A Knowledge Base for Personal Information Management

David Montoya
Square Sense
david@montoya.one

Thomas Pellissier Tanon
LTCI, Télécom ParisTech
ttanon@enst.fr

Serge Abiteboul
Inria Paris
& DI ENS, CNRS, PSL Research University
serge.abiteboul@inria.fr

Pierre Senellart
DI ENS, CNRS, PSL Research University
& Inria Paris
& LTCI, Télécom ParisTech
pierre@senellart.com

Fabian M. Suchanek
LTCI, Télécom ParisTech
suchanek@enst.fr

ABSTRACT

Internet users have personal data spread over several devices and across several web systems. In this paper, we introduce a novel open-source framework for integrating the data of a user from different sources into a single knowledge base. Our framework integrates data of different kinds into a coherent whole, starting with email messages, calendar, contacts, and location history. We show how event periods in the user's location data can be detected and how they can be aligned with events from the calendar. This allows users to query their personal information within and across different dimensions, and to perform analytics over their emails, events, and locations. Our system models data using RDF, extending the `schema.org` vocabulary and providing a SPARQL interface.

1 INTRODUCTION

Internet users commonly have their personal data spread over several devices and services. This includes emails, messages, contact lists, calendars, location histories, and many other. However, commercial systems often function as *data traps*, where it is easy to check information in but difficult to query and exploit it. For example, a user may have all her emails stored with an email provider – but cannot find out which of her colleagues she interacts most frequently with. She may have all her location history on her phone – but cannot find out which of her friends' places she spends the most time at. Thus, a user often has paradoxically no means to make full use of data that she has created or provided. As more and more of our lives happen in the digital sphere, users are actually giving away part of their life to external data services.

We aim to put the user back in control of her own data. We introduce a novel framework that integrates and enriches personal information from different sources into a single knowledge base (KB) that lives on the user's machine, a machine she controls. Our system, Thymeflow, replicates data of different kinds from outside services and thus acts as a digital home for personal data. This provides the user with a high-level global view of that data, which she can use for querying and analysis. All of this integration and analysis happens locally on the user's computer, thus guaranteeing her privacy.

Designing such a personal KB is not easy: Data of completely different nature has to be modeled in a uniform manner, pulled into the knowledge base, and integrated with other data. For example, we have to find out that the same person appears with different email addresses in address books from different sources. Standard KB alignment algorithms do not perform well in our scenario, as we show in our experiments. Furthermore, integration spans data of different modalities: to create a coherent user experience, we need to align calendar events (temporal information) with the user's location history (spatiotemporal) and place names (spatial).

We provide a fully functional and open-source personal knowledge management system. A first contribution of our work is the management of location data. Such information is becoming commonly available through the use of mobile applications such as Google's Location History [20]. We believe that such data becomes useful only if it is semantically enriched with events and people in the user's personal space. We provide such an enrichment.

A second contribution is the adaptation of ontology alignment techniques to the context of personal KBs. The alignment of persons and organizations is rather standard. More novel are alignments based on time (a meeting in the calendar and a GPS location), or space (an address in contacts and a GPS location).

Our third contribution is an architecture that allows the integration of heterogeneous personal data sources into a coherent whole. This includes the design of incremental synchronization, where a change in a data source triggers the loading and treatment of just these changes in the central KB. Conversely, the user is able to perform updates on the KB, which are made persistent wherever possible in the sources. We also show how to integrate knowledge enrichment components into this process, such as entity resolution and spatio-temporal alignments.

As implemented, our system can provide answers to questions such as: Who have I contacted the most in the past month (requires alignments of different email addresses)? How many times did I go to Alice's place last year (requires alignment between contact list and location history)? Where did I have lunch with Alice last week (requires alignment between calendar and location history)?

Our system, Thymeflow, was previously demonstrated in [32]. It is based on an extensible framework available under an open-source software license[1]. People can therefore freely use it, and researchers can build on it.

We first introduce out data model and sources in Section 2, and then present the system architecture of Thymeflow in Section 3.

LDOW 2018, April 2018, Lyon, France

[1] https://github.com/thymeflow/thymeflow

Section 4 details our knowledge enrichment processes, and Section 5 our experimental results. Related work is described in Section 6. Before concluding in Section 8, we discuss lessons learnt while building and experimenting with Thymeflow in Section 7.

2 DATA MODEL

In this section, we briefly describe the schema of the knowledge base, and discuss the mapping of data sources to that schema.

Schema. We use the RDF standard [9] for knowledge representation. We use the namespace prefixes `schema` for http://schema.org/, and `rdf` and `rdfs` for the standard namespaces of RDF and RDF Schema, respectively. A *named graph* is a set of RDF triples associated with a URI (its name). A *knowledge base* (KB) is a set of named graphs.

For modeling personal information, we use the `schema.org` vocabulary when possible. This vocabulary is supported by Google, Microsoft, Yahoo, and Yandex, and documented online. Wherever this vocabulary is not fine-grained enough for our purposes, we complement it with our own vocabulary, that lives in the namespace http://thymeflow.com/personal# with prefix `personal`.

Figure 1 illustrates a part of our schema. Nodes represent classes, rounded colored ones are non-literal classes, and an edge with label p from X to Y means that the predicate p links instances of X to instances of type Y. We use locations, people, organizations, and events from `schema.org`, and complement them with more fine-grained types such as Stay, EmailAddress, and PhoneNumber. Person and Organization classes are aggregated into a `personal:Agent` class.

Emails and contacts. We treat emails in the RFC 822 format [8]. An email is represented as a resource of type `schema:Email` with properties such as `schema:sender`, `personal:primaryRecipient`, and `personal:copyRecipient`, which link to `personal:Agent` instances. Other properties are included for the subject, the sent and received dates, the body, the attachments, the threads, etc.

Email addresses are great sources of knowledge. An email address such as "jane.doe@inria.fr" provides the given and family names of a person, as well as her affiliation. However, some email addresses provide less knowledge and some almost none, e.g., "j4569@gmail.com". Sometimes, email fields contain a name, as in "Jane Doe <j4569@gmail.com>", which gives us a name triple. In our model, `personal:Agent` instances extracted from emails with the same combination of email address and name are considered indistinguishable (i.e., they are represented by the same URI). An email address does not necessarily belong to an individual; it can also belong to an organization, as in edbt-school-2013@imag.fr or fancy_pizza@gmail.com. This is why, for instance, the sender, in our data model, is a `personal:Agent`, and not a `schema:Person`.

A vCard contact [36] is represented as an instance of `personal:Agent` with properties such as `schema:familyName`, and `schema:address`. We normalize telephone numbers, based on a country setting provided by the user.

Calendar. The iCalendar format [11] can represent events. We model them as instances of `schema:Event`, with properties such as name, location, organizer, attendee, and date. The location is typically given as a postal address, and we will discuss later how to associate it to geo-coordinates and richer place semantics. The Facebook Graph API [15] also models events the user is attending or interested in, with richer location data and list of attendees (a list of names).

