality metrics for a given thesaurus and one can use it for checks like label conflicts, incomplete language coverage, cyclic hierarchical relations, orphan concepts, missing out links, broken links etc.

For instance, it is not allowed to use `:pizza_margherita skos:broader :pizza` and `:pizza skos:broader :pizza_margherita`, as such cycles are prohibited. The same goes for narrower instead of broader, as well as cycles comprising of couple of broader (or narrower) property traversals and ending up in the same starting (graph) node.

2.2. Method

For the initial version of the prototype, we are going to restrict on only three type of SKOS properties that users can use for modelling:

- skos:broader;
- skos:narrower;
- skos:related

[5] https://www.w3.org/2001/sw/wiki/QSKOS

The other SKOS properties as well as ontological properties are left for future work, and we are not going to discuss them in this deliverable.

In order to compare the model suggested by the user and the thesaurus we rely on the notion of distance. As an assumption, in order to compute the distance correctly[6], for thesaurus we need triples to be materialised. PoolParty automatically materialises the broader and narrower relations.

As discussed previously there are two type of inconsistencies: intrinsic and general inconsistencies. We discuss general inconsistencies and see that intrinsic can be similarly computed when computing “contradicting triples”.

In the following we give the implementation using SPARQL queries for:

- confirming triples (triples existing in both the model and thesaurus)
- new triples (triples existing only in the model)
- contradicting triples (triples in the new model contradicting the thesaurus)
- distance

The following SPARQL queries assume that update and thesauri are stored in separate named graphs, namely on GRAPH `<http://profit.org/update>` and `<http://profit.org/thesaurus>` .

Confirming triples

The following SPARQL query implements the confirming triples. We check for triple bindings that are both in the model and in thesaurus, as well as bindings that are confirming by the implicit triples in thesaurus. This is implemented as separate UNION branch using `broader+` and `narrower+` property path patterns, which are covered in SPARQL 1.1.

```
CONSTRUCT { ?s ?p ?o }
WHERE
{
{
    GRAPH <http://profit.org/update> { ?s ?p ?o }
    GRAPH <http://profit.org/thesaurus> { ?s ?p ?o }
}
UNION
{
    GRAPH <http://profit.org/update> { ?s skos:broader ?o }
    GRAPH <http://profit.org/thesaurus> { ?s skos:broader+ ?o }
    BIND (skos:broader as ?p)
}
```

[6] Since the user suggested models are small compared to thesaurus, the contradicting triples with respect to thesaurus are only used as DELs, and not all triples.

```
UNION
{
    GRAPH <http://profit.org/update> { ?s skos:narrower ?o }
    GRAPH <http://profit.org/thesaurus> { ?s skos:narrower+ ?o }
    BIND (skos:narrower as ?p)
}
}
```

New triples

New triples are computed as all the triples in the update minus[7] the ones in the thesaurus. Note that we don't need to materialise the triples in the update graph, as the first element for comparison we use the update on hand, and then we check the differences with respect to thesaurus.

```
CONSTRUCT { ?s ?p ?o }
WHERE
{
GRAPH <http://profit.org/update> { ?s ?p ?o }
FILTER NOT EXISTS { GRAPH <http://profit.org/thesaurus> { ?s ?p ?o }
}}
```

To return only the set of the new triples, we must exclude the confirming (that takes into account implicit triples):

$$New\ triples = \{New\ triples\} - \{Confirming\}$$

Contradicting triples

Contradicting triples are computed by simply looking for patterns that violate SKOS modelling constraints, such as pairs of:

```
GRAPH <http://profit.org/update> { ?s skos:narrower ?o }
GRAPH <http://profit.org/thesaurus> { ?o skos:broader ?s }
```

And, in more intricate cases, where update contributes to a missing piece that contributes to inconsistency:

```
GRAPH <http://profit.org/update> { ?o skos:broader ?o1 . }
```

