université PARIS-SACLAY

école normale supérieure paris—saclay

NNT : 2017SACLN009

Thèse de doctorat de l'Université Paris-Saclay préparée à l'École normale supérieure Paris-Saclay

Ecole doctorale n°580
Sciences et technologies de l'information et de la communication
Spécialité de doctorat : Informatique

par

M. David Montoya

Une base de connaissance personnelle intégrant les données d'un utilisateur et une chronologie de ses activités

Thèse présentée et soutenue à Cachan, le 6 mars 2017.

Composition du Jury :

M.	Serge Abiteboul	Directeur de recherche Inria Paris	(Directeur de thèse)
M.	Nicolas Anciaux	Chargé de recherche Inria Saclay	(Examinateur)
Mme.	Salima Benbernou	Professeur Université Paris Descartes	(Président)
Mme.	Angela Bonifati	Professeur Université de Lyon	(Rapporteur)
M.	Patrick Comont	Directeur innovation et PI Engie	(Invité)
M.	Pierre Senellart	Professeur École normale supérieure	(Examinateur)
Mme.	Agnès Voisard	Professeur Université libre de Berlin	(Rapporteur)

Laboratoire Spécification et Vérification
École normale supérieure Paris-Saclay, UMR 8643 du CNRS
61 avenue du Président Wilson, 94235 Cachan Cedex, France

A personal knowledge base integrating user data and activity timeline

David Montoya

Abstract

Typical Internet users today have their data scattered over several devices, applications, and services. Managing and controlling one's data is increasingly difficult. In this thesis, we adopt the viewpoint that the user should be given the means to gather and integrate her data, under her full control. In that direction, we designed a system that integrates and enriches the data of a user from multiple heterogeneous sources of personal information into an RDF knowledge base. The system is open-source and implements a novel, extensible framework that facilitates the integration of new data sources and the development of new modules for deriving knowledge. We first show how user activity can be inferred from smartphone sensor data. We introduce a time-based clustering algorithm to extract *stay points* from location history data. Using data from additional mobile phone sensors, geographic information from OpenStreetMap, and public transportation schedules, we introduce a transportation mode recognition algorithm to derive the different modes and routes taken by the user when traveling. The algorithm derives the itinerary followed by the user by finding the most likely sequence in a linear-chain conditional random field whose feature functions are based on the output of a neural network. We also show how the system can integrate information from the user's email messages, calendars, address books, social network services, and location history into a coherent whole. To do so, it uses entity resolution to find the set of avatars used by each real-world contact and performs spatiotemporal alignment to connect each stay point with the event it corresponds to in the user's calendar. Finally, we show that such a system can also be used for multi-device and multi-system synchronization and allow knowledge to be pushed to the sources. We present extensive experiments.

Une base de connaissance personnelle intégrant les données d'un utilisateur et une chronologie de ses activités

David Montoya

Résumé

Aujourd'hui, la plupart des internautes ont leurs données dispersées dans plusieurs appareils, applications et services. La gestion et le contrôle de ses données sont de plus en plus difficiles. Dans cette thèse, nous adoptons le point de vue selon lequel l'utilisateur devrait se voir donner les moyens de récupérer et d'intégrer ses données, sous son contrôle total. À ce titre, nous avons conçu un système logiciel qui intègre et enrichit les données d'un utilisateur à partir de plusieurs sources hétérogènes de données personnelles dans une base de connaissances RDF. Le logiciel est libre, et son architecture innovante facilite l'intégration de nouvelles sources de données et le développement de nouveaux modules pour inférer de nouvelles connaissances. Nous montrons tout d'abord comment l'activité de l'utilisateur peut être déduite des données des capteurs de son téléphone intelligent. Nous présentons un algorithme pour retrouver les *points de séjour* d'un utilisateur à partir de son historique de localisation. À l'aide de ces données et de données provenant d'autres capteurs de son téléphone, d'informations géographiques provenant d'OpenStreetMap, et des horaires de transports en commun, nous présentons un algorithme de reconnaissance du mode de transport capable de retrouver les différents modes et lignes empruntés par un utilisateur lors de ses déplacements. L'algorithme reconnaît l'itinéraire pris par l'utilisateur en retrouvant la séquence la plus probable dans un champ aléatoire conditionnel dont les probabilités se basent sur la sortie d'un réseau de neurones artificiels. Nous montrons également comment le système peut intégrer les données du courrier électronique, des calendriers, des carnets d'adresses, des réseaux sociaux et de l'historique de localisation de l'utilisateur dans un ensemble cohérent. Pour ce faire, le système utilise un algorithme de résolution d'entité pour retrouver l'ensemble des différents comptes utilisés par chaque contact de l'utilisateur, et effectue un alignement spatio-temporel pour relier chaque point de séjour à l'événement auquel il correspond dans le calendrier de l'utilisateur. Enfin, nous montrons qu'un tel système peut également être employé pour faire de la synchronisation multi-système/multi-appareil et pour pousser de nouvelles connaissances vers les sources. Les résultats d'expériences approfondies sont présentés.

Contents

List of Figures

List of Tables

Acknowledgments

The works that are presented in this thesis have been funded by Sinovia, a French company and an entity of the Engie Ineo group. They are the object of a joint research agreement between Sinovia and Inria[1]*, a French public research institution, that allows Sinovia to employ a Ph.D. candidate.*

Throughout my doctoral studies and in the realization of this thesis, I have met, discussed with, worked with, been advised by, and benefited from the support of multiple individuals. I would like to acknowledge them here.

First, I would like to express my sincere gratitude toward Serge Abiteboul, my principal advisor. Despite his busy schedule, Serge remained available and committed to my progress. In this respect, I appreciated how he gave me the freedom to shape my work as I saw fit while giving me the necessary guidance to stay focused on attainable research objectives. While working with him, I appreciated both his frankness and humbleness and learned a lot from his pragmatic and goal-oriented approach to research. Serge's experience and optimism helped me gain confidence and give structure to my work. Above all, I am grateful to Serge for teaching and reminding me the value and importance of simple theories, precise definitions, and out-of-the-box thinking in research. Second, I would like to thank Pierre Senellart, my co-advisor, for his thoughtful insight and especially his direct involvement in some of the experiments described in this thesis. Pierre's high standards with respect to both form and content as well as helpful questions and suggestions allowed me to improve my work considerably. I greatly appreciated the authentic enthusiasm that Pierre expressed toward my work. Third, I would like to thank Franck Signorile, who was my supervisor at Sinovia, for his support and encouragements. Franck kept me at a balanced distance from Sinovia's business-oriented activities; he usually called me in only if he deemed that my help was essential and valuable, thus allowing me to acquire important industry experience while giving me the necessary time to work on my doctoral research. I am grateful to Serge, Pierre, and Franck for trusting me through this journey. Finally, I would like to thank the members of my jury, and especially Angela Bonifati and Agnès Voisard, for taking the time to review this thesis and for their helpful suggestions.