Location history. Smartphones are capable of tracking the user's location over time using different positioning technologies: satellite navigation, Wi-Fi, and cellular. Location history applications continuously run in the background, and store the user's location either locally or on a distant server. Each point in the user's location history is represented by time, longitude, latitude, and horizontal accuracy (the measurement's standard error). We use the Google Location History format, in JSON, as Google users can easily export their history in this format. A point is represented by a resource of type `personal:Location` with properties `schema:geo`, for geographic coordinates with accuracy, and `personal:time` for time.

3 SYSTEM ARCHITECTURE

A personal knowledge base could be seen as a *view* defined over personal information sources. The user would query this view in a mediation style [17] and the data would be loaded only on demand. However, accessing, analyzing and integrating these data sources on the fly would be expensive tasks. For this reason, Thymeflow uses a warehousing approach. Data is loaded from external sources into a persistent store and then enriched.

Thymeflow is a web application that the user installs, providing it with a list of data sources, together with credentials to access them (such as tokens or passwords). The system accesses the data sources and pulls in the data. All code runs locally on the user's machine. None of the data leaves the user's computer. Thus, the user remains in complete control of her data. The system uses adapters to access the sources, and to transform the data into RDF. We store the data in a persistent triple store, which the user can query using SPARQL.

One of the main challenges in the creation of a personal KB is the temporal factor: data sources may change, and these updates should be reflected in the KB. Changes can happen during the initial load time, while the system is asleep, or after some inferences have already been computed. To address these dynamics, Thymeflow uses software modules called *synchronizers* and *enrichers*. Figure 2 shows synchronizers on the left, and enrichers in the center. Synchronizers are responsible for accessing data sources, enrichers (see Section 4) for inferring new statements, such as alignments between entities obtained by entity resolution.

Modules are scheduled dynamically and may be triggered by updates in the data sources (e.g., calendar entries) or by new pieces of information derived in the KB (e.g., the alignment of a position in the location history with a calendar event). The modules may also be started regularly for particularly costly alignment processes. When a synchronizer detects a change in a source, a pipeline of enricher modules is triggered, as shown in Figure 2. Enrichers can also use knowledge from external data sources, such as Wikidata [44], Yago [41], or OpenStreetMap.

Synchronizer modules are responsible for retrieving new data from a data source. For each data source that has been updated, the adapter for that particular source transforms the source updates since last synchronization into a set of insertions/deletions in RDF.

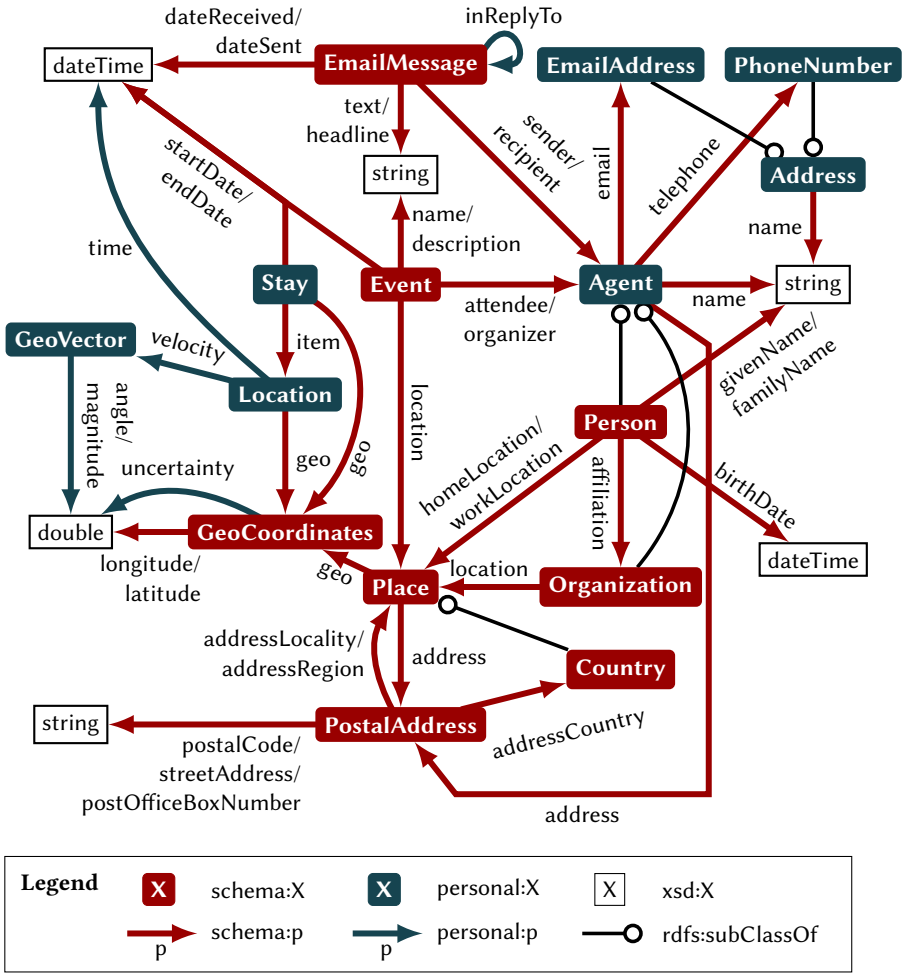

Figure 1: Personal data model

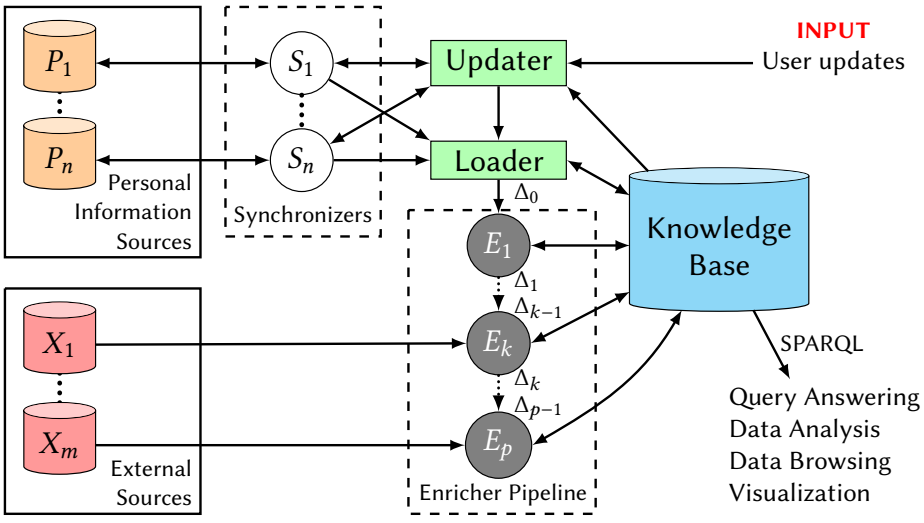

Figure 2: System architecture

This is of course relatively simple for data sources that track modifications, e.g., CalDAV (calendar), CardDAV (contacts) and IMAP (email). For others, this requires more processing. The result of this process is a delta update, i.e., a set of updates to the KB since the last time that particular source was considered.