[7] We could have opted for the MINUS keyword instead of FILTER NOT EXISTS; the outcome in this case would have been equivalent.

```
GRAPH <http://profit.org/thesaurus> { ?s skos:broader+ ?o . ?o1
skos:broader ?s . }
```

```
CONSTRUCT { ?s skos:narrower ?o . ?m skos:broader ?n  }
WHERE
{
{
    GRAPH <http://profit.org/update> { ?s skos:narrower ?o }
    GRAPH <http://profit.org/thesaurus> { ?o skos:broader ?s }
}
UNION
{
    GRAPH <http://profit.org/update> { ?m skos:broader ?n }
    GRAPH <http://profit.org/thesaurus> { ?n skos:broader ?m }
}
}

CONSTRUCT { ?o skos:broader ?s }
WHERE
{
GRAPH <http://profit.org/update> { ?o skos:broader ?s . }
GRAPH <http://profit.org/thesaurus> { ?s skos:broader+ ?o . }
}


CONSTRUCT { ?o skos:broader ?o1 }
WHERE
{
GRAPH <http://profit.org/update> { ?o skos:broader ?o1 . }
GRAPH <http://profit.org/thesaurus> { ?s skos:broader+ ?o . ?o1
skos:broader ?s . }
}

CONSTRUCT { ?o skos:narrower ?s }
WHERE
{
GRAPH <http://profit.org/update> { ?o skos:narrower ?s . }
GRAPH <http://profit.org/thesaurus> { ?s skos:narrower+ ?o . }
}
```

```
CONSTRUCT { ?o skos:narrower ?o1 }
WHERE
{
GRAPH <http://profit.org/update> { ?o skos:narrower ?o1 . }
GRAPH <http://profit.org/thesaurus> { ?s skos:narrower+ ?o . ?o1
skos:narrower ?s . }
}

CONSTRUCT { ?o skos:related ?s }
WHERE
{
GRAPH <http://profit.org/update> { ?o skos:related ?s . }
GRAPH <http://profit.org/thesaurus> { ?s skos:broader+ ?o . }
}

CONSTRUCT { ?o skos:broader ?o1 }
WHERE
{
GRAPH <http://profit.org/update> { ?o skos:broader ?o1 . }
GRAPH <http://profit.org/thesaurus> { ?s skos:broader+ ?o . ?o1
skos:related ?s }
}

CONSTRUCT { ?o skos:related ?s }
WHERE
{
GRAPH <http://profit.org/update> { ?o skos:related ?s . }
GRAPH <http://profit.org/thesaurus> { ?s skos:narrower+ ?o . }
}

CONSTRUCT { ?o skos:narrower ?o1 }
WHERE
{
GRAPH <http://profit.org/update> { ?o skos:narrower ?o1 . }
GRAPH <http://profit.org/thesaurus> { ?s skos:narrower+ ?o . ?o1
skos:related ?s }
}
```

Intrinsic inconsistencies

The same CONSTRUCT queries can be rewritten so that they check if the update (model) with respect to itself is consistent. This is done simply by doing the same checks, but this time on the same “update” named graph, for instance:

```
CONSTRUCT { ?s skos:narrower ?o . ?m skos:broader ?n  }
WHERE
{
{
    GRAPH <http://profit.org/update> { ?s skos:narrower ?o . ?o
skos:broader ?s }
}
UNION
{
    GRAPH <http://profit.org/update> { ?m skos:broader ?n .  ?n
skos:broader ?m }
}
}
…
// continues the same way as before
```

Distance

Distance is computed as cardinality of the set of triples that are contradicting multiplied by 2, plus the new ones:

$$Distance = |Contradicting| * 2 + |New\ triples|$$

Note that the distance does not distinguish between narrower, broader and related properties, these all contribute equally to the distance. For future work we plan to introduce the notion of weights, such that we give more weights to skos:broader or skos:narrower than to skos:related and compute the distance accordingly. This would bring new dimensions in computing the distance.

Voting scheme

To users are given some randomly selected triples that are suggested from other users, and thus they can either upvote or downvote such triples, in a triple-by-triple manner one at a time. The final score is computed for each new triple:

$$score[U] = upvote[U] - downvote[U]$$

For each new triple, we start initially with: $upvote[U] = downvote[U] = 0$. In order for triple to get finally accepted, we rely on the adaptive threshold as computed below:

$$threshold = max(0, 5 - |confirming \cup contradicting|) + 2 * |contradicting| + 1 .$$

Threshold depends on how well the user's model complies with the existing knowledge, therefore, more radical changes will require more votes:

- the more contradicting triples the more upvotes needed;
- the more confirming triples the less upvotes needed;
- less than 5 proposed concepts is considered too little as input.