This thesis would not have been possible without Sinovia's agreement with Inria. I would like to thank Carlos Moreno, who was Engie Ineo's chief scientific

[1]The agreement is based on a French framework called *convention industrielle de formation et recherche en entreprise* (Cifre).

advisor, for helping us set up this agreement. I would also like to thank Edwige Brossard and Thomas Peaucelle, Sinovia's former managing directors, for ensuring the proper realization of Sinovia's collaboration with Inria.

Some of the works presented in this thesis have resulted from collaborations with other researchers. In developing Thymeflow, I had the opportunity to work with Fabian M. Suchanek from Télécom ParisTech and Thomas Pellissier Tanon from École normale supérieure de Lyon. Thomas spent about six months in our Inria research group and contributed significantly to the development of Thymeflow. During the preliminary investigations that led to the development of Movup, Jakub Kallas from École normale supérieure Paris-Saclay spent some time in our research group and participated in these investigations. Finally, I am very grateful to the many anonymous participants that provided us with some of their data for research purposes.

During my time at the Laboratoire Spécification Vérification (LSV) as well as Télécom ParisTech, I had fruitful discussions with fellow Ph.D. candidates and postdoctoral researchers: Karima Rafes, Simon Theissing, Antoine Amarilli, Luis Galárraga, and Roxana Horincar. I am thankful to Thida Iem, Luc Segoufin, Stéphanie Delaune, Stéphane Demri, and Laurent Fribourg for ensuring that my stay at the laboratory was pleasant.

Special mentions go to the rest of my colleagues at Sinovia. In particular, I would like to thank Romain Castillos, Manuel Odesser, Jérôme Bonnet, and especially Dominique Tran, François Camus, Stéphane Michaud, and Erwan Becquet for their contributions to the development of Thymeflow. I am very grateful to Steve Sampson for his important contribution to the development of Movup. I am grateful to Ilinca Vanneste, Manuel Odesser, Erwan Becquet, and especially Lucas Griffaton-Sonnet for contributing to the discussion about the business cases of personal information management systems. I would also like to thank Omar Fantazi, Bouthie Dramé, Mahamadou Koné, Paul-Henri Richard, Armand Abedin, and Alejandro Mejía for their support.

Many thanks to some of the computer science students at the École normale supérieure Paris-Saclay, namely Martin Gleize, Raphaël Bonaque, Nicolas Rolin, Alban Levy, Charles-Pierre Astolfi, and Émile Contal, with whom I had the opportunity to discuss some of the topics presented in this thesis.

Finally, I would like to express my gratitude toward my family. My elder brothers for their support during my studies in France. My parents for introducing me to science at a very young age, encouraging me to pursue a Ph.D. degree, and their unconditional support through all these years. My parents created Ceprecyt, a science academy for young people, in which I explored and learned about the natural world and technology by doing fun experiments and building different kinds of fascinating things. Undeniably, this experience ignited my curiosity and developed my taste toward science. Last but not least, I am deeply grateful to my wife for her constant belief in me and her invaluable support.

Introduction

Typical Internet users today have their data spread over several devices, applications, and services. They encounter more and more difficulties managing these data and are losing control over them. In this thesis, we take the point of view of Abiteboul, André, and Kaplan (2015), which argues that the solution to this problem is that users should manage their information by themselves, in their best interest, in a personal information management system. In that direction, we introduce important components for such a system. More precisely, we introduce tools to integrate multiple heterogeneous sources into a single knowledge base by pulling data from the sources, reconciling references to the same entities, and inferring a timeline of user activity. Although users today lack the necessary knowledge to manage their data on their own, this is a first step toward empowering them to do so.

The information of typical Internet users includes the messages they send, pictures they publish, events they are invited to, products they buy, web pages they visit, and places they go to. Furthermore, a large amount of data that concerns a user is generated and/or managed by other parties: friends' pictures, messages from colleagues, credit card transactions, telephone bills. Applications are developed by distinct entities and fitted to different purposes.

Typical users are unable to take full advantage of their data. First, they are not able to list the systems that host their data, nor globally search across these systems to find information. Second, answering questions or performing tasks requiring user information from multiple systems is hard: "how much did I spend during my last holiday trip?" "tell all my guests that the rendezvous place has changed", and "how many times did I go to a restaurant this year?" These issues limit the ability of users to exert control over their data and hinder the creation of value in a world where information is paramount.

The reason of this limitation is the lack of global integration across the different devices, applications, and services that host knowledge that is useful to the user. Usually, when a new application is developed, or a new kind of data is stored, a new silo is created. It may be because the stored data is not retrievable, is only partially retrievable, or retrievable in an unusable format. The purported reasons may be technical, or even legal, but mostly economical as integration does not come for free and requires that companies relinquish part of the economic advantage of holding users' data. Some companies, such as Google, Microsoft, and Apple, alleviate this problem by offering their users the ability to put all their data in a single platform, built as a coherent suite of online services. However, free market competitiveness and the users' desire not to depend on a single company

are serious issues with such a solution. Some services have started providing mechanical ways for retrieving data (e.g., through public APIs) and integrating them as part of their value proposition. However, someone still has to be in charge of managing all the users' information and making it globally available. In this thesis, we adopt the viewpoint that the user should be given the means to gather and integrate her data, under her full control, thereby resulting in more symmetry between users and service providers.

We first show how the user's location history on her mobile phone can be used to derive knowledge about the user's whereabouts. For instance, this will allow finding out which places she spends the most time in, which modes of transportation she takes most often, etc. These exemplify the tasks of inferring knowledge from available information and performing personal analytics.

Second, we show how to integrate the users' data into a personal knowledge base. These tasks are implemented in a system that we designed that leverages the different sources of user information by integrating them into a single model and facilitating new enrichments. The system's goal is personal knowledge integration. We show some examples of queries that the system can answer and its potential uses, such as multi-device and multi-system synchronization and pushing knowledge to the sources.