The KB records the provenance of each newly obtained piece of information. Synchronizers record a description of the data source, and enrichers record their own name. We use named graphs to store the provenance. For example, the statements extracted from an email message in the user's email server will be contained in a graph named with the concatenation of the server's email folder URL and the message id. The graph's URI is itself an instance of `personal:Document`, and is related to its source via the `personal:documentOf` property. The source is an instance of `personal:Source` and is in this case the email server's URL. Account information is included in an instance of `personal:Account` via the `personal:sourceOf` property. Account instances allows us to gather different kinds of data sources, (e.g., CardDAV, CalDAV and IMAP servers) belonging to one provider (e.g., corporate IT services) to which the user accesses through one identification. This provenance can be used to answer queries such as "What meetings were recorded in my work calendar for next Monday?".

Finally, the system allows the propagation of information from the KB to the data sources. These can either be insertions/deletions derived by the enrichers, or insertions/deletions explicitly specified by the user. For instance, consider the information that different email addresses correspond to the same person. This information can be pushed to data sources, which may for example result in performing the merge of two contacts in the user's list of contacts. To propagate the information to the source, we translate from the structure and terminology of the KB back to that of the data source and use the API of that source. The user has the means of controlling this propagation, e.g., specifying whether contact information in our system should be synchronized to her phone's contact list.

The user can update directly the KB by inserting or deleting knowledge statements. Such updates to the KB are specified in the SPARQL Update language [18]. When no source is specified for recording this new information, the system considers all the sources that know the subject of the particular statement. For insertion, if no source is able to register a corresponding insertion, the system performs the insertion in a special locally persistent graph, called the *overwrite graph*. For deletions, if one source fails to perform a deletion (e.g., because the statement is read-only), the system removes the statement from the KB anyway (even if the data is still in some upstream source). A negative statement is added to the overwrite graph. This negative statement will prevent using a source statement to reintroduce the corresponding statement in KB: The negative statement overwrites the source statement.

4 ENRICHERS

We describe the general principles of enricher modules. We then describe two specific enrichments: agent matching and event geolocation.

After loading, enricher modules perform inference tasks such as entity resolution, event geolocation, and other knowledge enrichment tasks. An enricher works in a differential manner: it takes as input the current state of the KB, and a collection of changes Δ_i that have recently happened. It computes a new collection Δ_{i+1} of enrichments. Intuitively, this allows reacting to changes in a data source. When some Δ_0 is detected (typically by some synchronizer), the system runs a pipeline of enrichers to take these changes into consideration. For instance, when a new entry is entered in the calendar with an address, a geocoding enricher is called to locate it. Another enricher will later attempt to match it with a position in the location history. For performance, particularly costly enrichers wait until there are enough changes, or when no more changes are happening, before running on a batch of changes. This is the case for the entity resolution enricher. We now present this enricher and another one that has been incorporated into the system.

4.1 Agent Matching

Facets. The KB keeps information as close to the original data as possible. Thus, the knowledge base will typically contain several entities for the same person, if that person appears with different names or different email addresses. We call such resources *facets* of the same real-world agent. Different facets of the same agent will be linked by the `personal:sameAs` relation. The task of identifying equivalent facets has been intensively studied under different names such as record linkage, entity resolution, or object matching [5]. In our case, we use techniques that are tailored to the context of personal KBs: identifier-based matching and attribute-based matching.

Identifier-based matching. We can match two facets if they have the same value for some particular attribute (such as an email address or a telephone number), which, in some sense, identifies or determines the entity. This approach is commonly used in personal information systems (in research and industry) and gives fairly good results for linking, e.g., facets extracted from emails and the ones extracted from contacts. Such a matching may occasionally be incorrect, e.g., when two spouses share a mobile phone or two employees share the same customer relations email address. In our experience, such cases are rare, and we postpone their study to future work.

Two agent facets with the same first and family names have, for instance, a higher probability to represent the same agent than two agent facets with different names, all other attributes held constant. Besides names, attributes that can help determine a matching include `schema:birthDate`, `schema:gender`, and `schema:email`.

We tried holistic matching algorithms for graph alignments [40] that we adapted to our setting. The results turned out to be disappointing (see Section 5). We believe this is due to the following: (i) almost all agent facets have a `schema:email`, and possibly a `schema:name`, but most of them lack other attributes that are thus almost useless; (ii) names extracted from mails may contain pseudonyms, abbreviations, or lack family names, which reduces matching precision. (iii) we cannot reliably compute name frequency metrics from the knowledge base, since a rare name may appear many times for different email addresses if a person happens to be a friend of the user. Therefore, we developed our own algorithm, *AgentMatch*, which works as follows:

(1) We partition `Agents` using the equivalence relation computed by matching identifying attributes.

(2) For each `Agent` equivalence class, we compute its corresponding set of names, and, for each name, its number of occurrences (in email messages, etc.).
(3) We compute Inverse Document Frequency (IDF) scores, where the documents are the equivalence classes, and the terms are the name occurrences.
(4) For each pair of equivalence classes, we compute a numerical similarity between each pair of names using an approximate string distance that finds the best matching of words between the two names and then compares matching words using another string similarity function (discussed below). The similarity between two names is computed as a weighted mean using the sum of word-IDFs as weights. The best matching of words corresponds to a maximum weight matching in the bipartite graph of words where weights are computed using the second string similarity function. The similarity (in $[0, 1]$) between two equivalence classes is computed as a weighted mean of name pair similarity using the product of word occurrences as weights.
(5) Pairs for which the similarity is above a certain threshold are considered to correspond to two equivalent facets.

The second similarity function we use is based on the Levenshtein edit-distance, after string normalization (accent removal and lowercasing). In our experiments, we have also tried the Jaro–Winkler distance. For performance reasons, we use 2- or 3-gram-based indexing of words in agent names, and only consider in step (4.) of the process those `Agent` parts with some ratio S of q-grams in common in at least one word. For instance, two `Agent` parts with names "Susan Doe" and "Susane Smith" would be candidates.

4.2 Geolocating Events

We discuss how to geolocate events, e.g., how we can detect that Monday's lunch was at "Shana Thai Restaurant, 311 Moffett Boulevard, Mountain View, CA 94043". For this, we first analyze the location history from the user's smartphone to detect places where the user stayed for a prolonged period of time. We then perform some spatiotemporal alignment between such stays and the events in the user's calendar. Finally, we use geocoding to provide location semantics to the events, e.g., a restaurant name and a street address.