If score for a certain update, $score[U]$ is higher than the $threshold$ then it is accepted and waiting for the administrator for confirmation (see "Expert (moderator / admin / superuser) decision").

3. Implementation

3.1. User Interface

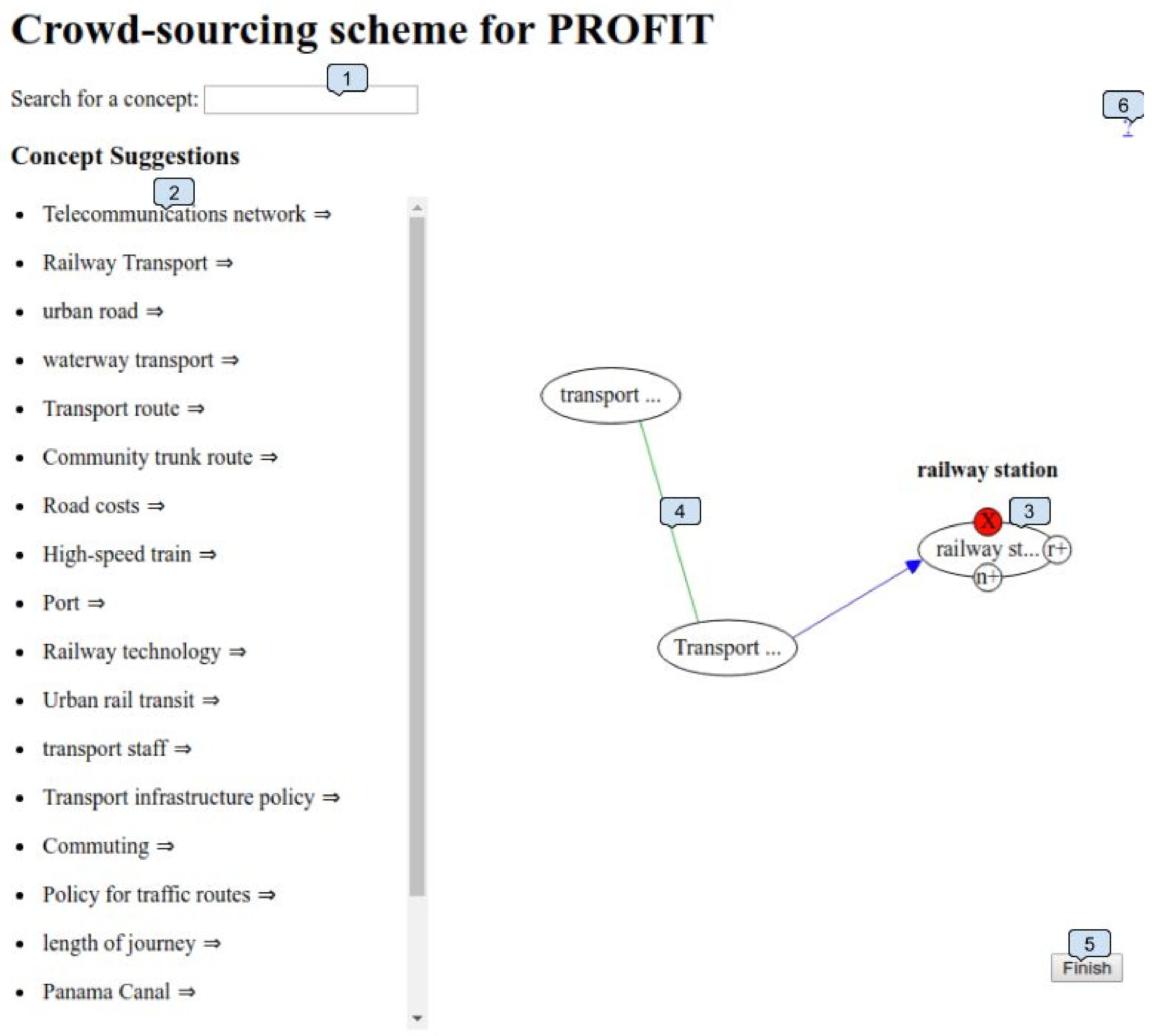

Figure 2: User interface

The user interface is divided into 3 main areas:

- Header at the top of the screen. The header contains the title and the autocomplete input field (1).
- Suggestions panel at the left side. This area contains suggested concepts (2) for the user.
- The interactive canvas. The interactive canvas allows for user interactions and drag&drop functionalities.