Our main contributions are as follows:

- A novel method for extracting the places that the user visited from her location history. It performs time-based clustering to process the locations in sequence, handling satellite, Wi-Fi, and cellular positioning data as well as the reported accuracy of each location.

- A transportation mode recognition algorithm that infers a detailed description of a user's journey, including public transportation routes, stops, and transfers from one mode to another. It uses data from the location, accelerometer, and Bluetooth sensors in a mobile device (smartphone), geographic information from OpenStreetMap, and transportation schedules published by local agencies in GTFS format. The algorithm infers the itinerary followed by the user by determining the most likely state sequence in a linear-chain conditional random field whose feature functions are based on the output of a feedforward neural network.

- An open-source system that integrates multiple heterogeneous sources of user information, including the user's calendar and location history, into a personal knowledge base. This is a web application that the user installs on her machine. She provides it with credentials to connect to different sources; then the system pulls data in and runs several algorithms to derive new knowledge. These include an entity resolution algorithm that reconciles the different avatars that a contact has and an alignment between events and the user's location history. Knowledge is stored in RDF using an ontology based on `schema.org`. Finally, the user can query the knowledge base via a SPARQL endpoint, and visualize the results in a table, in a map, or as a graph.

- An extensible framework that facilitates the integration of new data sources, such as web search or personal finance, and the development of new modules for deriving knowledge. To add a new kind of data source, a module that converts source data into RDF statements has to be implemented. To derive new kinds of knowledge, a routine that takes a delta of the knowledge base has to be implemented.

The outline of this thesis is as follows:

In Chapter 1, we define personal information in the context of an individual, and describe where it can come from. We then present issues users face when handling their information, present our vision of personal information management, and discuss how it helps in dealing with these issues. The chapter concludes with a review of the state of the art in the context of personal information management.

Chapter 2 presents a model for personal knowledge. First, we define personal knowledge by describing its nature and the different aspects that we intend to capture. Then, we specify an RDFS ontology for representing the different concepts and their relations. Finally, we present in detail some of the goals addressed by this thesis.

Chapter 3 presents the different sources of personal information specifically used in this thesis. For each source, we describe the significance of the information it provides and how acquisition can be performed, i.e., the technologies involved and the issues in accessing information. In particular, we present the two applications we developed for acquiring mobile device sensor data. The chapter concludes with related work in the domains of data extraction and transformation.

Chapter 4 presents our work on extracting stays from the user's location history. First, we define the problem and illustrate it through visualizations of a location history. Then, we describe Thyme, an algorithm we developed to solve this problem. Finally, we present an evaluation of this algorithm and some related work.

Chapter 5 presents our work on inferring the transportation mode and routes used by the user using mobile device sensor data, geographic information from OpenStreetMap, and public transportation schedules. We introduce Movup, a system and an algorithm that we designed and implemented to perform itinerary recognition. Then, we describe the different sources of information it uses, how it acquires them and preprocesses the data. After that, the algorithm is described and we present the evaluation we performed. Finally, we review some related work.

Chapter 6 presents Thymeflow, the system that we designed in an attempt to fulfill our vision of a personal information management system. We describe the specific problems it addresses and its overall architecture. Then, we specify the different knowledge integration and enrichment routines implemented within the system, and report on their evaluation. Finally, we give examples of queries and potential uses, and review some related work in this domain.

Finally, we conclude this thesis by outlining some lessons learned through our research and suggesting directions for future work.

Chapter 1

Personal information management

Information is fundamental to every human organization. In business, access to accurate, reliable, and timely information is a factor in competitive advantage. Since the beginning of recorded history, human settlements have kept track of productivity metrics of their various activities, for the collection of tax for instance, or for making informed decisions (Schmandt-Besserat 1996). The recognition of science and technology as a driver of increasing efficiency and progress has also meant that organizations making the most out of past experience, through the careful consolidation of best practices and strategic information sharing, would tend to lead in their respective fields (Porter and Millar 1985; Argote and Ingram 2000).

Today's Information Age is characterized by exponential trends in the velocity of information transmission, the capacity of information storage, and computational power. Such trends have transformed the traditional way organizations manage information and increased the speed at which information is generated: file cabinets have become data centers, and increasing automation and computer assistance have increased the opportunities for data collection.

Such transformations have brought about new challenges for large organizations. The increase in volume, velocity, but also variety of information has rendered traditional data processing tools inefficient, while at the same time creating new business opportunities for those making the most out of this new information. This has been the challenge of "big data".

In parallel, individuals have faced new difficulties. The digitalization of one's calendar, address book, and communications (email, SMS) have made it more convenient for individuals to keep more and more information in digital form, instead of paper form or their memory, thanks to the success and ubiquity of search in modern applications. Computers can now function as an extension of one's memory. However, this convenience have come at great costs. One's address book is no longer kept safe within paper sheets in one's pocket. Its contents are now replicated in multiple machines, in the "cloud", where others (e.g., system administrators, the government, or hackers) could technically have access to it. Meanwhile, individuals also face their own version of "big data". In fact, if we

consider individuals solely responsible for managing their own information and the only ones capable of doing so to the best of their interests, we realize that *volume*, *velocity*, and *variety* are also problematic to the individual, but at a much smaller scale, that is, at the scale of the individual. Individuals do not have the knowledge and resources to manage their own information, in their best interest.

In this chapter, we define what is the information of concern to an individual (one's personal information) and where it comes from. Finally, we discuss some challenges related to personal information and ways they can be tackled through personal information management.

1.1 What is personal information?

We view *personal information* as a particular kind of *information*. In information science, one can see *information* as an interpretation of *data*, which are the lowest kind of stimuli by which machines communicate with us and which each other. Informally, we define an individual's *personal information* as the set of all information that is linked to this individual *in some way*.

Our informal definition is similar to the notions of "personally identifiable information" and "personal data" respectively used in the privacy protection laws of the United States (93rd United States Congress 1974) and the European Union (European Parliament and Council 1995), but differ in context. Personally identifiable information (or personal data) is understood in the context of the information that an agency (e.g., an organization, a company) holds as:

> any information about an individual maintained by an agency, including (1) any information that can be used to distinguish or trace an individual's identity, such as name, Social Security number, date and place of birth, mother's maiden name, or biometric records; and (2) any other information that is linked or linkable to an individual, such as medical, educational, financial, and employment information. (Koontz 2008)

Instead, our definition understands the *personal information* of an individual in the context of all of the world's information. It includes all information that, given access to all systems and data stores in the world, could reasonably be inferred as being linked *in some way* to this individual.

