OST
Eastern Switzerland
University of Applied Sciences

Reconciliation of Point-of-Interest Locations on OpenStreetMap

Technical Report

Student Research Project Thesis

BSc – Computer Science
Eastern Switzerland University of Applied Sciences
Campus Rapperswil-Jona

Fall Term 2022

Authors	Damian Dasser & Marc Havrilla
Version	23/12/2022
Advisors	Prof. Stefan F. Keller & Nicola Jordan
External Partner	Sascha Brawer

Contents

Part I.

Technical Report

CHAPTER 1

The Current Problem

1.1. Creation of New Elements in OSM

OpenStreetMap (OSM) is a freely editable geographical database. Everyone is allowed to contribute to the project. The collected geographical data is mainly reflected on maps. One part of the data for the geodata database is collected by mappers from scratch. They use all sorts of tools like GPS units, notebooks, cameras and aerial photography. All this data is stored in the OSM database and can be viewed on www.openstreetmap.com. [**wikipedia-osm**]
Various software tools are available to upload data to OSM. Newly identified nodes or ways can be drawn on the map directly. In addition, tags must be added in order to specify what the node or the way actually represents. [**josm-intro**]
Tags consist of two items, a key and a value, which are free format text fields. There already exist hundreds of tags and new ones can be created. [**RT2008**] [**tag-convention**]

1.2. The Problem of Matching External Sources with OSM

The continuous change of the real world circumstances can hardly be managed and reflected on OSM. Consequently, only selective and punctual updates are made and this causes the data on OSM to be neither perfectly accurate nor complete. One of the confounding factors lies in the nature of freely creatable tags. Not everyone agrees and understands the same tags in the same way. It is also questionable whether existing tag values are updated on a regular basis or if required. A brand might change its name after a takeover and thus their shop locations on OSM are in need of a name update.
Data quality assurance tools which compare the data of publicly available sources with OSM geographical data do exist. However, matching OSM data with external data is challenging. There are defined standards for the OSM tags, but their compliance is not enforced. Additionally, the external data is often collected by web scraper. The quality of this data is varying as well. Some datasets might be complete, others miss important information.
Consequently, bringing an external data source together with the OSM database and running a matching algorithm is challenging.

CHAPTER 2

Aim and Scope of the Thesis

Goal of this work is to improve the reconciliation process between external data and OSM. To do so, two scraper systems and two judging system will be evaluated in order to find the most suitable tool for this task. The available scraping systems are Brandy and AllThePlaces.xyz (further referred to as ATP) and the judging systems are called Osmose and OSM Conflator. After the evaluation, the embedding of these tools in OSM landscape will be discussed.
The chosen scraper system and judging system will be used to test the data matching of five brands with the data from OSM. The results will be analysed in order to get some valuable insights how the matching works and how the data quality looks like. Main focus of this work will be on the judging system and how it can be improved to become more generic and scalable. Also, the data provided by the scraper system will be assessed and necessary improvements will be elaborated.
Literature research about information integration will be used to underpin the findings of this work.

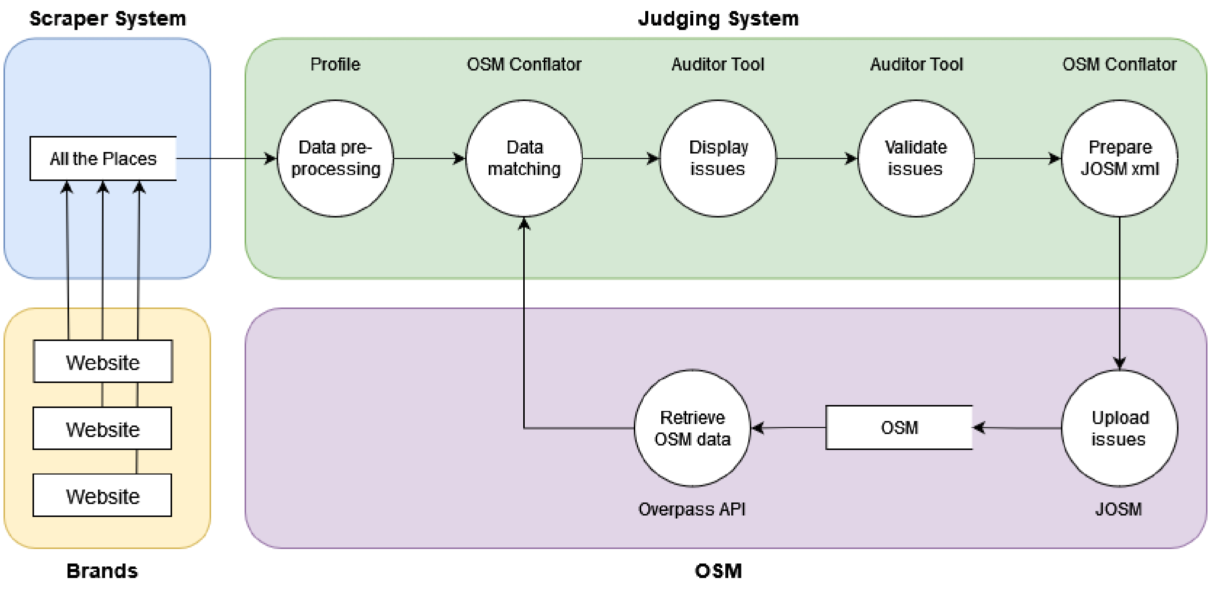


Figure 2.1.: Dataflow diagram

CHAPTER 3

Market Analysis

This project consists of two major software components. This chapter discusses current solutions on the market and their usefulness for this project.

3.1. Available Solutions

While there are several solutions, the selection was narrowed down to use previous knowledge inside the IFS. The solutions are sorted into two categories called scraping system for the data scraping part and judging system for the match making and visualisation. Closed-source software was eliminated without further analysis. It was a key requirement to be able to look into the code and to be able to make changes if need be.

3.1.1. Scraping System

A scraping system accesses a list of pre-defined websites to extract general data including geographical data of Point-Of-Interest (POI). The data found is then stored as GeoJSON and made publicly available for further processing of any kind.

- *Brandy*[1] was created as preliminary work for this thesis. It scrapes a hand full of brands mainly located in Switzerland.
- *AllThePlaces*[2] is actively maintained and unites location data of over 1000 brands all over the world and provides it under the Creative Commons' CC-0[3] license. It is Open Source and code additions and changes can be suggested with merge requests. [**alltheplaces**]

3.1.2. Judging System

These tools are fed by profiles which pre-process data of POI which should be made available on OSM. The first step is to match these points with already on OSM available counterparts in order to avoid duplicates. A list of issues, like points to modify and points to create, is generated and forwarded to a frontend. There an OSM editor can audit these issues and decide on whether to apply them to OSM or not.

- *OSM Conflator*[4] was used in an earlier project of the IFS. Its frontend part is called *OSM Conflator Auditor*[5]. OSM Conflator Auditor visualises one issue at a time and is built to validate issues at high pace. [**osm-conflator-audit-git**]

[1] https://github.com/brawer/brandy
[2] https://github.com/alltheplaces/alltheplaces
[3] https://creativecommons.org/publicdomain/zero/1.0/
[4] https://github.com/mapsme/osm_conflate
[5] https://github.com/mapsme/cf_audit

- *Osmose*[6] has been used in a bachelor thesis [7] this year and delivered great results. Osmose visualises all found issues on a map and requires an editor to update OSM manually.

3.1.3. Developing an Alternative from Scratch

Instead of using third party software everything could have been created from scratch during this project. The amount of added code would have been huge and there were concernes about maintenance and longevity and therefore the idea of building everything from scratch therefore has been discarded as a valid alternative.

[6] https://github.com/osm-fr/osmose-backend and https://github.com/osm-fr/osmose-frontend

[7] *Qualitätsprüfung der Verknüpfungen von OpenStreetMap nach Wikidata* by Timon Erhart and Jari Elmer

3.2. Evaluation

3.2.1. Criteria

For product selection, 12 criteria were identified and described in table 3.1. Not every criterion can be applied on the scraper system and the judging system.

Nr	Title	Description
C01	Single Source of Truth	All information needed to validate a suggested change are available on the same website.
C02	Internationalisation	The frontend for the validation is available in different languages or new languages could be added easily.
C03	Usability	The user interface for validation is intuitive and does not require special training.
C04	Added Code	Additional code needed for implementing further scrapers/profiles is small in size and low in complexity.
C05	Added Deployment	Additional deployment infrastructure, if not already available, is small in size and low in complexity.
C06	Extensibility	New features can be implemented easily.
C07	Documentation	A useful documentation made by the maintainer is present.
C08	Establishment	The application is already established or serves an unfulfilled need such that it can be established easily.
C09	Cost	Cost for additional hard- and software is low.
C10	Update Procedure	Code updates are easy to deploy and fast to push.
C11	Throughput	Manual validation of matching results allows fast processing by the user.
C12	API	The current API allows automatization, provides all necessary data and functions. Little to no adjustment is preferred.

Table 3.1.: Evaluation criteria

Weighting and Rating

For every criteria a weight will be applied individually for the scraping and the judging system. The selected solutions will then be rated on how much they fulfil the criteria. Both weighting and rating is done with numbers from 0 (not applicable/not met) to 3 (very important/well fulfilled).
The weighted score is the product of the weight of the criterion and the rating. All these weighted scores add up to the weighted total which is used for the final decision making.