Next follows description of UI elements in the figure:

(1) Autocomplete input field. As the user starts typing the dropdown with autocomplete suggestions appear. The autocomplete suggestions represent existing concepts from

the thesaurus. If the user clicks on the autocomplete suggestion, the respective concept is added to the canvas. Otherwise, if the user completes the input with “enter” key, a new concept with the input label is added to the canvas.

(2) Concept suggested to the user. Upon click on any of the concepts the respective concept is added to the canvas. These suggestions are recomputed after each user input and are based on the user preferences and the concepts already in the canvas.

(3) Concept in the canvas. The user can drag and drop the concept to change its location. The user may add new relations (currently only r+ stands for related, n+ for narrower) and remove the concept (double click on red cross).

(4) The relation edges. These edges denote relation between concepts as input by the user. The relations can be deleted: on hover action red cross appears, double click removes the relation. The directed relations are depicted as arrowed edges, the symmetric ones are without arrows.

(5) “Finish” button submits the results.

(6) “?” button displays the help information.

The tool features an additional html template for exposing the saved triples. In the template each triple is accompanied with “upvote”, “downvote”, and “remove vote” buttons.

3.2. Functionalities

The users are able to include the following elements into their input:

- Existing concepts from the PROFIT thesaurus,
- New concepts not included in the PROFIT thesaurus,
- SKOS:broader/SKOS:narrower relation pairs between the concepts (new and existing),
- SKOS:related relations between the concepts (new and existing).

In the next steps we plan to implement the possibility to provide:

- Alternative labels to the existing or new concepts,
- New custom relations from PROFIT Finance ontology between concepts,
- Application of custom classes from PROFIT Finance ontology to concepts

To make the interaction with the tool smooth and intuitive for the user the following essential functionalities have been implemented.

Intrinsic inconsistencies check

According to SKOS standard[8] certain patterns are prohibited. We look for these patterns in real-time and notify the user if any of the prohibited patterns was found, see Figure 3. The notification includes highlighting the prohibited pattern in the canvas and restricting the possibility to submit the input.