Detecting stays. Locations in the user's location history can be put into two categories: *stays* and *moves*. *Stays* are locations where the user remained for some period of time (e.g., dinner at a restaurant, gym training, office work), and *moves* are the others. *Moves* usually correspond to locations along a journey from one place to another, but might also correspond to richer outdoor activity (e.g., jogging, sightseeing). Figure 3 illustrates two *stay* clusters located inside the same building.

To transform the user's location history into a sequence of stays and moves, we perform time-based spatial clustering [29]. The idea is to create clusters along the time-axis. Locations are sorted by increasing time, and each new location is either added to an existing cluster (that is geographically close and that is not too old), or added to a new cluster. To do so, a location is spatially represented as a two dimensional unimodal normal distribution $\mathcal{N}(\mu, \sigma^2)$. The assumption of a normally distributed error is typical in the field of processing location data. For instance, a cluster of size 1 formed

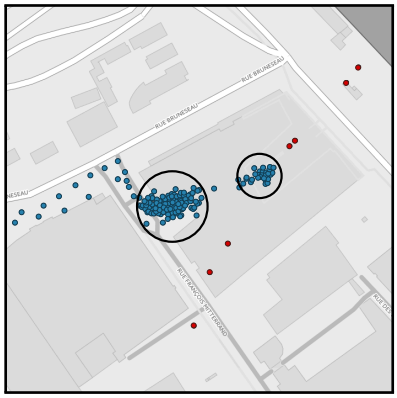

Figure 3: Two clusters of *stays* (blue points inside black circles) within the same building. Red points are outliers. The other points are *moves*.

by location point $p = (t, x, y, a)$, where t is the time, a the accuracy, and (x, y) the coordinates, is represented by the distribution $P = \mathcal{N}(\mu_P = (x, y), \sigma_P^2 = a^2)$. When checking whether location p can be added to an existing cluster C represented by distribution Q, the process computes the Hellinger distance [34] between the distribution P and the normal distribution $Q = \mathcal{N}(\mu_Q, \sigma_Q^2)$: $H^2(P, Q) = 1 - \sqrt{\frac{2\sigma_P\sigma_Q}{\sigma_Q^2+\sigma_Q^2}} \exp\left(-\frac{1}{4}\frac{d(\mu_P,\mu_Q)^2}{\sigma_P^2+\sigma_Q^2}\right) \in [0, 1]$, where $d(\mu_P, \mu_Q)$ is the geographical distance between cluster centers. The Hellinger distance takes into account both the accuracy and geographical distance between cluster centers, which allows us to handle outliers no matter the location accuracy. The location is added to C if this distance is below a certain threshold λ, i.e., $H^2(P, Q) \leqslant \lambda^2 < 1$. In our system, we used a threshold of 0.95.

When p is added to cluster C, the resulting cluster is defined with a normal distribution whose expectation is the arithmetic mean of location point centers weighted by the inverse accuracy squared, and whose variance is the harmonic mean of accuracies squared. Formally, if a cluster C is formed by locations $\{p_1, \ldots, p_n\}$, where $p_i = (t_i, x_i, y_i, a_i)$, then C is defined with distribution $\mathcal{N}(\mu, \sigma^2)$ where μ is the weighted arithmetic mean of location centers (x_i, y_i) weighted by their inverse accuracy squared a_i^{-2}, and the variance σ^2 is the harmonic mean of location accuracies squared a_i^{-2}.

$$\mu = \sum_{i=1}^{n} \frac{(x_i, y_i)}{a_i^2} \left(\sum_{i=1}^{n} \frac{1}{a_i^2}\right)^{-1} \qquad \sigma^2 = \left(\sum_{i=1}^{n} \frac{1}{a_i^2}\right)^{-1}$$

The coordinates are assumed to have been projected to an Euclidean plane locally approximating distances and angles on Earth around cluster points. If $n = 1$, then $\mu = (x_1, y_i)$ and $\sigma^2 = a_1^2$, which corresponds to the definition of a cluster of size 1.

A cluster that lasted more than a certain threshold is a candidate for being a stay. A difficulty is that a single location history (e.g., Google Location History) may record locations of different devices, e.g., a telephone and a tablet. The identity of the device may not be recorded. The algorithm understands that two far-away locations, very close in time, must come from different devices. Typically, one of the devices is considered to be stationary, and we try to detect a movement of the other. Another difficulty comes when traveling in

high speed trains with poor network connectivity. Location trackers will often give the same location for a few minutes, which leads to the detection of an incorrect stay.

Matching stays with events. After the extraction of stays using the previous algorithm, the next step is to match these with calendar events. Such a matching turns out to be difficult because: (i) the location of an event (address or geo-coordinates) is often missing; (ii) when present, an address often does not identify a geographical entity, as in "John's home" or "room C110"; (iii) in our experience, starting times are generally reasonable (although a person may be late or early for a meeting) but durations are often not meaningful (around 70% of events in our test datasets were scheduled for 1 hour; among the 1-hour events that we aligned, only 9% lasted between 45 and 75 minutes); (iv) some stays are incorrect.

Because of (i) and (ii), we do not rely much on the location explicitly listed in the user's calendars. We match a stay with an event primarily based on time: the time overlap (or proximity) and the duration. In particular, we match the stay and the event, if the ratio of the overlap duration over the entire stay duration is greater than a threshold θ. As we have seen, event durations are often unreliable because of (iii). Our method still yields reasonable results, because it tolerates errors on the start of the stay for long stays (because of their duration) and for short ones (because calendar events are scheduled usually for at least one hour). If the event has geographical coordinates, we filter out stays that are too far away from that location (i.e., when the distance is greater than δ). We discuss the choice of θ and δ for this process in Section 5.

Geocoding event addresses. Once stays associated with events, we enrich events with rich place semantics (country, street name, postal code, place name). If an event has an explicit address, we use a *geocoder*. Thymeflow allows using different geocoders, e.g., the Google Maps Geocoding API [21], which returns the geographic coordinates of an address, along with structured place and address data. The enricher only keeps the geocoder's most relevant result and adds its data (geographic coordinates, identifier, street address, etc.) to the location in the knowledge base. For events that do not have an explicit address but that have been matched to a stay, we use the geocoder to transform the geographic coordinates of the stay into a list of nearby places. The most precise result is added as the event location. If the event has both an explicit address and a match with a stay, we call the geocoder on this address, while restricting the search to a small area around the stay coordinates.

5 EXPERIMENTS

In this section, we present the results of our experiments. We used datasets from two real users, whom we call Angela and Barack. Angela's dataset consists of 7,336 emails, 522 calendar events, 204,870 location points, and 124 contacts extracted from Google's email, contact, calendar, and location history services. This corresponds to 1.6M triples in our schema. Barack's dataset consists of 136,301 emails, 3,080 calendar events, 1,229,245 location points, and 582 contacts extracted from the same sources. Barack's emails cover a period 5,540 days, locations cover 1,676 days. This corresponds to 10.3M triples, where 70.9 % come from the location history, 28.8 % from emails, 0.3 % from calendars and less than 0.1 % from contacts.