In the following, we describe more precisely the notion of *information* and what is meant by "linked *in some way*". Also, personal information shall hereinafter be understood from the perspective of a particular fictitious individual, *Alice*, our prototypical owner of personal information.

From information to knowledge

We understand *data* as the sequences of symbols stored, generated, and used by machines for keeping memory and communicating. In the context of the data, information, knowledge, and wisdom hierarchy (the DIKW hierarchy), information

is data that has meaning or purpose (Rowley 2007). For instance, an email message file is data when seen as a sequence of bits or characters, information are the contents of this message: from which address was it from? when was it sent? and what is the subject? Similarly, the sequence of GPS navigation messages picked up by Alice's smartphone are data, information are the geographic coordinates and time representing Alice's location. Such information is useful to the machine. The machine can use the information contained in an email message to provide a default reply-to address and reply subject, and add part of the body of the original message to the response. Geographic coordinates, on the other hand, can be used to center the map on Alice's current location when she requests for directions, which is a convenient feature.

This level of machine interpretation is not enough. A feature recently introduced by a popular email client suggests Alice three different possible responses to an email message based on its contents (Miklós 2015), saving her precious time 10 percent of the time (Moynihan 2016). To suggest appropriate responses, the machine might need to know the relationship that she has with the addressee: is it a business client or a close friend? The topic of the message might also be useful for inferring relevant Cc: addressees to add to the response: Alice's companion, her boss, etc. Her location, recorded over time using the GPS of her smartphone, becomes really useful when seen as a history of places she has been to: the supermarket, restaurants, the gym, work, home, etc. The machine could use such information to suggest possible meeting places for future appointments: how about meeting this particular *fish restaurant* that is near "Alice's workplace" for the "Lunch with Bob" event? Following the terminology of the DIKW hierarchy, we use the term *personal knowledge* to refer to *personal information* interpretable by a machine in such way. In simple terms, personal knowledge is a machine representation and interpretation of personal information using a structure and semantics that are more closely related to the tasks Alice wants to perform. For instance, such an interpretation could turn the list of contacts on Alice's smartphone into a coherent list of people she relates to: without duplicates, annotated with the kind relationship she has with them and the different ways she communicates or interacts with them (e.g., telephone calls, email or text messaging, or face-to-face communication).

In which ways can information be personal?

As mentioned earlier, Alice's personal information is information linked in one way or another to herself. One could argue that all information in the world is connected to her, assuming the small-world thesis that each one of us is at most six "steps" away from everything else in the world. However, we would like to retain only as personal, information that is linked to her "in one step", in a way that provides useful information for or about her activities and needs. To this extent, Jones (2010) defines several ways in which information can be personal, in a one-step connection:

1. Controlled by Alice: email messages in her email accounts, files in her computer's drive, contact information in her address book.

2. About her: credit, medical, web browsing histories.

3. Directed toward her: phone calls, email messages.

4. Sent by her: email messages, published articles, pictures posted on social networks.

5. Experienced by her: books she has read, web pages she has browsed, places she has visited.

6. Relevant to her: the perfect movie to watch tonight, her next holiday destination.

The classes above are non-exclusive. Together, they capture all kinds of personal information discussed in this thesis.

1.2 How much information is personal?

Personal information, as defined in the previous section, spans a wide range of domains. Information technology is now being used in most human activities as either a production or support tool, and with more than 6.4 billion devices now connected to the Internet (Gartner 2015), the amount of data generated every day totals 2.5 quintillion bytes (IBM 2016). What amount of it is personal? At first glance, information such as satellite imagery, air quality sensor measurements, and surveillance camera footage would appear to be non personal. Nevertheless, the satellite images of a city and information about its air quality could be useful to and used by at least some of its inhabitants, and surveillance camera footage is used by a security guard or police officer when an incident occurs. Overall, any information that is produced as part of a process or an activity involving human monitoring or intervention could at least end up being *experienced* by some individual. Thus, aside from information produced, exchanged, and used by machine controlled processes with neither a human interface nor log keeping for possible future human analysis, a good amount of information is personal to at least one individual.

In this thesis, we focus on a representation of personal information that captures many activities concerning many individuals, and avoid delving into overly particular cases, such as information generated by Alice's specific occupation. Next, given this context, we discuss how information about Alice is produced, and where personal information is stored.

How is information about an individual produced?

Information about Alice is produced either *actively* or *passively*. For instance, Alice *actively* sends email messages, buys books online, posts pictures on social media, performs web search queries, calls friends, and fills tax declarations. At the same time, *passively*, email servers record the IP address of her email client, the online book retailer builds a profile of her reading taste, the search engine builds a

history of her queries, the telecommunications operator builds a metadata record of her calls, and the state's online tax service computes the probability that she may be committing tax evasion. Passively also, for instance, connected devices that Alice owns record her location, heart rate, and the temperature at home. Sometimes, this is done by devices that she does not own: surveillance cameras record images of her walking outdoors, and the local transportation agency logs the times and places where she uses her transit pass (a smart card) to enter or leave the network.

The distinction between active and passive production of personal information is helpful in that it highlights that in many human activities in which a machine is directly or indirectly involved, information is generated not only for the proper fulfillment of this activity (e.g., invoice generation when the user orders a book online), but also for other purposes, possibly serving different interests (e.g., building a profile of the user's reading preferences for recommending books to her). Depending on the interests served, some of this information might be readily available to Alice (e.g., her email messages), or nearly impossible to obtain (e.g., an image of her taken by surveillance camera).

Where is personal information stored?

Personal information can be found practically everywhere. The storage location of personal information depends on who should be able to access it, what it is used for, and whose interests it responds to. Let us enumerate the different locations where Alice's information may reside:

Personal computer Alice has a desktop computer at home and uses a laptop for work. These computers hold all kinds of documents and media that Alice views, edits and organizes. On each computer, she has installed an email client that constantly synchronizes email messages in her inbox from a remote server. She downloads documents and media that she finds on the Web, or that other people have shared with her. She also has installed a file synchronization client (e.g., Dropbox, Google Drive, or Microsoft OneDrive) that constantly synchronizes documents and photos on her computer with a server in the "cloud", so that she can share them with friends or colleagues.

Private servers Alice owns a home server, but she has also acquired a private server that is provided by an Internet hosting service. On her home server, she has installed a file sharing system that functions as network-attached storage for home devices. On the remove private server, she hosts an email server for her family. The modem provided by her Internet service provider also comes with basic network-attached storage functionalities.