3.2.2. Scraping System

Description

For the scraper system the two applications Brandy and ATP are evaluated.

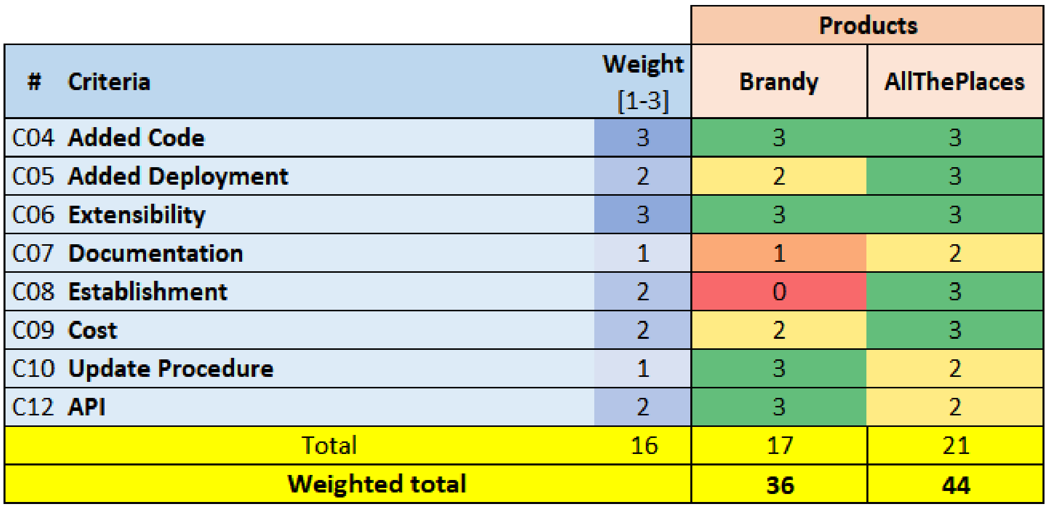
Question

Should AllThePlaces or Brandy be used?

Decision

In the context of our term thesis,
... facing the need of a scraping software,
... we decided for AllThePlaces
... to achieve low maintenance and reach a established community,
... and neglected Brandy,
... accepting downside of lesser control of the deployment.

Rating



			Products	
#	Criteria	Weight [1-3]	Brandy	AllThePlaces
C04	**Added Code**	3	3	3
C05	**Added Deployment**	2	2	3
C06	**Extensibility**	3	3	3
C07	**Documentation**	1	1	2
C08	**Establishment**	2	0	3
C09	**Cost**	2	2	3
C10	**Update Procedure**	1	3	2
C12	**API**	2	3	2
	Total	16	17	21
	Weighted total		**36**	**44**

Figure 3.1.: Scraping system decision matrix

Not Applicable

The following criteria were found to be not applicable for this category, since there is no frontend available. Their weight and rating will be set to 0 for the calculation.

- C01: Single Source of Truth
- C02: Internationalisation
- C03: Usability
- C11: Throughput

C04: Added Code

Weight: The more code added to a scraper the more code has to be maintained and with increasing size it will be harder to find people to maintain it. A weight of 3 seems suitable for this criteria.
Rating: Both projects were created and are maintained by third parties and their scraped data can be used without adjustments to their code. The size of the code for a new scraper is almost equally. Thus both options are rated with 3.

C05: Added Deployment

Weight: The project needs some infrastructure to deploy the code. The financial effort required to host the scraper system is not necessarily high and we rate this with 2.
Rating:

- **Brandy:** We would have to deploy it ourselves. The needed deployment is marginal and thus we rate it with 2.
- **AllThePlaces:** No deployment is needed as the maintainer has been hosting one for years. We apply a rating of 3.

C06: Extensibility

Weight: High extensibility is needed since the scraper system should access a variety of websites and other data provider. The weight of this criteria is therefore 3.
Rating: For obvious reasons both projects had this in mind and allow an easy extensibility which we rate with 3.

C07: Documentation

Weight: As we could always create our own documentation even for existing third party software, we weight this criteria with 1.
Rating:

- **Brandy:** Only small parts are documented and require further documentation which is rated with 1.
- **AllThePlaces:** The documentation level is good, but could be extended. We rate this with 2.

C08: Establishment

Weight: With an established community our contribution will have an impact much faster and the probability of contribution is higher. Since our project is only useful if people use it, a community is needed. A weight of 2 seems reasonable.
Rating:

- **Brandy:** Currently this project does not have a community and is thus rated with 0.
- **AllThePlaces:** A community exists and contributes actively with new scrapers. We rate this with 3.

C09: Cost

Weight: On the one hand there is no budget for this project and thus the costs must be low. On the other hand it might be possible to set up some funding and thus create a budget. We weight this with 2.
Rating:

- **Brandy:** It would need a hosting infrastructure and thus create monthly costs which we rate with 2.
- **AllThePlaces:** No monthly costs are expected and this is rated with 3.

C10: Update Procedure

Weight: We assume that there will rarely be an update and if so that they are not time critical. Thus we weight this with 1.
Rating:

- **Brandy:** The deployment would be controlled by us and thus can be maintained at any time. This is rated with 3.
- **AllThePlaces:** We do not host the service and therefore we cannot update at will. Nonetheless the maintainer updates frequently. We rate this with 2.

C11: API

Weight: The API decides if the data transmission to the judging system is rather smooth or needs code changes. We weight this with 2 because we think even if needed, scraper code changes are implementable quickly.
Rating:

- **Brandy:** A list of scraped brands is provided and every brand's data can be accessed individually. This allows a high flexibility which we rate with 3.
- **AllThePlaces:** The scraped data is provided as GeoJSON in a single ZIP file. The growing file size might lead to problems later. We rate this with 2.

3.2.3. Judging System

Description

Osmose is one of many tools created to support OSM editors to maintain OSM data. Another tool is the *OSM Conflator* and focuses more on working off a list of possible changes. Both tools consists of a backend and a frontend part. The backend handles data analysis and provides suggestions, while the frontend visualises these suggestions for OSM editors.

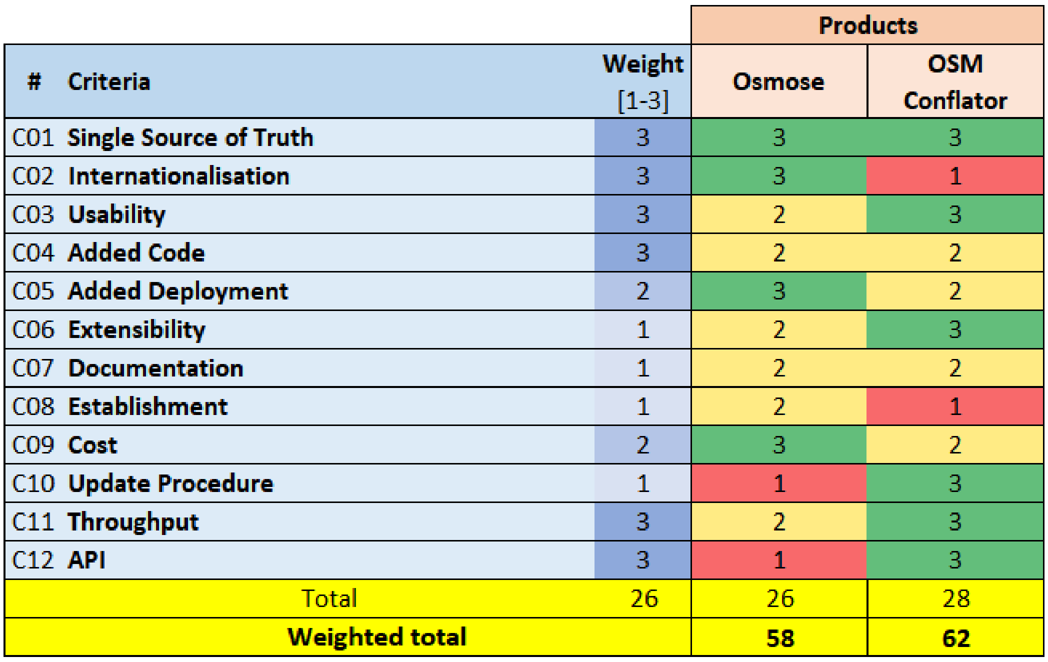
Question

Is Osmose or OSM Conflator the better tool for our task?

Decision

In the context of our term thesis,
. . . facing the need for a front- and backend system,
. . . we decided for OSM Conflator
. . . to achieve a high throughput and good usability,
. . . and neglected Osmose,
. . . accepting downside of non-existing internationalisation and currently small establishment.

Rating



			Products	
#	**Criteria**	**Weight** [1-3]	**Osmose**	**OSM Conflator**
C01	**Single Source of Truth**	3	3	3
C02	**Internationalisation**	3	3	1
C03	**Usability**	3	2	3
C04	**Added Code**	3	2	2
C05	**Added Deployment**	2	3	2
C06	**Extensibility**	1	2	3
C07	**Documentation**	1	2	2
C08	**Establishment**	1	2	1
C09	**Cost**	2	3	2
C10	**Update Procedure**	1	1	3
C11	**Throughput**	3	2	3
C12	**API**	3	1	3
	Total	26	26	28
	Weighted total		**58**	**62**

Figure 3.2.: Judging system decision matrix

C01: Single Source of Truth

Weight: We see this criteria as crucial for the acceptance of the application among the OSM members, we therefore weight it with the highest value of 3.
Rating:

- **Osmose:** The Osmose frontend fulfils this criteria by visualising the suggested change as *issue* on a map. Therefore we rate this with a 3.