We measured the loading times of Angela's dataset into the system in two different scenarios: from source data on the Internet (using Google API, except for the location history which is not provided by the API and was loaded from a file), and from source data stored in local files. Loading took 19 and 4 minutes, respectively, on a desktop computer (Intel i7-2600k 4-core, 3.4 GHz, 20 GB RAM, SSD).

5.1 Agent Matching

We evaluated the precision and recall of the AgentMatch algorithm (Section 4) on Barack's dataset. This dataset contains 40,483 `Agent` instances with a total of 25,381 `schema:name` values, of which 17,706 are distinct; it also contains 40,455 `schema:email` values, of which 24,650 are distinct. To compute the precision and recall, we sampled 2,000 pairs of distinct `Agents`, and asked Barack to manually assign to each possible pair a ground truth value (true/false). Barack was provided with the email address and name of each agent, and was allowed to query the KB to get extra information.

We tested both Levenshtein and Jaro–Winkler as secondary string distance, with and without IDF term weights. The term q-gram match ratio (S) was set to 0.6. We varied λ so as to maximize the F1 value. Precision decreases while recall increases for decreasing threshold values. Our baseline is IdMatch, which matches two contacts iff they have the same email address.

As competitor, we considered PARIS [40], an ontology alignment algorithm that is parametrized by a single threshold. We used string similarity for email addresses, and the name similarity metric used by AgentMatch, except that it is applied to single `Agent` instances. PARIS computes the average number of outgoing edges for each relation. Since our dataset contains duplicates, we gave PARIS an advantage by computing these values upfront.

We also considered Google's "Find duplicates" feature. Google was not able to handle more than 27,000 contacts at the same time, and so we had to run it multiple times in batches. Since the final output depends on the order in which contacts were loaded, we present two results, one for which the contacts were supplied sorted by email address (Google1), and another for a random order (Google2). Since Google's algorithm failed to merge contacts that IdMatch did merge, we also tested running IdMatch on Google's output (GoogleId) for both runs. We also tested Mac OS X contact de-duplication feature. However, its result did not contain all the meta data from the original contacts, so that we could not evaluate this feature.

The results are shown in Table 1. As expected, our baseline IdMatch has a perfect precision, but a low recall (43%). Google, likewise, gives preference to precision, but achieves a higher recall than the baseline (50%). The recall improves further if the Google is combined with IdMatch (61%). PARIS, in contrast, favors recall (92%) over precision (83%), and achieves a better F1 value overall. The highest F1-measure (95%) is reached for AgentMatch with the Jaro–Winkler distance for a threshold of 0.825. It has a precision comparable to Google's, and a recall comparable to PARIS's.

5.2 Detecting Stays

We evaluated the extraction of stays from the location history on Barack's dataset. We randomly chose 15 days, and presented him

Table 1: Precision and recall of the Agent Matching task on Barack's dataset, for different parameters of the AgentMatch, IdMatch, PARIS and Google algorithms

Algorithm	Similarity	IDF	λ	Prec.	Rec.	F1
IdMatch				1.000	0.430	0.601
Google1				0.995	0.508	0.672
Google2				0.996	0.453	0.623
GoogleId2				0.997	0.625	0.768
GoogleId1				0.996	0.608	0.755
PARIS	Jaro–Winkler	T	0.425	0.829	0.922	0.873
AgentMatch	Levenshtein	F	0.725	0.945	0.904	0.924
AgentMatch	Levenshtein	T	0.775	0.948	0.900	0.923
AgentMatch	Jaro–Winkler	F	0.925	0.988	0.841	0.909
AgentMatch	Jaro–Winkler	T	0.825	0.954	0.945	0.949

Table 2: Stay extraction evaluation on Barack's dataset

Method	θ	#D	DΘ	Prec.	Recall	F1
Thyme	15	33.3 %	0.6 %	91.3 %	98.4 %	94.7 %
Thyme	10	46.0 %	0.9 %	82.9 %	98.4 %	90.0 %
Thyme	5	62.5 %	1.0 %	62.1 %	100.0 %	76.6 %
Google		17.5 %	N/A	87.7 %	89.1 %	88.4 %

a web interface with (1) the raw locations on a map, (2) the stays detected by Thymeflow, and (3) the stays detected by his Google Timeline [20]. Barack was then asked to annotate each stay as "true", i.e., corresponding to an *actual* stay, or "false", and to report missing stays, based on his memories. He was also allowed to use any other available knowledge (e.g., his calendar). The exact definition of an *actual stay* was left to Barack, and the reliability of his annotations were dependent on his recollection. In total, Barack found 64 *actual stays*. Sometimes, an algorithm would detect an actual stay as multiple consecutive stays. In that case, we counted one true stay and counted the number of duplicates and the resulting move duration in-between. For instance, an *actual stay* of 2 hours output as two stays of 29 min and 88 min, with a short move of 3 minutes in-between would count as 1 true stay, 1 duplicate and 3 minutes of duplicate move duration.

Table 2 shows the resulting precision and recall for each method, with varying stay duration thresholds for Thymeflow. We also show the duplicate ratio #D (number of duplicates over the number of true stays) and the move duration ratio DΘ (duplicate move duration over the total duration of true stays). Overall, Thymeflow obtains results comparable to Google Timeline for stay extraction, with better precision and recall, but more duplicates.

Matching stays with events. We also evaluated the matching of stays in the user's location history with the events in their calendar. We sampled stays from Angela and Barack's datasets, and produced all possible matchings to events, i.e., all matchings produced by the algorithm whatever the threshold. Angela and Barack were then asked to manually label the matches as correct or incorrect. The matching process relies on two parameters, namely the duration ratio threshold θ and the filtering distance δ. We varied θ and found that a value of 0.2 leads to best F1 values. With this value, we varied δ, and found that the performance improves consistently with larger values. This indicates that filtering stays which are too far from event location coordinates (where available) should not be taken into consideration. With these settings, the matching performs quite well: We achieve a precision and recall of around 70%.

Table 3: Geocoding task on matched (event, stay) pairs in Barack's dataset (in %)

Method	M	F	T	P_T	T|A	$P_{T|A}$	F1
GoogleTimeline	0	82.8	14.8	14.8	17.2	17.2	29.4
EventSingle	50	40.8	4.0	7.9	9.6	19.0	27.7
Event	26	63.6	4.0	5.4	10.0	13.6	22.9
Stay	0	69.6	0.8	0.8	30.4	30.4	46.6
StayEvent	50	16.4	27.2	54.4	33.6	67.2	57.3
StayEvent|Stay	0	50.0	28.4	28.4	50.0	50.0	66.7

5.3 Geocoding

We evaluated different geocoding enrichers for (event, stay) pairs described in Section 4. We used the Google Maps Geocoding API. We considered three different inputs to this API: the event location address attribute (Event), the stay coordinates (Stay), and the event location attribute with the stay coordinates given as a bias (StayEvent). We also considered a more precise version of Event, which produces a result only if the geocoder returns a single unambiguous location (EventSingle). Finally, we devised the method StayEvent+Stay, which returns the StayEvent result if it exists, and the Stay result otherwise.