Connected devices Alice owns all kinds of connected devices. She carries a smartphone and a smartwatch with her all the time, which keep a log of her activity and location over time. Home automation devices, as well as mobile appliances such as connected cars, are not only capable of delivering their principal function, e.g., home surveillance, smart heating, or mobility.

They also serve as sensors capable of gathering richer information that is relevant to Alice: the times when the members of her family were at home, how comfortable is it to live in it, or how much time she spends commuting per week. Recent measurement data may be stored up to some extent on each of these devices, the rest is uploaded for archival purposes or further analysis to a private server or to the "cloud". Her smartphone and her tablet also function as ultra mobile alternatives to her personal computer: they are equipped with a web browser, a calendar application, a contact manager, an email client, and are able to hold substantial document and media content. Aside from information that is also present in her personal computer, Alice can use these devices to take pictures and record videos and store them, and her smartphone keeps a record of her text messages and metadata about her phone calls.

Organizational servers The company where Alice works at provides email, calendar, file, and directory services, which are hosted on-premises, to all of its employees and associates. These services contain information about Alice, her work, and the people she works with.

The Cloud Web-based email services (webmail), search engines, social networking services, communication platforms, video hosting services, as well as e-commerce, e-banking, and e-government services provide many facilities that Alice interacts with. Through this interaction, Alice's personal information is collected and ends up being stored, temporarily or permanently, in what we refer to as *the Cloud*.

Protected/Offline servers Transportation agencies, shops, local government agencies keep information about Alice in their people and client databases, which she cannot interact with.

Here are some figures. The average Internet user (ages 16-64) owns 3.64 connected devices (including personal computers) (Buckle 2016). She owns 5.54 social media accounts (Mander 2015). In a different sample of Internet users, the number of online accounts a user has on average was estimated at 90 (Le Bras 2015). In another study, it was estimated that the average employee of a small or medium-size business uses 5.5 applications provided by the organization (Intermedia 2014). While we do not have statistics on the number of private servers owned by the typical Internet user, we believe that as of now not many users have one, as they tend to require system administration knowledge to install and maintain. However, we believe that the trend is on the rise, as the cost of hosting and maintaining a private server is driven down by more efficient virtualization and more powerful deployment tools, and offers by Internet and television service providers increasingly diversify to include cloud-like services hosted at home using a set-top box.

1.3 Issues with personal information

The use of personal information, while bringing a lot of convenience and features (e.g., quickly finding the telephone number of a friend) and enhancing Alice's own capabilities through information technology, does not come without issues.

Information overload

Information overload refers to the difficulty in understanding and making good decisions caused by the exposure to too much information. Speier, Valacich, and Vessey (1999) concluded, based on prior research, that:

> Information overload occurs when the amount of input to a system exceeds its processing capacity [...]. Decision makers have fairly limited cognitive processing capacity [...]. Consequently, when information overload occurs, it is likely that a reduction in decision quality will occur. Research from a number of disciplines (e.g., accounting, finance, consumer behavior) has found, for example, that information overload decreases decision quality [...], increases the time required to make a decision, and increases confusion regarding the decision [...].

In the context of a single individual, it is not new that personal information (the input) exceeds the processing capacity of the system (the individual). It was reported that the average U.S. person spent 12 hours of leisure time per day consuming 100,500 words and 34 gigabytes (Bohn and Short 2009). This includes information consumed through mobile phones, the Internet, email, television, radio, newspapers, books, etc. This accounts for *experienced* personal information. To put it into perspective, the average white collar U.S. worker spends 6 hours a day checking email (Naragon 2015). The term *email overload* has been used to refer to the fact that many individuals use email well beyond its original purpose of asynchronous communication (they use it also for task management, scheduling, and archiving), and to the feeling of disorganization associated with high unread message counts (Whittaker and Sidner 1996; Fisher et al. 2006; Grevet et al. 2014). Overall, not all of the information that Alice consumes is precisely relevant nor useful to her. However, the low signal-to-noise ratio caused by an overload of not-so-useful information means that useful information becomes less accessible and this hinders her ability to make good decisions. Another problem with personal information is that the definition of usefulness depends on the user and this is not initially clear to the machines that process them. For instance, some information that reaches the user is the result of contradictory interests: marketers are sometimes forced to reach wide audiences when advertising their products, which includes uninterested consumers, yet consumers want to know about deals for products they might be interested in.

Loss of functionalities due to fragmentation

A natural problem caused by the variety of sources and storage locations of personal information is *fragmentation*. Fragmentation is the phenomenon by

which personal information of a single individual is scattered throughout multiple information systems and storage devices, and access to them, by computing agents or the individual, cannot be performed, uniformly and efficiently, from a single point. Before the Information Age, fragmentation occurred each time information migrated away from the human brain, to be stored for instance in notebooks, files, or paper calendars, as the notes would typically find themselves scattered at home or throughout the office. A smartphone now offers the possibility of having these notes in one location, within arm's reach. However, inherently *social* information, i.e., information that is personal to multiple individuals, is naturally fragmented. From the moment an individual is born, social information about the individual exists: civil records, birth medical records, baby pictures on the fridge, baby congratulations cards from the friends and colleagues of the parents. Also, the generation, management, and representation of information is fragmented by the fact that applications are developed by distinct entities and fitted for different purposes. But why is fragmentation a problem?

To answer this question, we distinguish two kinds of fragmentation: *storage fragmentation* and *access fragmentation*. *Storage fragmentation* means that no single computing agent is able to retrieve all of Alice's information without communicating with other agents. Storage fragmentation is a problem when the cost of communication (i.e., in terms of bandwidth and latency) is relatively high with respect to the performed task. Differently, *access fragmentation* means that no single computing agent is able to retrieve all of Alice's information in an interoperable fashion, in a unified representation, and for use in a single coherent task. Certain tasks become harder or even impossible to perform. For instance, consider WhatsApp (WhatsApp Inc. 2017), an instant messaging service for smartphones used by one billion people each month (WhatsApp Inc. 2016). Although WhatsApp stores all received and sent messages in Alice's smartphone, Alice is unable to search across the different types of messages (email, SMS, MMS, and WhatsApp) stored in her smartphone from a single interface. The reason: WhatsApp does not provide a programmatic way of reading or searching messages. WhatsApp, works as an *information silo*, incapable of handing over the information acquired or generated by Alice while using it. Storage fragmentation is a problem for efficiency, access fragmentation is a problem for functionality. The latter prevents Alice from performing certain tasks in which we need to query information from multiple sources. Functionally speaking, access fragmentation is thus the most problematic. Due to this fragmentation, Alice cannot for instance *globally search* within her personal information, and lacks a complete overview of what information she has, and where it is.