- **OSM Conflator:** The OSM Conflator Auditor makes suggestions, visualises them by giving users the option to adjust some details. We rate this with a 3 as well.

C02: Internationalisation

Weight: More languages can result in more users to validate the suggestions which is essential for this project to have an impact. We weight this with 3.
Rating:

- **Osmose:** The frontend not only allows different languages, but also has several languages implemented. This is rated with 3.
- **OSM Conflator:** The Auditor is hard coded English and thus requires some work to allow internationalisation and thus rated with 1.

C03: Usability

Weight: We weight this with 3 as a good frontend supports its user base which is important to make them stay.
Rating:

- **Osmose:** The default frontend is useful, but irritates newcomers with too many options and unusual loading times which we rate with a 2.
- **OSM Conflator:** The Auditor is kept simple and easy to understand. Despite some graphical bugs we rate this a 3.

C04: Added Code

Weight: The more code added to a profile the more code has to be maintained and with increasing size it will be harder to find people to maintain it. A weight of 3 seems suitable for this criteria.
Rating: Both projects need some code addition although only a single file called *Analyser* in Osmose and *Profile* in OSM Conflator. Both files contain some complexity for which we therefore rate both with 2. For a 3 they should need no changes as is the case for both frontends.

C05: Added Deployment

Weight: The project needs some infrastructure to deploy the code. The financial effort required to host the scraper system is not necessarily high and we rate this with 2.
Rating:

- **Osmose:** The deployment will be done on the current Osmose servers. As this would not need anything from our side, we rate this with 3.
- **OSM Conflator:** It requires to host front- and backend. There will not be a lot of traffic and thus require only a small deployment. We rate this with 2.

C06: Extensibility

Weight: We do not see a high demand for extensibility for this project as the sole goal is to compare data from a single scraping system with OSM. Due to the lack of high demand for extensibility we weight this criteria with 1.
Rating:

- **Osmose:** Osmose provides various functions and some might fulfil new use cases. Since the every change needs approval and thus limited. We rate this with 2.
- **OSM Conflator:** We have full control over front- and backend and can therefore build it fully extensible if needed, thus a rating of 3.

C07: Documentation

Weight: As we could always create our own documentation even for existing third party software we weight this criteria with 1.
Rating:

- **Osmose:** A documentation for Osmose front- and backend exists even though it could need some updates. Thus we rate it with 2.
- **OSM Conflator:** Same is applicable for the Conflator documentations and it is rated with 2 as well.

C08: Establishment

Weight: The frontend is already established or serves an unfulfilled need such that it can be easily established and used.
Rating:

- **Osmose:** Osmose already has a community which processes results and updates the OSM data. We rate this with a 2.
- **OSM Conflator:** The Conflator seems not to have an active community and is rated with 1.

C09: Cost

Weight: On the one hand there is no budget for this project and thus the costs must be low. On the other hand it might be possible to set up some funding and thus create a budget. We weight this with 2.
Rating:

- **Osmose:** The used software is Open Source and we do not need hardware for either front- or backend as it will be hosted by the Osmose team. We rate this with 3.
- **OSM Conflator:** All software components are Open Source. Besides that we need to pay for the frontend and backend deployment. Due to low traffic expectations a small setup might be enough and is therefore rated with 2.

C10: Update Procedure

Weight: We assume that there will rarely be an update and if so that they are not time critical. Thus we weight this with 1.
Rating:

- **Osmose:** The update will have to pass a merge request in the Osmose backend and then be deployed on the official Osmose deployment. Both can barely be influenced by us and therefore we rate it with 1.
- **OSM Conflator:** We operate the server and can update at any time. This is rated with a 3.

C11: Throughput

Weight: The number of errors in OSM is significant and continues to rise. There is even an ethical aspect as we do not want to waste lifetime with bad workflows or slow loading times. Fast and efficient data processing is preferred. We weight this clearly with 3.
Rating:

- **Osmose:** We often experienced undesired loading times and had to edit the OSM objects manually which used additional time. We rate this with 2.
- **OSM Conflator:** The frontend is fast and allows bulk uploads to OSM. This allows fast processing and does not rely on copy past by humans. We rate this with 3.

C12: API

Weight: Depending on the API functionality it is easier to implement new services or components. We rate this with 2 as we could adjust the API even though it is undesired.
Rating:

- **Osmose:** An editor is required to transfer at least some of the data manually to OSM. We rate this with 1.
- **OSM Conflator:** OSM Conflator has a function for bulk uploads onto OSM and thus can update validated objects without repetitive manual tasks for which we rate it with a 3.

CHAPTER 4

Theory

Most of the data for POI in OSM comes from sources external to OSM. This data is not uniformly formatted and OSM cannot influence this. These distributed sets of data nonetheless can and should be used to maintain the OSM dataset. Therefore, this chapter will focus on the theoretical aspects of information integration i.e. data heterogeneity, schema matching and data integration. The insights of this chapter will then be used to improve the profiles in OSM Conflator.

This chapter is an excerpt of the book *Informationsintegration* from the authors Ulf Leser and Felix Naumann [**LN2007**]. The content of the relevant topics was translated into English. Furthermore, some sections were summarised in order to provide the reader with a concise overview of the relevant theoretical aspects.

4.1. The Trinity of Basic Issues

To integrate different data sets, there arise three basic issues. The distribution of the data, the autonomy and the heterogeneity of the different data sources.

4.1.1. Distribution

Distribution means the data is stored on different systems. The distribution of data is often a design decision in order to ensure redundancy, backup opportunities or load balancing. Nevertheless, there exists also a type of distribution that was either not necessarily intended or that grew historically.

4.1.2. Autonomy

Independently taking decisions regarding the data structure or the access rights is called autonomy. Autonomy is often the case, if different parties are involved in the data integration. There exist different types of autonomy i.e. design autonomy, interface autonomy, access autonomy and legal autonomy. Due to the relevance for OSM, we will solely focus on design autonomy. If the integration data is retrieved from other web sources, the grade of autonomy is high and hence the autonomy is often called complete autonomy.

Design Autonomy

A data source has design autonomy, if it can independently decide how it make, its data accessible. This means that the data source can independently decide which data format, data model, schema, syntactic representation of the data, usage of keys or unit of values should be used. Having an autonomy in these aspects, is the main reason for heterogeneity in information integration.

4.1.3. Heterogeneity

Heterogeneity is the main issue during information integration. It means two information systems do not have the same methods, models and structures for accessing the data. Heterogeneity can be caused by distribution, but not necessarily. Autonomy has more impact on heterogeneity. This is even the case, when the data sources store the same data. Furthermore, heterogeneity is the result of different requirements, different developers and historically grown applications. The following sections will provide an overview of the different kinds of heterogeneity.

Technical Heterogeneity

Technical heterogeneity is about the communication of the information system with the data source. Hence, technical heterogeneity deals with the query language, communication protocol, canned queries and the format for exchanging data e.g. XML, HTML, etc.

Syntactic Heterogeneity

Syntactic heterogeneity means that the same information is represented differently. For example in one database the data is stored as Unicode and in another ASCII is used.

Heterogeneity on the Data Modelling Layer

Heterogeneity on the data modelling layer exists when the information system and the data source have different data models. One database could be object oriented, whereas the other database is relational or XML. The issues are especially obvious when the data source stores the same data, but different data models are used.

Structural Heterogeneity

If two schemas are different even though they reflect the same real world circumstances, then this is called structural heterogeneity. Or in other words, different schemas describe the same world.

Schematic Heterogeneity

It is called schematic heterogeneity, if two schemas use different elements in order to reflect the same real world circumstances. In a relational model information can be stored as a relation, as attributes or as values. How information is stored has an impact on the query languages as they need to explicitly call the relation or the attribute.

Semantic Heterogeneity

Semantic heterogeneity is about the interpretation of data in order to get information. However, two identical schemas could have different semantics. For this reason the context has to be considered as well. The problem of sematic heterogeneity affects the schema elements as well as the stored values. The most common semantic conflicts are synonyms and homonyms. Synonyms are names which are different, but have the same intension. Homonyms are names which are identical, but have a different intension. "Fly" is an example for an homonym,as it could refer to the insect or the movement.

4.2. Schema Matching

The aim of schema management is merging heterogenous data into one information set. Schema management consists of several tasks i.e. schema integration, schema mapping, schema matching and data transformation. Based on the scope of this thesis, the main focus will be on the schema matching.
Another important concept of schema management is correspondence respectively mapping. A mapping describes the semantic relation between elements of different schemas. A mapping will either be created by hand or semi automatically by schema matching. Afterwards the mappings will be used for schema integration or schema mapping.

4.2.1. Difficulties in Finding Correspondence

The following list provides an overview of the challenges of finding a correspondence between schemas.

- Grand schemas: In complex applications the schemas can become very complex. Furthermore, attributes might have the same names. In such schemas it is difficult to keep an overview.
- Unclear schemas: Besides the size of schemas also the complexity makes understanding the schemas difficult. In relational databases the foreign key relationships can create a complex structure.
- Foreign schemas: The creation of a correspondence between two schemas is difficult, as the knowledge of one scheme is often missing or it lacks documentation.
- Heterogenic schemas: Schemes for complex applications are almost always highly heterogenous.
- Foreign language schemas: To specify correct correspondence if the schemas are written in different languages, is challenging.
- Cryptic element identifiers: Restrictions in the length of attributes or relation names often lead to hard to read schemas.