For each input, the geocoder gave either no result (M), a false result (F), a true place (T), or just a true address (A). For instance, an event occurring in "Hôtel Ritz Paris" is true if the output is for instance "Ritz Paris", while an output of "15 Place Vendôme, Paris" would count as a true address. For comparison, we also evaluated the place given by Google Timeline [20].

Due to limitations of the API, geocoding from stay coordinates mostly yielded address results (99.2% of the time). To better evaluate these methods, we computed the number of times in which the output was either a true place, or a true address (denoted T|A). For those methods that did not always return a result, we computed a precision metric $P_{T|A}$ (resp., P_T), that is equal to the ratio of T|A (resp., T) to the number of times a result was returned. We computed a F1-measure based on the $P_{T|A}$ precision, and a recall assimilated to the number of times the geocoder returned a result ($1 - M$).

The evaluation was performed on 250 randomly picked (stay, event) pairs in Barack's dataset. The results are shown in Table 3. The Google Timeline gave the right place or address only 17.2% of the time. The EventSingle method, likewise, performs poorly, indicating that the places are indeed highly ambiguous. The best precision ($P_{T|A}$ of 67.2%) is obtained by Geocoding with the StayEvent, but this method returns a result only 50.0% of the time. StayEvent+Stay, in contrast, can find the right place 28.4% of the time, and the right place or address 50.0% of the time, which is our best result. We are happy with this performance, considering that around 45% of the event locations were room numbers without mention of a building or place name (i.e., C101).

5.4 Use Cases

The user can query the KB using SPARQL [25]. Additionally, Thymeflow uses a modern triple store supporting full-text and geospatial queries. Since the KB unites different data sources, queries can seamlessly span multiple sources and data types. This allows the user (Angela), to ask for instance:

- What are the phone numbers of her birthday party guests? (combining information from the contacts and the emails)
- What places did she visit during her last trip to London? (combining geocoding information with stays)
- For each person she meets more than 3 times a week, what are the top 2 places where she usually meets that particular person? (based on her calendar and location history)

Such queries are not supported by current proprietary cloud services, which do not allow arbitrary queries.

Hub of personal knowledge. Finally, the user can use the bidirectionality of synchronization to enrich her existing services. For instance, she can enrich her personal address book (CardDav) with knowledge inferred by the system (e.g., a friend's birth date extracted from Facebook) using a SPARQL/Update query.

6 RELATED WORK

We now review the related work, on personal information management, on information integration, and on the specific tasks of location analysis and calendar matching.

6.1 Personal Information Management

This work is motivated by concept of personal information management (PIM), taking the viewpoint of [1] as to what a PIM system should be. [28] groups PIM research and development into the following problems: finding and re-finding, keeping and organizing personal information. We now present some notable contributions.

Finding and Re-finding. PIM has been concerned with improving how individuals go about retrieving a piece of information to meet a particular need.

For searching within the user's personal computer, desktop full-text search tools capable have been developed for various operating systems and platforms [46]. Search entries are for instance the files and folders on the file-system, email messages, browser history pages, calendar events, contacts, and applications. Search may be performed on both the content and the metadata. In particular, the IRIS [4] and NEPOMUK [24] projects used knowledge representation technologies to provide semantic search facilities and go beyond search to provide facilities for exchanging data between different applications within a single desktop computer.

Other research efforts have focused on ameliorating the process of finding things the user has already seen, using whatever context or meta-information that the user remembers [14, 39, 42].

Keeping. PIM has also addressed the question: What kind of information should be captured and stored in digital form?

A central idea of [3]'s vision is creating a device that is able to digitally capture all of the experiences and acquired knowledge of the user, so that it can act as a supplement to her memory. *Lifelogging* attempts to fulfil this vision by visually capturing the world that we see in our everyday lives [23]. MyLifeBits is a notable documented example [19]. The advent of cheaper, more advanced and efficient wearable devices able to capture the different aspects of one's life has made lifelogging indiscriminate of what it logs. Lifelogging activities include recording a history of machine enabled tasks (e.g., communications, editing, web browser's history), passively capturing what we see and hear (e.g., via a wearable camera), monitoring personal biometrics (e.g., steps taken, sleep quality), and logging mobile device and environmental context (e.g., the user's location, smart home sensing).

Different from lifelogging, which does not focus on the analysis of the logged information, the *quantified self* is a movement to incorporate data acquisition technology on certain focused aspects of one's daily life [23]. The quantified self focuses on logging experiences with a clearer understanding of the goals, such as exercise levels for fitness and health care.

Organizing. PIM also deals with the management and organization of information. It is for instance concerned with the management of privacy, security, distribution, and enrichment of information.

A *personal data service* (PDS) lets the user store, manage and deploy her information in a structured way. The PDS may be used to manage different identities and/or as central point of information exchange between services. For instance, an application that recommends new tracks based on what the user likes to listen may need to use adapters and authenticate with different services keeping a listening history. Instead, the PDS centralizes this information and the application only needs an adapter to connect to this PDS. Higgins [43], OpenPDS [10] and the Hub of All Things [26] are examples of PDSs.

MyData [37] describes a consent management framework that lets user control the flow of data between a service that has information about her and a service that uses this information. In this framework, which is still at its early stage of development, a central system holds credentials to access the different services on the user's behalf. The user specifies the rules by which flows of information between any two of those services are authorized. The central system is in charge of providing or revoking the necessary authorizations on each of those services to implement these rules. Contrary to a PDS, the actual data does not need to flow through the central system. Two services may spontaneously share information about the user with each other if legally entitled (e.g., two public bodies), in which case the central system is notified. It is an all-or-nothing approach that represents a paradigm shift from currently implemented ad-hoc flows of personal information across organizations.

Organizing information as a time-ordered stream of documents (a *lifestream*) has been proposed as a simple scheme for reducing the time the user spends in manually organizing documents into a classic hierarchical file system [16]. It has the advantage of providing unified view of the user's personal information. Lifestreams can be seen as a natural representation of lifelog information. The Digital Me system uses this kind of representation to unify data from different loggers [38].

For managing personal information, different levels of organization and abstraction have been proposed. *Personal data lakes* [45]

and *personal data spaces* [12] offer little integration and focus on handling storage, metadata, and search. On the other end, *personal knowledge bases*, which include more semantics, have been used: Haystack [30], SEMEX [13], IRIS [4], and NEPOMUK [24]. Such a structure allows the flexible representation of things like "this file, authored by this person, was presented at this meeting about this project". They integrate several sources of information, including documents, media, email messages, contacts, calendars, chats, and web browser's history. However, these projects date from 2007 and before and assume that most of the user's personal information is stored on her personal computer. Today, most of it is spread across several devices [1].