[8] https://www.w3.org/TR/skos-reference/#semantic-relations

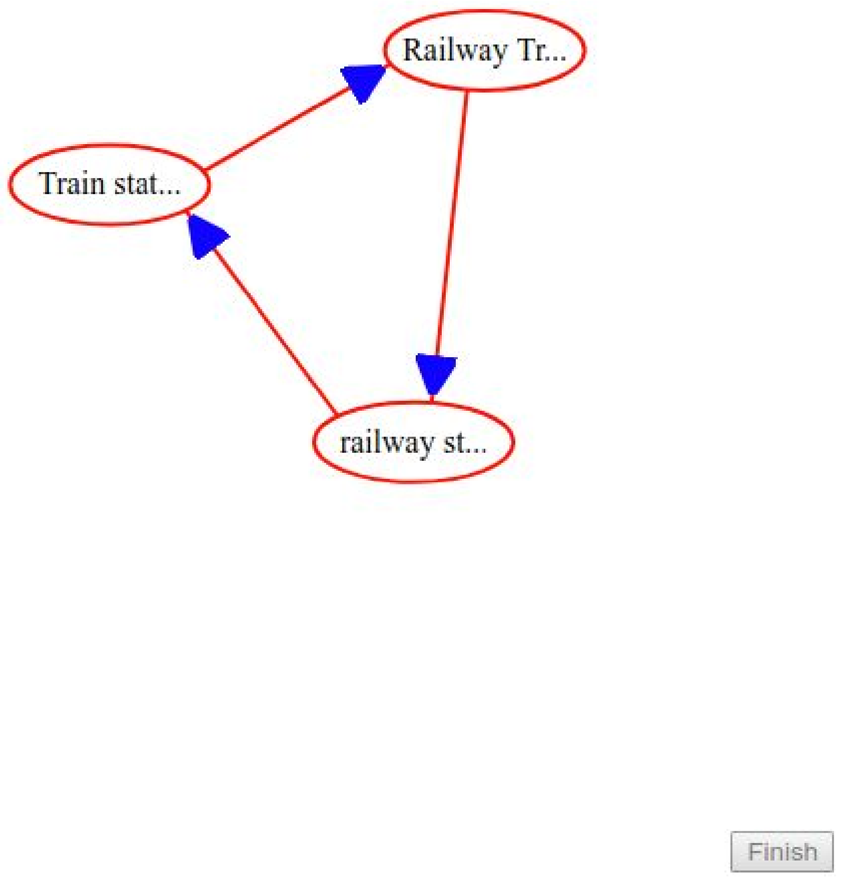

Figure 3: Example inconsistency notification

Currently we prohibit the following to patterns:

1. SKOS:related and SKOS:narrower clash, see Example 28 in SKOS reference.
2. Hierarchical cycles of the form "x SKOS:narrower x", see https://www.w3.org/TR/skos-reference/#L2449. Though it is not prohibited in the SKOS specifications, the semantics of such a cycle in the PROFIT thesaurus is obscure, hence, such cycles are not desirable for our use case.

Next step:

1. Prohibit new concepts without broader.

Feedback: confirming, new, contradicting

One of the important aspects of educational process is the ability to get feedback. In the case of the crowd-sourcing tool the first feedback is automatic and is provided visually. Example of the feedback is in Figure 4.

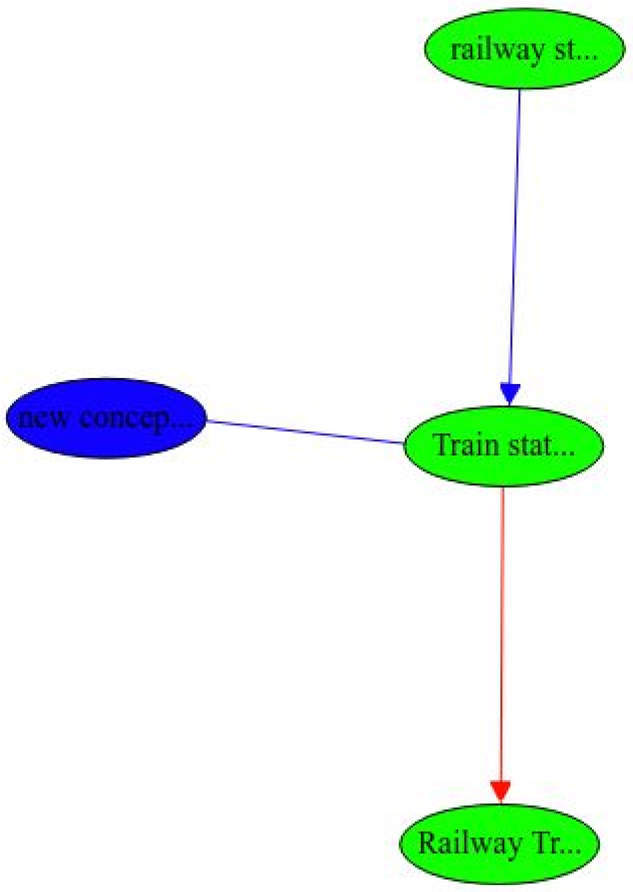

Figure 4:: Feedback example

The coloring is supposed to be intuitive:

- Green stands for confirming;
- Red stands for contradicting;
- Blue stands for new.

Adaptive scoring / thresholding

Each new (in the sense it is not in the thesaurus and does not contradict any existing triple in the thesaurus) triple is saved in the platform and exposed to the users. The users are able to vote on the triple. When (if) the triple collects enough upvotes then the triple becomes "accepted". This acceptance criteria is dynamic and depends on the rest of the user input, namely the threshold is computed as shown in Section "Voting Scheme". Hence, the threshold depends on how well the user's model complies with the existing knowledge, therefore, more radical changes will require more votes.