As a consequence the correspondence could be either false-positive or false-negative. A correspondence is false-negative in the case of the correspondence being missing.

4.2.2. What is Schema Matching?

Schema matching is the process of automatically identifying the correspondence between two semantic equivalent elements of two different schemas. For this purpose a matcher is used. A matcher analyses the schema, the structure, the integrity conditions as well as some data examples. Based on this analysis, the matcher creates some recommendations for correspondences, which have to be validated. Are two schemas identical it is called a full match [**RB2001**].

4.2.3. Classification of Schema Matching Methods

The fundamental approach to identify 1:1 correspondences is to compare every attribute of the source schema to the attributes of the destination schema based on an similarity measure. Are two attributes sufficiently equal, a correspondence will be proposed.
The following sections will provide a classification of the similarity measure and a description of the classes.

Schema-based Schema Matching

A schema-based matcher only considers the schema of a database. The schema includes the name and structure of the schema elements and additional information, like data types and key properties. In order to match the schema elements the edit distance is often used as a similarity measure. However, there exist further means of similarity measures:

- Name equivalence: For schema elements with the same name, a correspondence will be created.
- Equality after normalisation: Normalisation is used for example to bring a word into its root form by using stemming, abbreviations are resolved, or naming of the elements can be translated in one language.
- Synonymy: Provided that a thesaurus is available for the domain, synonyms can be considered.
- Hypernymy: Given that a thesaurus is available, elements, which are in a is-a relationship, can be identified. Hypernymy is a strong indicator for a correspondence.
- Similarity of name: To check the similarity of the element names, the edit distance is used.

Instance-based Schema Matching

An instance-based schema matcher considers the instances of a database, meaning the data itself rather than the schema elements. The matcher extracts for this purpose data instances from the different databases and compares them. If the properties are almost similar, there is the possibility for a correspondence.
It is differentiated between vertical schema matching and horizontal schema matching:

- Vertical schema matching: For vertical schema matching, predefined properties of attribute values will be taken and compared to each other. If the properties correspond to each other a correspondence exists. In simple words, columns of data are compared to each other.
- Horizontal schema matching: First, a horizontal matcher looks for duplicates of duplicated instances in the different databases by comparing the tuples. If the matcher detects corresponding instances, it will check the attribute values of these duplicates. The correspondence does exist, where the most attribute values match.

An important condition for instance-based matching is the availability of instances for both schemas. If this is not the case, some example data can be inserted. Are all conditions met, the instance-based matching is superior to the scheme-based schema matching.

Combined Schema Matching

Combined matchers use different techniques, which has the advantage that different techniques can be used in different situations.

- Hybrid schema matching: Hybrid matchers nest various matching techniques.
- Composed schema matching: Composed matchers use various matching techniques independently of each other. In each step the result will be combined.

Extension

Until now, the 1:1 schema matching was discussed. However, this can be extended in both directions. A matcher can try to identify n:1 and 1:n correspondences. A correspondence is called polyvalent, if more than one element is affected by the correspondence within a schema. A typical example is the street name and the street number on one side of the correspondence and on the other side it is combined in an element of the address. Matching procedures, which identify polyvalent correspondence are called complex matcher. The complex matcher does not only have to look for correspondences between two schemas, it also has to check every subset of both schemas.

Global Schema

In the previous chapters, the matching between single attributes was discussed. The goal of global schema matching is to get a complete matching between two schemas. For global schema matching, all possible correspondences will be marked with a similarity measure. Afterwards the maximum weighted matching algorithm is used to find an overall solution.

4.3. Data Integration

The data itself can be error prone. A date could be formatted as mm/dd/yyyy or yyyy-mm-dd or the house number before the street name, there could be a wrong letter in a name, or a postcode does not match the city name. [**ms-address-format**] All of them are classified as data error. Duplicates are another type of error and should be eliminated to prevent extra workload in the future. Data quality classifies if a data entry is trustworthy. At last, the completeness needs to be checked.

4.3.1. Data Cleaning - The Art of Removing Data Errors

Data errors can be classified in four groups which shall be explained here.

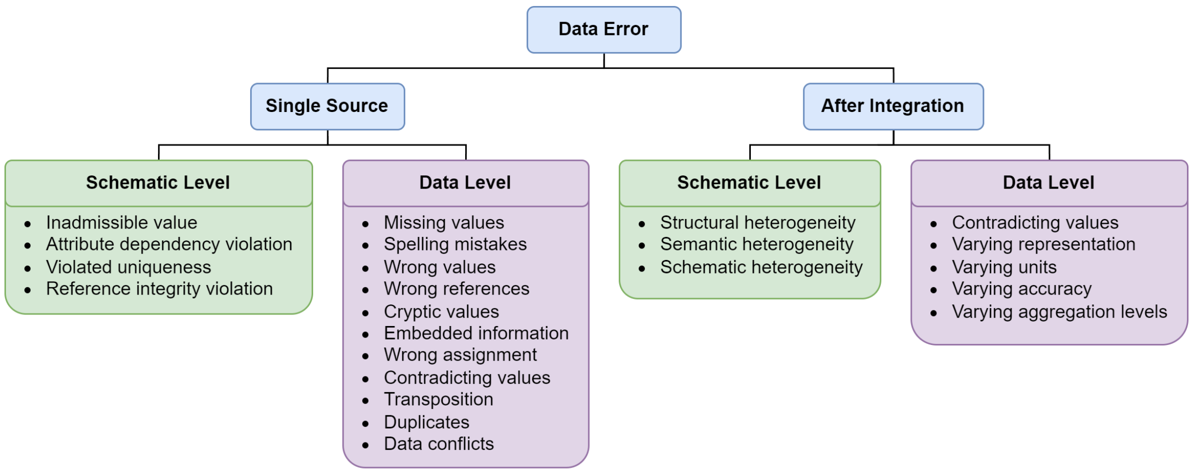


Figure 4.1.: Data error classification

Error Classification

Single Source Schematic Level Errors

- Inadmissible value: A value lies outside the defined range or list of possible values, e.g. an age attribute containing a negative value.
- Attribute dependency violation: A predefined attribute dependency violation, e.g. an attribute $age = today - birthday$ with $age = 22$ and $birthday = 01.01.2020$.
- Violated uniqueness: A unique value appears more than once.
- Reference integrity violation: The value of a foreign key points to a non-existing entry.

Single Source Data Level Errors

- Missing values: Attributes with *null*
- Spelling mistakes: As they happen from time to time, e.g. $Zuurich$ or $Raperswil$. These errors can usually be fixed by a domain expert.
- Wrong values: A city named $Bahnhofstrasse$. This error is only correctable by checking the real world.
- Wrong references: Referring to an existing entry which is not the correct one.
- Cryptic values: Unclear values like an abbreviation which could stand for several equally meaningful words.
- Embedded information, violation of atomicity: E.g. adding the creation date of a movie in its name attribute.
- Wrong assignment: Correct value, but in the wrong attribute so it just needs to be swapped.
- Contradicting values: For example the postcode 1000 and the city name $Zurich$. The values are valid on their own, but together they contradict each other.
- Transposition, varying order in an attribute. The name attribute with an entry $MaxMustermann$ and one with $MusterfrauMia$
- Duplicates: A real world object described twice.
- Data conflicts, duplicates with contradicting values. A real world object described twice, but at least one value is different and not null.

Schematic Level Errors After Integration

Structural, semantic and schematic heterogeneity was described in Section 4.1.3.

Data Level Errors After integration

- Contradicting values: One source with $age = 42$ another source with $birthday = 01.01.2020$ for the same object.
- Varying representation: $type = node$ from one source and $type = 1$ from the other.
- Varying units: Like $length = 12m$ and $length = 1200cm$. If known, it can be converted during integration.
- Varying accuracy: One source describing the latitude with 7 decimal units, while the other sources only uses 3.
- Varying aggregation levels: The attribute movie actors lists only main actors in a source while another source lists every actor.

Origin of Data Errors

Data errors can occur due to **incorrect capture**. Be it through typing errors, insertion of dummy values to bypass null-checks, automated entries through scanners and other sensors or because the input field is misinterpreted and hence filled with the wrong content. This can lead to duplicates or other data errors.
Data can **age** and get outdated. A name can change due to marriage, or a credit score is outdated due to financial changes. Some data age quite fast while other data changes only rarely.
Transformation is usually done on the schema level to change units or transpositions. During these transformations new errors can arise. A wrong constant for multiplication could be one of the reasons.
Integrating data from different sources can contain conflicting values even though they try to represent the same. This can occur since the data sources are autonomous and therefore the creation of duplicates cannot be prevented.

Error Handling

Handling data errors consists of three processes, namely profiling, assessing and monitoring. Detailed knowledge of the application and the domain is a requirement for all of them.
Profiling describes the process of analysing data and their pattern. Detected patterns can then be used during the assessment.
For an **assessment** of data errors a list of requirements for the data is created. These requirements can be derived from the profiles or be created by the person doing the assessment.
To measure the success of the error correction after an assessment a **monitoring** process is implemented.

4.3.2. Duplicates

Multiple tuples describing the same real world object are defined as duplicates. Duplicates can occur in a single database due to errors during creation of new entries or they can occur after a data integration process when the source systems provide overlapping datasets.