Some proprietary service providers, such as Google and Apple, have arguably come quite close to our vision of a personal knowledge base. They integrate calendars, emails, and address books, and allow smart exchanges between them. Some of them even provide *intelligent personal assistants* that proactively interact with the user. However, these are closed source proprietary solutions that promote vendor lock-in. In response, open-source alternative solutions have been developed, to cloud storage in particular, such as ownCloud [35] and Cozy [7]. These have evolved into application platforms that host various kinds of other user-oriented services (e.g., email, calendar, contacts). They leverage multi-device synchronization facilities and standard protocols to facilitate integration with existing contact managers and calendars. Cozy is notable for providing adapters for importing data from different kinds of services (e.g., activity trackers, finance, social) into the system into a document-oriented database. These tools bring the convenience of modern software-as-a-service solutions while letting the user be in control, not give away some of her privacy and free herself from vendor lock-in.

6.2 Information Integration

Data matching (also known as record or data linkage, entity resolution, object/field matching) is the task of finding records that refer to the same entity across different sources. It is extensively utilized in data mining projects and in large-scale information systems by business, public bodies and governments [5]. Example application areas include national census, the health sector, or fraud detection. In the context of personal information, SEMEX [13] integrates entity resolution facilities. SEMEX imports information from documents, bibliography, contacts and email, and uses attributes as well associations found between persons, institutions and conferences to reconcile references. However, different from our work, the integration is done at import time so the user cannot later manually revoke it through an update, and incremental synchronization is not handled. Recently, contact managers from known service providers have started providing de-duplication tools for finding duplicate contacts and merging them in bulk. However, these tools are often restricted to contacts present in the user's address book and do not merge contacts from social networks or emails.

Common standards, such as vCards and iCalendar, have advanced the state of the art by allowing provider-independent administration of personal information. There is also a proposed standard for mapping vCard content and iCalendars into RDF [6, 27]. While such standards are useful in our context, they do not provide the means to match calenders, emails, and events, as we do. The only set of vocabularies besides `schema.org` which provides a broad coverage of all entities we are dealing with is the OSCAF ontologies [33]. But their development was stopped in 2013 and they are not maintained anymore, contrary to schema.org which is actively supported by companies like Google and widely used on the web [22]. Recently, a personal data service has been proposed that reuses the OSCAF ontologies, but they use a relational database instead of a knowledge base [38].

6.3 Location Analysis and Calendar Matching

The ubiquity of networked mobile devices able to track users' locations over time has been greatly utilized for estimating traffic and studying mobility patterns in urban areas. Improvements in accuracy and battery efficiency of mobile location technologies have made possible the estimation of user activities and visited places on a daily basis [2, 20, 29]. Most of these works have mainly exploited sensor data (accelerometer, location, network) and readily available geographic data. Few of them, however, have exploited the user's calendar and other available data for creating richer and more semantic activity histories. Recently, a study has recognized the importance of using the location history and social network information for improving the representation of information contained in the user's calendar: e.g. for distinguishing genuine real-world events from reminders [31].

7 DISCUSSION

The RDF model seems well-suited for building a personal knowledge base. However powerful, making full use of it relies on being able to write and run performant queries. At this point, we cannot focus on optimizing for a specific application. It is not yet clear up to what extent Thymeflow should hold raw data (such as the entire location history), which, depending on how it is used, may be loaded on demand, in mediation style. Additionally, we would like to raise the following issues that could drive future research and development:

- Gathering data for experiments: The research community might benefit from building and maintaining sets of annotated multi-dimensional personal information for use in different kinds of tasks. This is challenging, specially due to privacy concerns.
- Opportunities: Internet companies that already hold a lot of user data are not yet integrating everything they have in a coherent whole, and are not performing as well as we think they could. For instance, Google Location History does not integrate the user's calendar, unlike we do. We think that there are still many opportunities to create new products and functionalities from existing data alone.
- Inaccessible information: The hard truth is that many popular Internet-based services still do not provide an API for conveniently retrieving user data out of them, or that such an API is not feature-complete (e.g., Facebook, WhatsApp).

8 CONCLUSION

The Thymeflow system integrates data from emails, calendars, address books, and location history, providing novel functionalities

on top of them. It can merge different facets of the same agent, determine prolonged stays in the location history, and align them with events in the calendar.

Our work is a unique attempt at building a personal knowledge base. First, the system is complementary to and does not pretend to replace the existing user experience, applications, and functionalities, e.g., for reading/writing emails, managing a calendar, organizing files. Second, we embrace personal information as being fundamentally distributed and heterogeneous and focus on the need of providing knowledge integration on top for creating completely new services (query answering, analytics). Finally, while the system could benefit from more advanced analysis such the extraction of entities from rich text (e.g., emails), for linking them with elements of the KB, our first focus is on enriching existing semi-structured data, which improves the quality of data for use by other services.

Our system can be extended in a number of directions, including incorporating more data sources, extracting semantics from text, complex analysis of users' data and behavior. Future applications include personal analytics, cross-vendor search, intelligent event planning, recommendation, and prediction. Also, our system could use simpler query language, perhaps natural language, or even proactively interact with the user, in the style of Apple's Siri, Google's Google Now, Microsoft's Cortana, or Amazon Echo.

While the data obtained by Thymeflow remains under the user's direct control, fully respecting her privacy, the data residing outside of it may not. However, using Thymeflow, the user could have a better understanding of what other systems know about her, which is important first step in gaining control about it.

REFERENCES

[1] Serge Abiteboul, Benjamin André, and Daniel Kaplan. 2015. Managing your digital life. *CACM* 58, 5 (2015).

[2] Daniel Ashbrook and Thad Starner. 2003. Using GPS to learn significant locations and predict movement across multiple users. *Personal and Ubiquitous Computing* 7, 5 (2003).

[3] Vannevar Bush. 1945. As We May Think. *The Atlantic* (1945).

[4] Adam Cheyer, Jack Park, and Richard Giuli. 2005. *IRIS: Integrate. Relate. Infer. Share.* Technical Report. DTIC Document.

[5] Peter Christen. 2012. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection.* Springer.

[6] Dan Connolly and Libby Miller. 2005. *RDF Calendar - an application of the Resource Description Framework to iCalendar Data.* http://www.w3.org/TR/rdfcal/.

[7] Cozy Cloud. 2016. Cozy – Simple, versatile, yours. (2016). https://cozy.io/

[8] David H. Crocker. 1982. *Standard for the format of ARPA Internet text messages.* RFC 822. IETF. https://tools.ietf.org/html/rfc822

[9] Richard Cyganiak, David Wood, and Markus Lanthaler. 2014. *RDF 1.1 Concepts and Abstract Syntax.* http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[10] Yves-Alexandre de Montjoye, Erez Shmueli, Samuel S Wang, and Alex Sandy Pentland. 2014. openpds: Protecting the privacy of metadata through safeanswers. *PloS one* 9, 7 (2014), e98790.