Loss of control: sharing & privacy

As mentioned previously, fragmentation is in part responsible for Alice's lack of a complete overview of her information. One result is that, in a lot of cases, she does not entirely control her personal information, who she shares or does not share it with, and what people do with it. Imagine, for instance, that Alice uploaded a photo album to her favorite social networking site. On the one hand,

she might want to *share it with a friend* who unfortunately does not have an account on this site, and not be able to do so. On the other hand, she might want to only share it with certain friends, and only be able to share it with her whole network. But even if Alice was allowed fine-grained control of who can see her album, she might not be able to prevent her friends from sharing it with other people as well. Her ability to *control the dissemination of her personal information* is limited. This problem is not new to the Information Age. However, the ease by which digital information can be searched for, copied, shared, and reshared with today's technology exacerbates it. This was alleviated by the development of new mechanisms by which Alice can exert control over the dissemination of personal information, such as the "right to be forgotten" advocated by the European Union (European Parliament and Council 2016). By this right, Alice can request a search engine to remove the URL of a page that talks about her from its search results. Nevertheless, this right has limitations and it only allows her to restrain the further dissemination of a piece of information that has already been disclosed.

A related concern is *the protection of personal information* against *misuse*, e.g., including *fraud* or *identity theft*. According to Gemalto (2016), the theft of non-financial personally identifiable information has been of increasing interest to hackers, and has recently superseded the theft of financial data as the leading type of data breach, with 64% out of the 974 breaches reported in the first half of 2016 being associated with identity theft. At the same time, recent disclosures of mass global surveillance programs involving large Internet companies that manage personal information of millions of users (Greenwald 2013; Gellman and Poitras 2013) has decreased the relative trust people had on their services and increased the importance of data control and security in organizations. Advocates of such programs argue that the privacy of users that have "nothing to hide" is not threaten by surveillance. However, Solove (2007) has criticized this argument as relying on the assumption "that privacy is about hiding bad things", which they argue is problematic assumption since it leads to an "unproductive discussion of information people would likely want or not want to hide". On a related note, some companies, known as "information brokers", specialize in the trading of personal information, which they collect from various sources and sell to other parties without the knowledge nor the consent of the concerned individuals (Office of Oversight and Investigations Majority Staff 2013). Can better control of personal information help improve security and privacy?

Control of personal information has been recognized as an essential component of privacy (Solove 2008). Paradoxically, privacy concerned Internet users sometimes engage in behaviors that are not consistent with their concerns (Barnes 2006; Dienlin and Trepte 2015). While the reasons behind these behaviors remain unclear, we believe that having a better overview of one's personal information, knowing what it contains, where it resides, and who might have it, is instrumental to improving the information security and privacy of individuals. The availability of such an overview may even lead to the development of tools that raise Internet users' awareness of these problems, by providing them insight on the particular risks they are exposed to.

Loss of freedom: vendor lock-in

Vendor lock-in is a practice that makes Alice dependent on a single vendor for products and services. Alice is not able to switch to another vendor without incurring substantial costs. In the context of personal information, vendor lock-in limits Alice's ability to have at her disposal, at any time, and in an interoperable format, all of the personal information about her contained in the products and services of the vendor. While vendor lock-in has mainly economic reasons, Alice's inability to export her information can also be a side-effect of the vendor's enforcement of strict privacy policies or laws and technical limitations of certain features. Depending on the amount and the kind of information that the vendor holds about her, she could be tempted to switch vendors without migrating her data. However, some vendors, such as Google, Microsoft, and Apple, provide a comprehensive suite of services and products coherently integrated with each other in a single platform. While this provides a partial relief to personal information fragmentation, it also encourages Alice to put most of her information in one platform, making the switch much less tempting.

1.4 What is personal information management?

We can view personal information management (PIM) as the set activities concerned with the cycle of personal information: the acquisition of personal information from one or more sources, the *protection* and the *distribution* of that information to those who need it and should have access to it, and finally their *archival* or *deletion*. This definition is borrowed from the more general concept of *information management*, which concerns the cycle of organizational activity. The relations between *information management*, *data management*, and *knowledge management* are analogous to the relations between *data*, *information*, and *knowledge* in terms of the DIKW hierarchy (Venkatraman 1994). While organizations have always been interested in improving their efficiency, productivity, and decision making through better information management, individuals are also genuinely concerned, although at their own scale.

Jones (2010) gives a more specific definition of PIM:

> Personal information management refers to both the practice and the study of the activities a person performs in order to acquire or create, store, organize, maintain, retrieve, use and distribute the information needed to meet life's many goals [...] and to fulfill life's many roles and responsibilities [...].

This definition puts an emphasis on the simple goal that drives most individuals: managing information is a means to managing one's life. To this end, one of the main concerns of Alice is finding and re-finding information (Jones 2010), which she needs when solving trivial tasks, e.g., checking if she is available tomorrow for an appointment at 10 a.m. with her dentist or if she is the one in charge of picking up the kids after school later in the day. Other PIM tasks, such as keeping and organizing information, are in fact of no direct concern to her when

she needs to meet her goals. These tasks, which are concerned with determining what information should be kept and in which format, are however necessary in anticipation that they will later help Alice find the right information in a prompt and efficient manner. In his visionary essay, Bush (1945) expressed that speed and flexibility of retrieving information be an essential feature of a device that keeps all the personal information of an individual:

> A memex is a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory.

In this thesis, we take the point of view that PIM is a means to take back control of one's life, by making the most out of one's personal information and empowering oneself. Individuals, overwhelmed by information overload, afraid that their information might fall into the wrong hands, and lured by vendors to give away their information in exchange of services, are still unable to efficiently find the right information within their data and make the most out of it in an age where algorithms are capable of phenomenal feats.

We consider two aspects of PIM: overcome *fragmentation* and the *loss of functionalities* by bringing together and enriching personal information. We believe that these aspects are the first steps toward describing Alice's personal information as *personal knowledge*. The first aspect is about providing Alice with a standardized access to her personal information, no matter the source, the location, and the purpose. It is also concerned with making connections between elements in her personal information more explicit: that an event present in her work calendar and another one in the calendar of her social network service represent the same entity, or that two contacts in her smartphone represent the same person. The second aspect is concerned with deriving new facts from Alice's information, e.g., through automated reasoning, in order to remove incoherences and infer a timeline of the user's activity.