Effectiveness

Duplicate detection can be done by pairwise comparison of all tuples which results in a similarity score. If this score exceeds a defined threshold the two tuples are considered a duplicate. The comparison algorithm needs to be effective and efficient. The effectiveness can be determined by its precision and its recall.
The precision measures the TruePositives out of all the detected positives:

$$\text{Precision} = \frac{TruePositives}{TruePositives + FalsePositives}$$

A high precision confidently indicates two tuples of being a duplicate and a manual check is unnecessary. Actual duplicates with lower similarity however achieve a lower score and go undetected.
A recall measures the TruePositives out of all the actual positives:

$$\text{Recall} = \frac{TruePositives}{TruePositives + FalseNegatives}$$

A high recall indicates a good detection rate even though this might contain many FalsePositives since the recall score can be improved by lowering the similarity requirements.
The problem of which score should weight more needs an individual decision for every use case. Usually it is a combination of both and is calculated as *f-measure*:

$$\text{f-measure} = \frac{2 * Precision * Recall}{Precision + Recall}$$

Efficiency

To detect all duplicates with a given similarity score, threshold and a cardinality of n an algorithm requires $\frac{n^2}{2} - n$ comparisons. Even with simple comparisons the workload in $\mathcal{O}(n^2)$ is too high. This calls for optimisation.
Avoidance of tuple comparisons is one of those optimisations. The data pool is split into partitions and comparisons are only done in those. This reduces the total number of comparisons. Since this partitioning might not be perfect, it degrades the recall score.
The conflict of objectives between precision, recall, and efficiency does not allow for a perfect algorithm. Increasing one score lowers another and thus a trade-off has to be made.

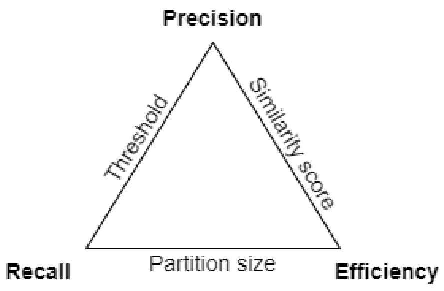


Figure 4.2.: Conflict of objectives in duplicate detection

4.3.3. Data Fusion

During data fusion the goal is to merge duplicates from different sources so that each real world object is represented once at the most. This process not only eliminates duplicates, but also combines them. A tuple from one source with missing information can be enriched by a tuple from another source describing the same real world object. However, data from the second source can as well conflict with data from the first. These conflicts need to be solved somehow.

Conflicts and Resolution

Two conflicting tuples can be in four different relationships. These are visualised with example entries in table 4.1 which will be referred to by their Tuple ID.

- Equality: All attribute values are equal in both tuples. This is the case for (A) and (B).
- Subsumption: A tuple subsumes another if it has fewer zero values (B) and has the same value as the other tuple (C) in each attribute with a non-zero value and thus contains more information.

- Complementation: Two tuples complement each other when none of them subsumises the other and if for every null value one of the tuples provides a value while every not null value is identical. Only tuple (C) and (D) are in this relationship.
- Conflict: When two not null values are different the situation is called conflict. Tuple (E) is in conflict with every other tuple.

Tuple ID	ID	Name	Age	Married
(A)	7	Peter Griffin	42	Yes
(B)	7	Peter Griffin	42	Yes
(C)	7	Peter Griffin	42	*null*
(D)	7	Peter Griffin	*null*	Yes
(E)	7	Peter Griffin	999	No

Table 4.1.: Example of Tuples

Fusion with Unions

Two basic operations provided by relational algebra for combining two relations into one are join and union.

Union:

The union operation requires both source schemas to be identical. The result will contain all tuples from both sources while duplicates will be eliminated.

Outer-Union:

The Outer-Union relaxes the requirements for the union operation. The sources are not required to have an identical schema. The result will unite all attributes of both sources and set missing values to *null*.

Minimum Union:

The minimum union combines entries only if one of them subsumises another like tuple (B) and (C) in table 4.1.

4.3.4. Information Quality

The usefulness of data for its desired purpose is called information quality and sometimes data quality. Information quality depends on individual use cases. Criteria for the calculation have to be defined and can include relevance, completeness, accuracy or trustworthiness. The resulting score is referred to as *fitness for use*. Figure 4.3 shows the four classifications and their criteria.

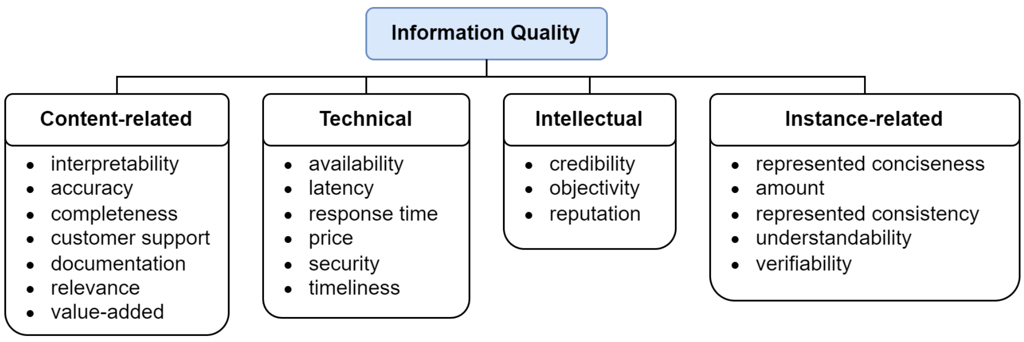


Figure 4.3.: Information quality classification

CHAPTER 5

Embedding of OSM Conflator in the OSM Landscape

The aim of this chapter is to provide an overview of how the data reconciliation of external data and OSM data with the OSM Conflator is implemented. It is based on the information given by the author of OSM Conflator and his code. [**osm-conflator-wiki**] [**osm-conflator-git**] [**osm-conflator-audit-git**] In addition, it will be illustrated, where the profiles interact with the OSM Conflator and how the data is processed.
The first diagram provides a basic overview of how the different systems are connected. Web Scrapers set up at ATP collect data from different brand websites. Then the data is obtained by the OSM Conflator. The OSM Conflator tries to identify discrepancies between the data of ATP and OSM. Finally, the OSM Conflator uploads the issues to OSM.

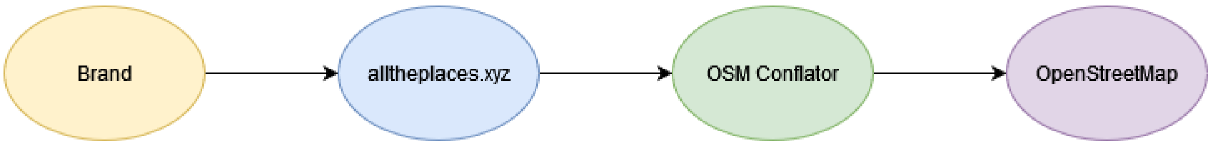


Figure 5.1.: Overview landscape

The context diagram in Figure 5.2 provides an overview of how the OSM Conflator is embedded in the OSM landscape. The OSM Conflator receives the external data from the ATP website. The data from OSM is received via the Overpass API by a query created in the OSM Conflator. Once the OSM Conflator did the data matching, the identified discrepancies are passed to the Auditor tool. The Auditor visualises the discrepancies and allows their validation. The validator is able to either accept or reject the identified issues. Afterwards, the data will be transferred back to the OSM Conflator, where a XML file will be created and sent to the JOSM Editor. The JOSM Editor is a software tool used to upload data to OSM. [**RT2008**]
The container diagram shown in Figure 5.3 provides a more detailed view of how a profile and the matching algorithm are connected within the OSM Conflator. Currently, for each brand an individual profile is needed. A profile pre-processes the external data from ATP. The pre-processed data will then be passed on as a SourceNode object to the conflator.py class, where the matching algorithm is located.