Explicit vs implicit voting

Explicit voting via clicking upvote or downvote buttons is a usual possibility to provide feedback. However, thanks to the possibility to compute the updates and represent them as triples, the tool implements a possibility of providing implicit voting. Namely, if user1 provides a new triple that is already suggested by user2 then two possibilities exists:

1. If the existing threshold for the triple is less or equal to the new threshold then user1 upvotes the triple;
2. If the new threshold is less than the existing threshold then the new owner of the triple is user1 and user2 upvotes the triple.

Expert (moderator / admin / superuser) decision

When the triple is “accepted” it becomes immutable. There exists the technical possibility to automatically add the triple to the PROFIT knowledge graph, however currently a different workflow is specified. Namely, a moderator is required to review the triple, provide feedback and decide if the triple should be added. The moderator adds the triple manually, possibly providing additional information such as synonyms, translations, relations to existing concepts.

“Warm” start

Currently the user starts working with the visual interface with having suggestions on the concepts the user could use. These suggestions can be personalized, i.e. take the user preferences into account and show the concept from the rated/liked articles.

Moreover, in the next steps we consider a possibility to pre-populate canvas for the users in order to guide the crowd-sourcing process and being able to tackle certain subdomains.

3.3. Usage Scenario

Let us have a look at a concrete usage scenario:

1. User1 sends his input to the platform. The input consists of several concepts and relations between them.
2. The backend computes the "score" of the input. The score takes into account (i) new, (ii) confirming, (iii) contradicting concepts and relations. If the user has provided too little input (currently less than 5 concepts) then she gets an additional penalty.
3. All the new elements provided by the user are stored in the platform and are open for other user to vote upon. The threshold for the new element to be "accepted" is equal the score computed previously. For instance, if the threshold is equal to 5 then if 5 other users upvotes and no one downvotes the element becomes "accepted".

3.4. Examples

Thesaurus:

```
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ns1:     <http://profit.poolparty.biz/> .
@prefix skos:   <http://www.w3.org/2004/02/skos/core#> .
ns1:a   rdf:type skos:Concept ;
    skos:broader ns1:b ;
    skos:broader ns1:c ;
    skos:prefLabel  "a"@en .

ns1:b   rdf:type skos:Concept ;
    skos:prefLabel  "b"@en .

ns1:c   rdf:type skos:Concept ;
    skos:prefLabel  "c"@en .

ns1:d   rdf:type skos:Concept ;
    skos:prefLabel  "d"@en .
```

User proposed model:

```
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ns1:     <http://profit.poolparty.biz/> .
@prefix skos:   <http://www.w3.org/2004/02/skos/core#> .

ns1:a  skos:broader    ns1:c ;
    skos:broader ns1:d .

ns1:b skos:related     ns1:a .
```

New:

```
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ns1:     <http://profit.poolparty.biz/> .
@prefix skos:   <http://www.w3.org/2004/02/skos/core#> .
```

```
ns1:a  skos:broader    ns1:d .
```

Confirming:

```
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ns1:     <http://profit.poolparty.biz/> .
@prefix skos:   <http://www.w3.org/2004/02/skos/core#> .

ns1:a  skos:broader    ns1:c .
```

Contradicting:

```
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ns1:     <http://profit.poolparty.biz/> .
@prefix skos:   <http://www.w3.org/2004/02/skos/core#> .

ns1:a skos:broader     ns1:b .
ns1:b skos:related     ns1:b .
```

$threshold = max(0,\ 5-(2+1)) + 2 * 2 + 1 = 7$

In order the new triple $U =$ `ns1:a skos:broader ns1:d` to be accepted it needs $score[U] = upvote[U] - downvote[U] \geq 7$.

References

[ACPS16] Albin Ahmeti, Diego Calvanese, Axel Polleres, Vadim Savenkov. Handling Inconsistencies Due to Class Disjointness in SPARQL Updates. ESWC 2016.

[AG08] Renzo Angles, Claudio Gutierrez. The Expressive power of SPARQL. International Semantic Web Conference 2008.