A system for personal information management

One vision of PIM is that of a system, the *personal information management system* (PIMS) (Abiteboul, André, and Kaplan 2015), that centralizes and provides an integrated and coherent view the user's personal information. The PIMS has complete access to the user's personal information residing in external services. Alice completely controls her PIMS and retains the ability, if she wants so, to retrieve all the information it contains, extend it, modify it, and share it with whoever she wants. Able to function as a manager of the user's information that is spread out across different systems, the PIMS gives Alice better control over her personal information. By centralizing and providing a coherent view of her information, the PIMS allows new features previously prevented by fragmentation: global search, automatic synchronization between devices, a hub for her connected devices, and advanced data analysis. Finally, the improved control over her information puts Alice in a better position to negotiate services with vendors

interested in this information. In this regard, a PIMS may be used to establish some symmetry between users (customers) and vendors. One goal of this thesis is to develop such a PIMS.

The state of the art of personal information management

There have been significant research efforts and projects attempting to fulfill the different aspects of PIM: *finding and re-finding*, *keeping*, and *organizing* personal information. Some of them have focused on one aspect, while other have tackled multiple ones. In the following, we present some notable contributions. A little outdated but very thorough review of the state of the art may be found in Jones and Teevan (2011)'s book.

Finding and re-finding personal information

Significant effort has been put into understanding and improving how individuals go about retrieving a piece of information to meet a particular need.

Desktop search For searching within the information stored in Alice's personal computer, desktop search tools have been developed for various operating systems and platforms. Alice performs full-text search: a query is a sequence of words, and the result is a collection of *documents* containing one or several of these words. At its most basic level, *documents* are the individual files and folders on the file-system. However, documents may also be the email messages, calendar events, contacts, pages in the web browsing history, and applications stored or installed in the user's computer. Search may be performed on both the content (e.g, the content of a text document, the description of a calendar event, or the text content of a web page) and the metadata (e.g., the filename, folder name, file type, file author, modification date, or page title), and queries may include special constructs to filter on each of these attributes. A comprehensive list of desktop search tools can be found in (Wikipedia contributors 2016). Many of these tools rely on being able to parse several file formats and the maintenance of sophisticated indexes. In particular, the IRIS (Cheyer, Park, and Giuli 2005) and NEPOMUK (Groza et al. 2007) projects used Semantic Web (Berners-Lee, Hendler, and Lassila 2001) technologies to provide an integrated view of the knowledge stored in the user's computer. These two projects went beyond desktop search by providing facilities for exchanging information between different desktop applications.

Re-finding and personalized search Some research efforts have focused on ameliorating the process of finding information that the user remembers having seen in a certain context. This process is called *re-finding*. One idea is to exploit what Alice remembers about the moment she saw this information (the meta-information) (Dumais et al. 2003). Another idea is to exploit for instance the *temporal locality* of a file, i.e., the set of files that were accessed immediately after or before this file (Soules and Ganger 2005), or the sequence of links followed by the user to find a web page (Teevan 2007). On another note, a profile of

the user's interests built from personal information such as the user's email messages, calendar events, documents, or web browsing history, can be used to provide personalized search, which has been show to be more efficient than non-personalized web search (Teevan, Dumais, and Horvitz 2005).

Intelligent personal assistants The emergence of *intelligent personal assistants* has opened up new possibilities and may improve in the long run the user experience of a PIMS. Also known as *(intelligent) virtual assistants*, such assistants are characterized by natural language user interfaces (text or voice-based) and the ability to perform advanced tasks on behalf of user (e.g., making dinner reservations, purchasing event tickets, or making travel arrangements) or provide information on request (e.g., "when is my next appointment?" or "what is the weather like tomorrow?") by using multiple sources of information and interacting with other systems. A list of open-source and proprietary such assistants can be found in Wikipedia contributors (2017).

Keeping personal information

The *keeping* aspect of PIM addresses the question: what kind of information should be captured and stored in digital form? Today, some aspects of Alice's life still eludes digital form (e.g., post-its and whiteboard notes). However, evidence of unmet design needs in today's PIM tools suggests that the improvement of these tools may reduce the amount of information eluding digital form (Bernstein et al. 2008). In the future, Alice might expect increasingly more aspects of her life being captured digitally. But can too much be captured?

Total capture A central idea of Bush (1945)'s vision is the creation of a device that is able to digitally capture all the experiences and knowledge acquired by Alice, so that it can act as a supplement to her memory. *Lifelogging* is an approach that consists in visually capturing the world that the user sees in her everyday life. It started in the 1980s with pioneers such as Steve Mann, Gordon Bell, Jennifer Ringley, and Josh Harris, a history that is well described by O'Hara, Tuffield, and Shadbolt (2008). MyLifeBits is a notable example of a system that was built to store all kinds of lifelog data, for which Gordon Bell was the experimental subject (Gemmell, Bell, Lueder, et al. 2002; Gemmell, Bell, and Lueder 2006; Bell and Gemmell 2009). Driven by the advent of more advanced and efficient wearable devices able to capture the different aspects of one's life and also facilitated by the development of cheaper and faster storage, the activity evolved into the indiscriminate logging of more data, and more kinds of data of one's daily life. Lifelogging activities include recording a history of machine enabled tasks (e.g., electronic communications, web browsing history, and document manipulation), actively capturing life's activities (e.g., writing on an electronic diary, blogging, photo taking, and video recording), passively capturing what the user sees and hears (e.g., via a wearable camera), monitoring personal biometrics and activity tracking (e.g., steps taken, distance traveled, caloric output, and sleep quality), logging the mobile device context (e.g., the

user's outdoor or indoor location, her movement, the ambient pressure, temperature, and humidity, and the identification of nearby users via Bluetooth), and logging environmental context (e.g., smart home sensing, presence detectors, and electricity, gas, and water consumption) (Gurrin, Smeaton, and Doherty 2014). Lifelogging does not require that analysis of the logged information be known or understood at the time logging occurs. One issue of indiscriminate logging is the accumulation of information without much use besides the prospect of its huge future potential (Bell and Gemmell 2009). In turn, *lifelogging* puts a burden on Alice by increasing information overload and creating more data management problems. Thus, one insight provided by prior research is that the effort should be put on selectivity rather than total capture (Sellen and Whittaker 2010).