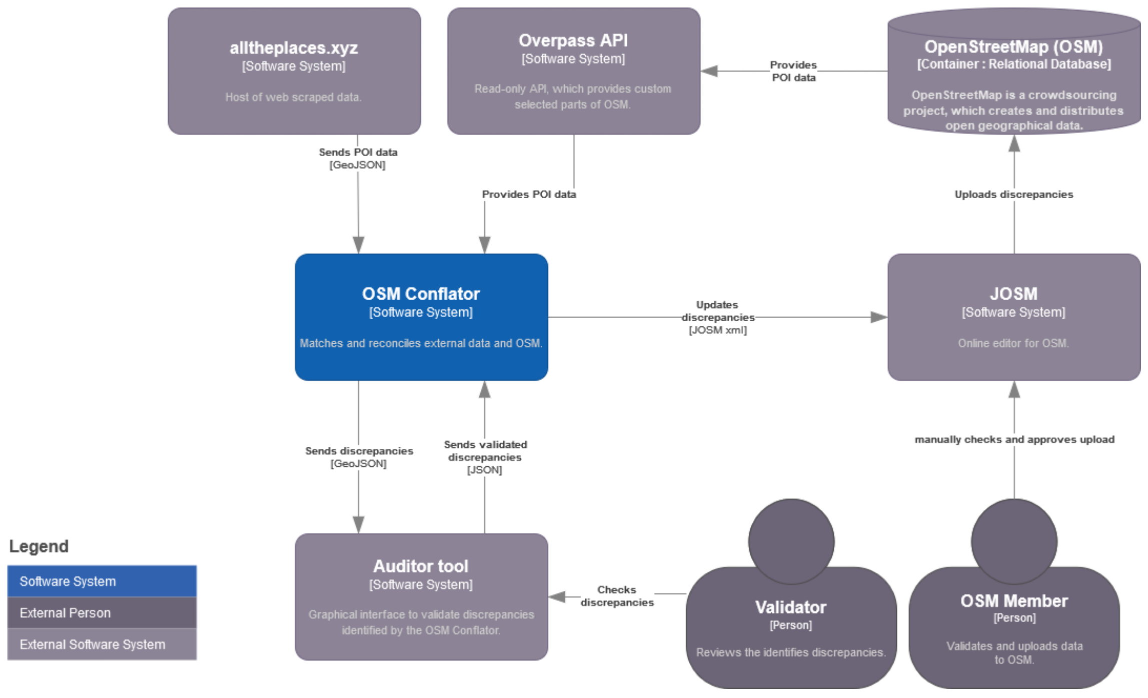

alltheplaces.xyz
[Software System]
Host of web scraped data.
Overpass API
[Software System]
Read-only API, which provides custom selected parts of OSM.
Provides POI data
OpenStreetMap (OSM)
[Container : Relational Database]
OpenStreetMap is a crowdsourcing project, which creates and distributes open geographical data.
Sends POI data [GeoJSON]
Provides POI data
Uploads discrepancies
OSM Conflator
[Software System]
Matches and reconciles external data and OSM.
Updates discrepancies [JOSM xml]
JOSM
[Software System]
Online editor for OSM.
Sends discrepancies [GeoJSON]
Sends validated discrepancies [JSON]
manually checks and approves upload
Legend
Software System
External Person
External Software System
Auditor tool
[Software System]
Graphical interface to validate discrepancies identified by the OSM Conflator.
Checks discrepancies
Validator
[Person]
Reviews the identifies discrepancies.
OSM Member
[Person]
Validates and uploads data to OSM.

Figure 5.2.: Context diagram

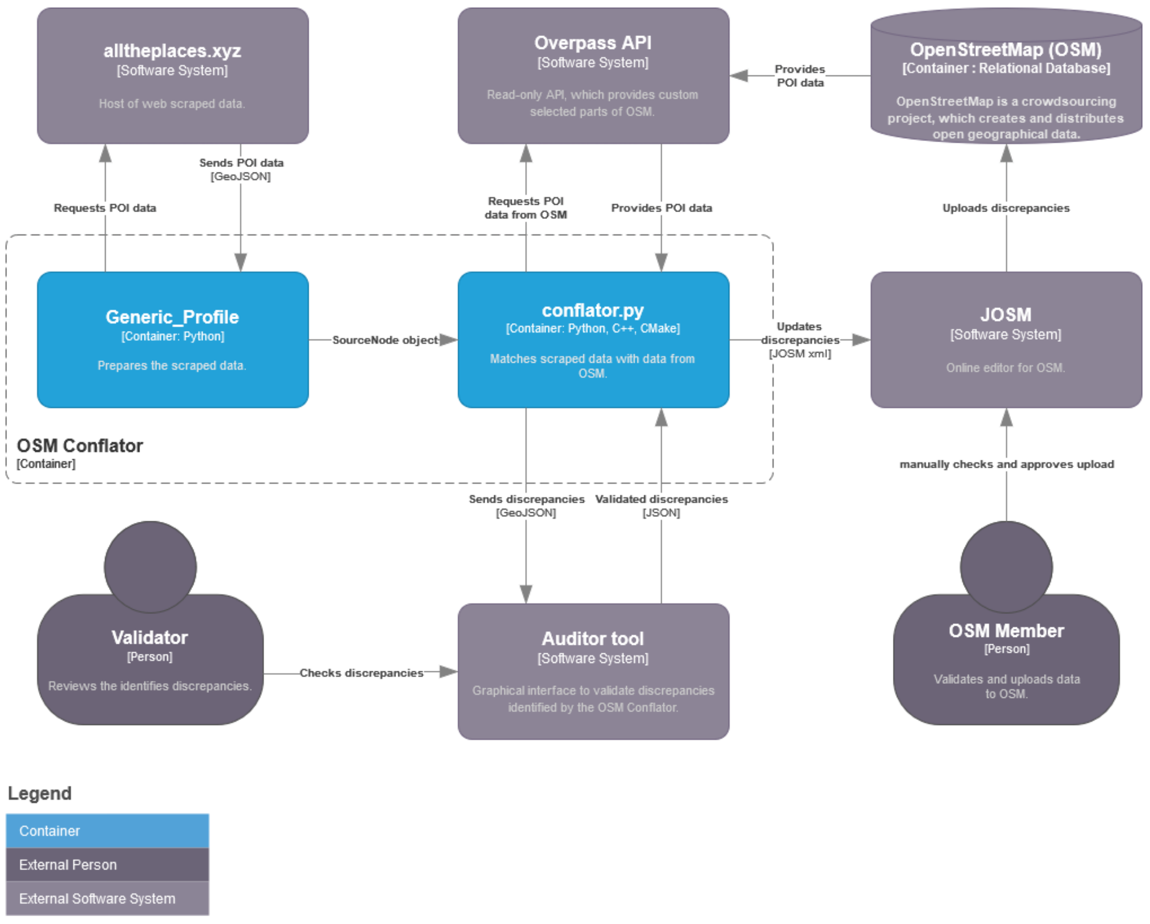

alltheplaces.xyz
[Software System]
Host of web scraped data.
Overpass API
[Software System]
Read-only API, which provides custom selected parts of OSM.
Provides POI data
OpenStreetMap (OSM)
[Container : Relational Database]
OpenStreetMap is a crowdsourcing project, which creates and distributes open geographical data.
Sends POI data [GeoJSON]
Requests POI data
Requests POI data from OSM
Provides POI data
Uploads discrepancies
Generic_Profile
[Container: Python]
Prepares the scraped data.
SourceNode object
conflator.py
[Container: Python, C++, CMake]
Matches scraped data with data from OSM.
Updates discrepancies [JOSM xml]
JOSM
[Software System]
Online editor for OSM.
OSM Conflator
[Container]
manually checks and approves upload
Sends discrepancies [GeoJSON]
Validated discrepancies [JSON]
Validator
[Person]
Reviews the identifies discrepancies.
Checks discrepancies
Auditor tool
[Software System]
Graphical interface to validate discrepancies identified by the OSM Conflator.
OSM Member
[Person]
Validates and uploads data to OSM.
Legend
Container
External Person
External Software System

Figure 5.3.: Container diagram

CHAPTER 6

Profile Implementation and Assessment

6.1. Creation of a Specific Profile

We created five profiles for specific use cases. We picked available GeoJSON files from ATP with data origin in Switzerland and an entry size below 300. Our decision fell on the brands *Fust*, *Aldi*, *Coop Vitality*, *Jumbo* and to vary the types the *public toilets Zurich city*.
The template profile contains several *#TODO x* comments, where the *x* represents the step number this todo should consider. The process of profile creation requires knowledge of the data quality of both source and OSM. While we might already know the source's quality from the creation of its scraper, we might not be aware of the quality of elements already present in OSM. The step-by-step template takes this into account.

Step 1 - The Data Source

To get a first idea of the data a minimal working profile can be created completing the variables *download_url*, *source*, *query* and *data_url*. This profile can be executed and provides an output file which can be imported into OSM Conflator Auditor. This first audit is only superficial and is meant to get a feeling for the data by looking at five to twenty elements. You might find that OSM objects use other values than the one you used in your *query*. Adjust it, if needed. If the problem is more complex than the query syntax allows, we can adjust it later in step 4 or even step 5.

Step 2 - The Schema Adjustment

Ideally the scraper fulfils our requirements and this step can be skipped, but for now every scraper provides good data quality. Sometimes information like the address is stored together in one tag instead of using atomic tags. For example we would have to split *addr:street_address* into *addr:street* and *addr:housenumber*. Depending on the source other data transformations are required.

Step 3 - Distances

Depending on the brand's industry the maximum matching distance and the duplicate distance need to be adjusted. We provided default values which worked well for us. Running the OSM Conflator after adjusting these variables provides information on the potential effectiveness.

Step 4 - Increasing Complexity

Sometimes the matching problem is more complex than just increasing distances. There might have been a takeover and the elements in OSM still use the old brand's name as we have seen with *Coop Bau+Hobby* and *Jumbo*. Not using hard coded values is preferred, but some real world problems do not allow such luxury.

Step 5 - The Alternative

An alternative to step 4 and its usage of the *matches()* function is the *weight()* function. It is less flexible as it has only access to the OSM element's data. Ideally this step is not needed and can be skipped.

Step 6 - Final Cleaning

This step is to polish the dataset given to the OSM Conflator Auditor and to increase the comfort of the auditor. While tags from *master_tags* will preselect values for the auditor, the *tags.pop('tag_name')* gets rid of tags which might just pollute OSM with useless information.

6.2. Outcome Statistics

To measure the correctness and usability of the created profiles we verified them manually. In this chapter we discuss the findings summarised in Table 6.1.