[11] B. Desruisseaux. 2009. *Internet Calendaring and Scheduling Core Object Specification (iCalendar).* RFC 5545. IETF. https://tools.ietf.org/html/rfc5545

[12] Jens-Peter Dittrich and Marcos Antonio Vaz Salles. 2006. iDM: A Unified and Versatile Data Model for Personal Dataspace Management. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006.* 367–378. http://dl.acm.org/citation.cfm?id=1164160

[13] Xin Dong and Alon Y. Halevy. 2005. A Platform for Personal Information Management and Integration. In *CIDR.* 119–130. http://www.cidrdb.org/cidr2005/papers/P10.pdf

[14] Susan T. Dumais, Edward Cutrell, Jonathan J. Cadiz, Gavin Jancke, Raman Sarin, and Daniel C. Robbins. 2003. Stuff I've seen: a system for personal information retrieval and re-use. In *SIGIR 2003: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, July 28 - August 1, 2003, Toronto, Canada.* 72–79. DOI: https://doi.org/10.1145/860435.860451

[15] Facebook. 2016. The Graph API. (2016). https://developers.facebook.com/docs/graph-api/

[16] Eric Freeman and David Gelernter. 1996. Lifestreams: A storage model for personal data. *ACM SIGMOD Record* 25, 1 (1996), 80–86.

[17] Hector Garcia-Molina, Yannis Papakonstantinou, Dallan Quass, Anand Rajaraman, Yehoshua Sagiv, Jeffrey Ullman, Vasilis Vassalos, and Jennifer Widom. 1997. The TSIMMIS approach to mediation: Data models and languages. *Journal of intelligent information systems* 8, 2 (1997).

[18] Paula Gearon, Alexandre Passant, and Axel Polleres. 2013. *SPARQL 1.1 Update.* https://www.w3.org/TR/sparql11-update/.

[19] Jim Gemmell, Gordon Bell, and Roger Lueder. 2006. MyLifeBits: a personal database for everything. *Commun. ACM* 49, 1 (2006), 88–95.

[20] Google. 2016. Google Maps Timeline. (2016). https://www.google.fr/maps/timeline

[21] Google. 2017. Google Maps APIs. (2017). https://developers.google.com/maps/documentation/

[22] RV Guha, Dan Brickley, and Steve Macbeth. 2016. Schema.org: Evolution of structured data on the web. *CACM* 59, 2 (2016).

[23] Cathal Gurrin, Alan F Smeaton, and Aiden R Doherty. 2014. Lifelogging: Personal big data. *Foundations and trends in information retrieval* 8, 1 (2014), 1–125.

[24] Siegfried Handschuh, Knud Möller, and Tudor Groza. 2007. The NEPOMUK project-on the way to the social semantic desktop. In *I-SEMANTICS.*

[25] Steve Harris, Andy Seaborne, and Eric Prud'hommeaux. 2013. *SPARQL 1.1 Query Language.* http://www.w3.org/TR/sparql11-query/.

[26] HATDeX Ltd. 2017. Hub of All Things. (2017). https://hubofallthings.com

[27] Renato Iannella and James McKinney. 2014. *vCard Ontology - for describing People and Organizations.* http://www.w3.org/TR/2014/NOTE-vcard-rdf-20140522/.

[28] William Jones and Jaime Teevan. 2011. *Personal Information Management.* University of Washington Press, Seattle, WA U.S.A.

[29] Jong Hee Kang, William Welbourne, Benjamin Stewart, and Gaetano Borriello. 2004. Extracting Places from Traces of Locations. In *WMASH.*

[30] David R Karger, Karun Bakshi, David Huynh, Dennis Quan, and Vineet Sinha. 2005. Haystack: A customizable general-purpose information management tool for end users of semistructured data. In *Proc. of the CIDR Conf.*

[31] Tom Lovett, Eamonn O'Neill, James Irwin, and David Pollington. 2010. The calendar as a sensor: analysis and improvement using data fusion with social networks and location. In *UbiComp.*

[32] David Montoya, Thomas Pellissier Tanon, Serge Abiteboul, and Fabian Suchanek. 2016. Thymeflow, A Personal Knowledge Base with Spatio-temporal Data. In *CIKM.* Demonstration paper.

[33] Nepomuk Consortium and OSCAF. 2007. OSCAF Ontologies. (2007). http://oscaf.sourceforge.net/

[34] Mikhail S Nikulin. 2001. Hellinger distance. *Encyclopedia of Mathematics* (2001).

[35] ownCloud. 2016. ownCloud – A safe home for all your data. (2016). https://owncloud.org/

[36] S. Perreault. 2011. *vCard Format Specification.* RFC 6350. IETF. https://tools.ietf.org/html/rfc6350

[37] A Poikola, K Kuikkanieni, and H Honko. 2014. MyData – A Nordic Model for human-centered personal data management and processing. (2014). https://www.lvm.fi/documents/20181/859937/MyData-nordic-model/2e9b4eb0-68d7-463b-9460-821493449a63?version=1.0

[38] Mats Sjöberg, Hung-Han Chen, Patrik Floréen, Markus Koskela, Kai Kuikkaniemi, Tuukka Lehtiniemi, and Jaakko Peltonen. 2016. Digital Me: Controlling and Making Sense of My Digital Footprint. (2016). http://reknow.fi/dime/

[39] Craig AN Soules and Gregory R Ganger. 2005. Connections: using context to enhance file search. *ACM SIGOPS operating systems review* 39, 5 (2005), 119–132.

[40] Fabian M Suchanek, Serge Abiteboul, and Pierre Senellart. 2011. PARIS: Probabilistic alignment of relations, instances, and schema. *PVLDB* 5, 3 (2011).

[41] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *WWW.*

[42] Jaime Teevan. 2007. The re: search engine: simultaneous support for finding and re-finding. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology, Newport, Rhode Island, USA, October 7-10, 2007.* 23–32. DOI: https://doi.org/10.1145/1294211.1294217

[43] Paul Trevithick and Mary Ruddy. 2012. Higgins – Personal Data Service. (2012). http://www.eclipse.org/higgins/

[44] Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: A Free Collaborative Knowledgebase. *CACM* 57, 10 (2014).

[45] Coral Walker and Hassan Alrehamy. 2015. Personal Data Lake with Data Gravity Pull. In *Fifth IEEE International Conference on Big Data and Cloud Computing, BDCloud 2015, Dalian, China, August 26-28, 2015.* 160–167. DOI: https://doi.org/10.1109/BDCloud.2015.62

[46] Wikipedia contributors. 2016. List of search engines – Desktop search engines. (2016). https://en.wikipedia.org/w/index.php?title=List_of_search_engines