Source	Fust	Aldi	Coop Vitality	Jumbo	Public Toilets Zurich
Source Entries	145	235	88	124	84
Correct Match	105	217	66	103	82
No Match - Poor OSM Data Quality	7	1	2	3	0
No Match - Coordinates Too Far Off	4	3	1	3	0
False Match	0	0	0	0	0
Correct New Entry	29	14	19	15	2
Questionable Input Data	0	0	0	0	0

Table 6.1.: Matching Statistic

Explanation for the Evaluated Aspects

- Source Entries: Number of entries provided by ATP. This number contains all instances collected by the web scrapers from the publicly available POI data.
- Correct Match: Number of correct matches between an existing entry from OSM and an entry of the external source. Only if an OSM element represents the same real world element as the element from the external data source, it is considered as a correct match. Consequently, the tags and tag values did match between OSM and external source.
- Poor OSM Data Quality: Entries that should have matched, but did not because of the poor quality of the entries from OSM. Some tags contain values far from the expected like a *shop=electronics* is tagged as *shop=bathroom_refurbishing*.
- Coordinates Too Far Off: Entries from external sources that come with coordinates far away from the actual location.
- False Match: Is an element matched with a wrong element from OSM, we consider it a false match. There is no margin, if an element is matched it has to be correct, otherwise this match is counted as a false match. We tried to optimise our profiles as to not generate false matches with the downside of a higher number of falsely not-matched elements.
- Correct New Entry: Entries of this category would create new elements in OSM. To out rule cases where a shop moved and it was not possible to locate its previous position we verified these cases manually.
- Questionable Input Data: Some external data sources contain wrong data points, like wrong tags or tag values. An example could be the city name tag value only contains numbers and the tag postcode only contains letters.

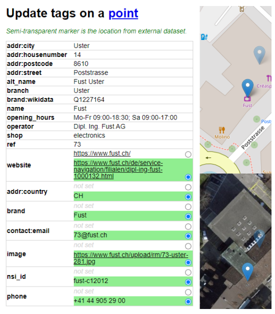
6.3. Findings from Profile Implementation

6.3.1. Matching Result

During this task, five profiles were created and the matching result between external data and OSM data was analysed. Even though this is not a meaningful sample size, it provides a good first impression. Significant differences in the data quality of the different data sources were noticed. Depending on the profile, entries, which did not exist in OSM, range from 2.3% for one profile up to 21.5% for another profile. On average 12.6% of the entries did not exist on OSM before. These entries will be added to OSM as new entries.
In some cases, a matching was not possible due to poor quality. These entries were manually reviewed and assigned to either poor OSM data quality or questionable coordinates provided by the external data source. On average 1.6% of the entries could not be matched due to unprecise coordinates. Another 1.9% could not be matched as the data quality and schemas in OSM were in an insufficient state.

6.3.2. General Issues in Schema Matching

The schemas of OSM and ATP are very heterogeneous. This is due to the fact that the databases are highly autonomous. In an ideal world, standards would exist and the attributes or tags of an OSM element and the attributes from ATP would therefore match fully. However, for the matching purpose, a full match is not necessarily required, as not all attributes are relevant for the matching. Having a large intersection of the same attributes, opens opportunities to create a very specific and accurate profile, which will result in a proper query to OSM. [**alltheplaces-git-dataformat**] [**tag-convention**]

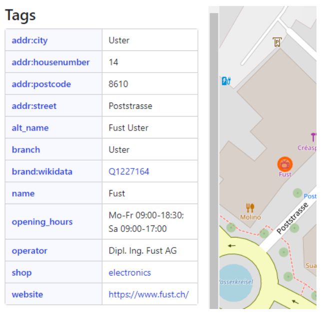


Update tags on a point

Semi-transparent marker is the location from external dataset.

addr:city	Uster	
addr:housenumber	14	
addr:postcode	8610	
addr:street	Poststrasse	
alt_name	Fust Uster	
branch	Uster	
brand:wikidata	Q1227164	
name	Fust	
opening_hours	Mo-Fr 09:00-18:30; Sa 09:00-17:00	
operator	Dipl. Ing. Fust AG	
shop	electronics	
ref	73	
website	https://www.fust.ch/	○
	https://www.fust.ch/de/service-navigation/filialen/dipl-ing-fust-1000132.html	●
addr:country	*not set*	○
	CH	●
brand	*not set*	○
	Fust	●
contact:email	*not set*	○
	73@fust.ch	●
image	*not set*	○
	https://www.fust.ch/upload/rm/73-uster-281.jpg	●
nsi_id	*not set*	○
	fust-c12012	●
phone	*not set*	○
	+41 44 905 29 00	●

Figure 6.1.: Conflator Auditor view of the large attribute intersection



Tags

addr:city	Uster
addr:housenumber	14
addr:postcode	8610
addr:street	Poststrasse
alt_name	Fust Uster
branch	Uster
brand:wikidata	Q1227164
name	Fust
opening_hours	Mo-Fr 09:00-18:30; Sa 09:00-17:00
operator	Dipl. Ing. Fust AG
shop	electronics
website	https://www.fust.ch/

Figure 6.2.: OSM view of the large attribute intersection

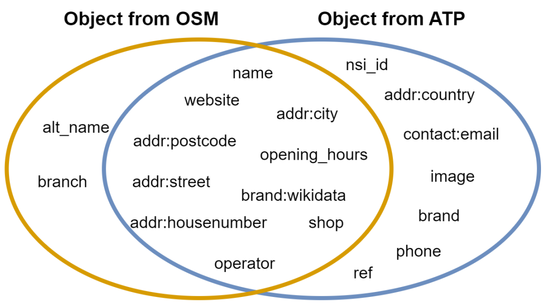
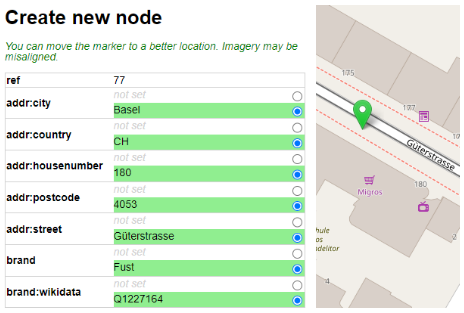


Figure 6.3.: Example of a large attribute intersection

Even though there are schemas with a large number of attributes which are similar, this is often not the case. Whereas the schemas of the external data sources are quite consistent, those of OSM objects are varying.
As the attributes of OSM object are not consistent, the most common attributes within the OSM objects in question have to be identified. Based on these attributes the query for OSM will be created. However, the consequence of an unspecific query is that a lot of elements will be loaded which have a performance impact on the Overpass API as well as the OSM Conflator.

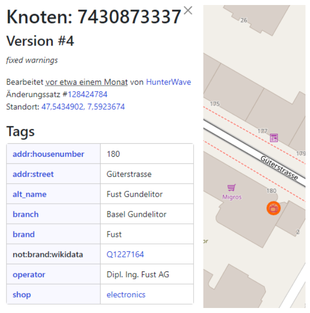


Create new node

You can move the marker to a better location. Imagery may be misaligned.

ref	77	
addr:city	*not set*	○
	Basel	●
addr:country	*not set*	○
	CH	●
addr:housenumber	*not set*	○
	180	●
addr:postcode	*not set*	○
	4053	●
addr:street	*not set*	○
	Güterstrasse	●
brand	*not set*	○
	Fust	●
brand:wikidata	*not set*	○
	Q1227164	●

Figure 6.4.: Conflator Auditor view of the small attribute intersection



Knoten: 7430873337

Version #4

fixed warnings

Bearbeitet vor etwa einem Monat von HunterWave
Änderungssatz #128424784
Standort: 47,5434902, 7,5923674

Tags

addr:housenumber	180
addr:street	Güterstrasse
alt_name	Fust Gundelitor
branch	Basel Gundelitor
brand	Fust
not:brand:wikidata	Q1227164
operator	Dipl. Ing. Fust AG
shop	electronics

Figure 6.5.: OSM view of the small attribute intersection

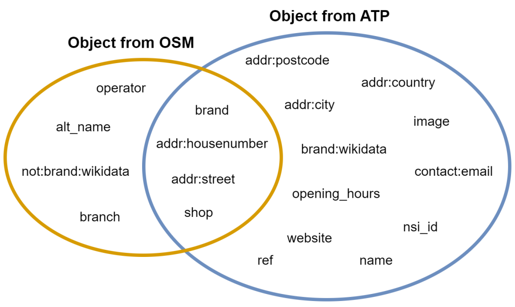


Figure 6.6.: Example of a small attribute intersection

An additional shortcoming for the creation of a profile concerns the supported logical operations by the OSM Conflator. The tags or attributes provided by the profile can only be connected by an AND operation. Especially in the situation where the attributes can vary a lot it would be helpful, if a UNION or OR operation is supported as well. In the above example, it would be possible to look for the tag name and if this tag does not exist, the tag operator could be checked. As either of these tags would trigger a match, this entry would not come up as a new point.

6.3.3. Poor OSM data Quality

Tag values have the potential for a variety of problems. OSM served several examples which can be traced back to single edits without a coordinated standard and without a naming concept for the brand at hand.

Capitalisation

The query of the Overpass API is case sensitive. However, it seems as the conventions regarding the capitalisation are not adhered. Sometimes all letters are capitalised, which should not be the case at all. Furthermore, capital letters for the first word only as well as for every first letter of all words were seen. Therefore, the tag values for the profile *Aldi* needs to be adjusted, so that *aldi* and *ALDI* will be checked as well. We found this inconvenience while profiling the OSM dataset as described in section 4.3.1. These capitalisation differences occur among the objects of the OSM dataset and can therefore not be corrected by the scraper. We implemented this as a simple convert to lower case letters before comparing. This allows the algorithm to increase its matching capability, but does not solve the situation of the capitalisation of the OSM object. To tackle this we suggested the overwrite of the OSM name by adding the *name* tag to the profiles *master_ tags*.

Language regions

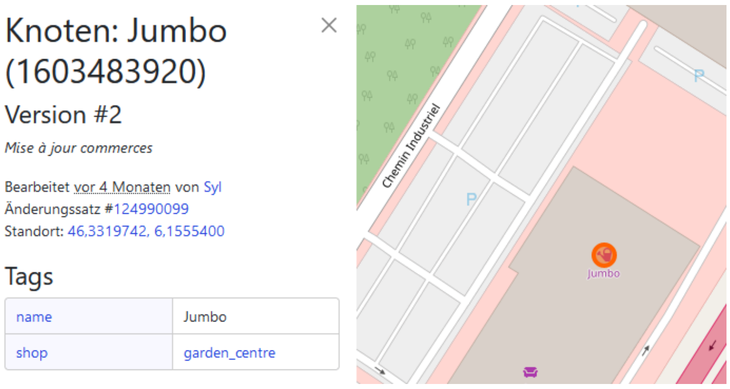
While we can check for a name tag with the value *Coop Bau+Hobby* in Zürich, this will not match any of their stores in Lausanne called *Coop Brico+Loisir*. For the OSM objects of some brands there are no consistent naming conventions and thus some values are quite unique at times. We were able to capture this with a specialised profile. For a more generic profile we believe it requires preceding steps to increase the data quality. After that a more generic profile can be used for further maintenance.

Real world circumstances

While profiling the *Jumbo* dataset another interesting problem occurred. The *Jumbo* brand was bought by *Coop* and they now operate their *Coop Bau+Hobby* under the name *Jumbo*. Thus requiring the profile to match objects, where in the worst case only the key-value pair [shop=doityourself] matches. We implemented this with a name comparison using wildcards. This increased the precision enormously while it slightly decreased the efficiency as expected in section 4.3.2.

Inconsistent tag values

Analysing the output of the OSM Conflator revealed that the tag values are in some cases inconsistent. It is the decision of an OSM member which tag values will be captured. However, it is not the case that for a brand the tag values for all its entities will be the same. There are differences as shown in 6.7.
The tag value for a shop of Jumbo is mainly *doityourself*. However, there exist exceptions. While *garden_centre* could be added to the query passed to OverpassAPI it would load hundreds of additional entries from OSM even though we are interested in only two, one of them shown in Figure 6.7.



Knoten: Jumbo (1603483920)

Version #2

Mise à jour commerces

Bearbeitet vor 4 Monaten von Syl
Änderungssatz #124990099
Standort: 46,3319742, 6,1555400

Tags

name	Jumbo
shop	garden_centre

Figure 6.7.: Example of a tag causing problems

6.3.4. Coordinates Too Far Off

In an ideal data set the coordinates and the *shop* or respectively *amenity* would be reliable and thus allow a match. A slight deviation of the coordinates could be overcome by enlarging the radius of the search field. This is needed at the moment since the coordinates published by the brands are of questionable accuracy. However, such an adjustment is not free. It increases the computation time and the potential of mismatches with other locations closer to the incorrect coordinates. A trade off has to be made and the basis for decision varies by source.
In figure 6.8 the coordinates given by the external source (bottom right) were 4km off from the actual site (top left).
To overcome this limitation, we can get the coordinates from the address as the address information is of higher accuracy in our tested dataset.

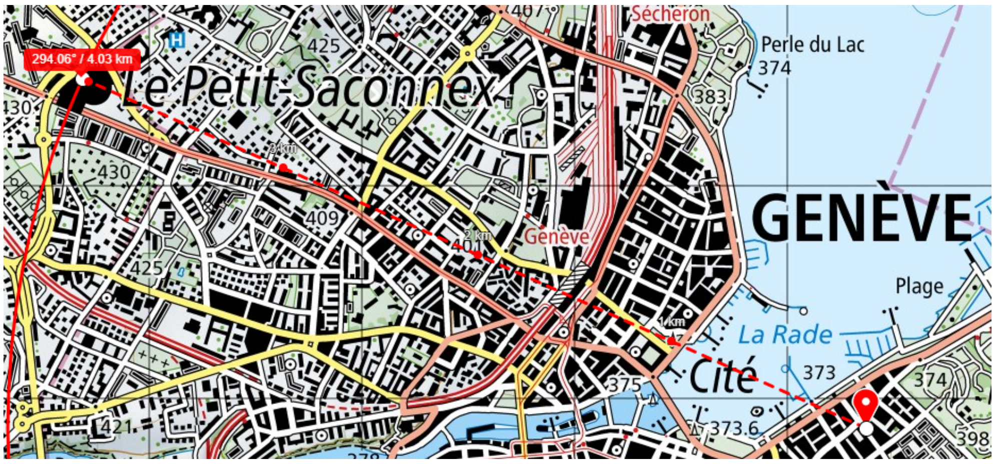


Figure 6.8.: Example of a coordinate mismatch of 4km

6.4. Requirements for the External Data

According to Table 6.1, there was no questionable data provided by ATP, except the unprecise coordinates, which were reported separately. However, during a high-level data profiling of the ATP sources, instances with questionable data quality were detected. To create a more generic profile it is helpful to define standards for the collected data from the web scrapers.

- Atomicity: A tag must hold an atomic value. The profile cannot take care of extracting concatenated information.
- Schema: The tags used must correspond to tags of OSM. Since the data is to be added to OSM it should be prepared for OSM from the beginning.
- Relevancy: Tags must be useful and not be outdated. Property values for example do not contribute.
- Data format: Format conversion must be done by the scraper to ensure a profile knows how to interpret the data.
- Localisation: Values which are dependent on language regions need to be adjusted by the scraper. A profile cannot necessarily detect whether translations have to be done.
- Validation: The profile will not automatically validate tag values on their completeness and correctness. This must be done by the scraper and its creator.

Part II.

Conclusion and Outlook

We Conclude

Using the OSM Conflator rather than Osmose appears to be the right decision. While editing only one brand at a time, the OSM editors work with the same schema and are more aware of the data quality than they would be while jumping around between various brands. This benefits the data quality since the editor can introduce solid standards throughout the whole brand in regard to naming and tag usage. It also allows to react to company takeovers which are hard to detect in an automated fashion. Also regional differences of brand names could be considered.

Furthermore, the profile creation was optimised by defining a template, which explains step by step, how a new profile has to be created. Although the optional feature of a more generalised profile was not fulfilled, we think our result already supports OSM maintenance. The template will assist OSM editors by the creation of new profiles and hence, contribute to improved data quality in OSM.

The current functionalities of the profiles and the OSM Conflator are not sufficient to meet the requirements of a more generic profile. A profile currently requires hard coded queries done manually by the creator. It is not possible to dynamically extract relevant tags from the external data source to build a suitable query. In addition, the OSM Conflator only supports the logic operation AND between tags, which is not ideal to create queries for different POIs or regions. However, the logic OR is allowed between tag values.

While selecting the external data for the creation of the five profiles, we came across some data errors residing in the data collected by the scrapers: Namely, a lack of atomicity and sometimes wrong assignments of tag values. Additionally, the accuracy of the provided location data of various brands was deficient. The coordinates are sometimes unprecise and even their own websites show maps with markers far off their actual store location. This is one of the reasons why we support OSM with their approach of manually reviewing changes. The quality of the coordinates provided by brandy is too low for an automated approach and would without doubt affect OSM negatively.

Our Outlook

We encourage to continue the reconciliation of OSM data and the generation of a more generic profile to optimise the maintenance process even further.

We also suggest an extension of the current profile code to allow a dynamic generation of the profile's Overpass API query string. The hard coded query string seems to limit the potential of the OSM Conflator. It would be much more performant and handy, to just download one file with a set of geographical data and the OSM Conflator would be able to identify the relevant tags and create the correct queries automatically. Alternatively, a profile creator will be introduced. The creator will be an interface between the external data and a profile. The creator will establish various profiles automatically based on the data provided by ATP. Afterwards, the OSM Conflator will obtain numerous profiles and execute precise queries for each profile. Besides the suggested enhancements for the profiles, also the code of the OSM Conflator needs an adjustment to support more logic operations for the query generator. This would enable more precise queries via the Overpass API to OSM.

Another aspect to be further analysed is the external data quality. For this work, only external data with an acceptable quality was considered. However, it was noticed that some scrapers provide data with poor quality, which makes the matching even harder. This issue can be tackled by a higher request for standardised tag creation on ATP's side. Deprecating tags like *addr:full* would force scrapers to a schema which supports the first normal form. This benefits downstream systems and might increase ATP's relevancy.

The topic of duplicates is almost neglected in this work. A first review indicates that the recognition of duplicates within the OSM Conflator needs to be adjusted, or more guidance for the creator of a profile should be available. This issue came up, when the profile for the Police of Zurich was briefly implemented. Within a police station, there could be different departments, which is relevant information for a person looking this up. However, due to the hard coded query of the profile, the OSM Conflator found only one match within the police station and the others were considered duplicates. The construction of the query does not allow the addition of a tag department with a tag value, as otherwise only police stations with this department would match. Consequently, more analysis of this aspect has to be done in subsequent studies.

It is unfortunate that the time for this work has already run out. Therefore, we hope to see this project grow as we experienced its potential during our study and are convinced it will have a positive impact on the data quality of OSM.