Quantified self Different from total capture, the *quantified self* is a movement to incorporate data acquisition technologies on certain focused aspects of the user's daily life (Gurrin, Smeaton, and Doherty 2014). The quantified self focuses on logging experiences with a clearer understanding of the goals, such as exercise levels for fitness and health care. This has led to the development of specialized wearable devices (e.g., smartbands and smartwatches) for monitoring personal biometrics and activity tracking.

Organizing personal information

The last aspect of PIM deals with the management and organization of information. It is also concerned with the management of privacy, security, distribution, and enrichment of information.

Personal data service A *personal data service* (PDS) lets Alice store, manage, and deploy her information in a structured way. A PDS may be used to manage different identities, called "cards". These identities are pieces of her personal information (e.g., her address, preferences, affiliations, and contacts) that can she can share with an external service (e.g., an e-commerce or a social networking site). An example is Higgins (Trevithick and Ruddy 2012), which is a PDS that works as Alice's identity provider. Alice can use Higgins to register and authenticate to different services while controlling the information that Higgins provides to those services.

A PDS may also work as the central point of information exchange between external services. For instance, an application that recommends new tracks based on what Alice likes to listen may need to access Alice's listening history. To do so, the application can request access to external services that have information about her listening history and use different adapters to extract this information in the required format. Instead, the PDS centralizes Alice's personal information so that the application only needs to authenticate to one service (the PDS) using one adapter. OpenPDS (Montjoye et al. 2014) and the Hub of All Things (HAT Data Exchange Ltd. 2017) are examples of such PDSs. OpenPDS is particular in that it focuses on preserving the Alice's privacy by providing aggregated information to third-party applications instead of raw data: Applications submit queries to

the user's PDS, which ensures that all the sensitive data processing takes places within the PDS and only information of reduced dimensionality is returned to the application. The Hub of All Things particularly focuses on centralization the information of the user's connected devices.

Managing consent and information flow MyData (Poikola, Kuikkanieni, and Honko 2015) describes a consent management framework that allows Alice to control the data flow between a service that has information about her and a service uses this information. In this framework, a central system holds all the necessary credentials to access Alice's services on her behalf. Contrary to a PDS, the central system does not keep a copy of the information contained in those services. Alice specifies rules that determine the information that is authorized to flow from one service to another. The central system implements these roles by providing or revoking the necessary authorizations between each of those services. When two services exchange Alice's information, the actual information does not need to flow through the central system. The framework makes provisions for the possibility that two services exchange Alice's information without her consent if they are legally entitled to do so (e.g., the exchange of information between two public bodies for tax matters), in which case the central system simply notifies Alice of this exchange. The framework is a proposition that is still at its early stage. It is an all-or-nothing approach that represents a paradigm shift from currently implemented flows of personal information between organizations and services.

Lifestreams Organizing information as a time-ordered stream of documents, called a *lifestream*, has been proposed as a simple scheme for reducing the time Alice spends on manually organizing documents into a hierarchical file system (E. Freeman and Gelernter 1996; E. T. Freeman 1997). It has the advantage of providing a unified view of Alice'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 information loggers (Sjöberg et al. 2016).

Knowledge bases Several projects have leveraged knowledge representation technologies to organize Alice's personal information into a graph with an extensible schema and semantics. Such a structure allows the representation of things like "this file, authored by this person, was presented at this meeting about this project". Notable projects include Haystack (Karger et al. 2005), SEMEX (Dong and A. Y. Halevy 2005), IRIS (Cheyer, Park, and Giuli 2005), and NEPOMUK (Groza et al. 2007). They integrate several sources of user information, including documents, media, email messages, contacts, calendars, chats, and web browsing history.

Personal dataspaces A *dataspace* (Franklin, A. Halevy, and Maier 2005) is an abstraction in data management by which access to data from different sources is provided along with very basic integration. Data is said to *co-exist* in a

dataspace. A dataspace is able to interpret the data formats used by the sources but does not necessarily integrate their information into a common representation. For example a dataspace may provide full-text search over the information stored. Data integration tasks are postponed until absolutely necessary. In the context of PIM, personal dataspaces have been proposed as a way of dealing with heterogeneity while providing basic navigation and search tools that are useful to the user (Dittrich and Salles 2006; Blunschi et al. 2007).

Personal data lakes A *data lake* is a method of storing data in its raw format within a single system. It defers the responsibility of understanding the data to the consumer. To manage Alice's personal information, the *personal data lake* (Walker and Alrehamy 2015) has been proposed. Data is decomposed into entities that are given unique identifiers and associated with some metadata (e.g., the type of data, modification date, source, and context). Consumers retrieve entities by using queries that filter on these metadata attributes.

Self-hosted personal clouds In response to proprietary storage service providers (e.g., Dropbox and Google Drive), open-source solutions have been developed that Alice can use to host her own *personal cloud*. Some of these solutions, including ownCloud (ownCloud 2016) and Cozy (Cozy Cloud 2016), have evolved into application platforms that host various kinds of user-oriented services, e.g., email, calendar and contact management. These systems leverage multi-device synchronization facilities and standard protocols to facilitate integration with existing contact managers and calendars. The Cozy system is notable for its ability to import data from different kinds of services (e.g., activity tracking and financial services) into the system's document-oriented database. These tools bring the convenience of modern software-as-a-service solutions (e.g., there is nothing to install client-side) while allowing Alice to be in control, not give away all of her privacy, and free herself from vendor lock-in.

1.5 Conclusion

In this chapter, we motivated the need for personal information management by discussing some of the issues that users face with their information, and presented our vision of PIM as the realization of a system under the user's control that integrates and provides a coherent view of the user's personal information. While PIM is not a new domain, the user's inability of exerting control over her information is a problem of increasing concern. Building upon such a system, personal information could be elevated into personal knowledge, leading to the realization of a *personal knowledge base*. Storage, integration, and search are aspects of PIM that have been well considered in the past, but the development of personal knowledge bases has fallen behind the development of more convenient tools for a greater range of uses that most often create new personal information silos. Such trend has not only caused the integration of personal information to be more difficult, as a consequence of the increased heterogeneity and fragmentation

of this information, but has also made this integration more desirable, as a result of the new opportunities created by the availability of richer information. One of the goals of this thesis is to develop a system that attempts to fulfill our vision of a PIMS and show some novel ways in which the user's personal information can be enriched based on the integration of different sources of personal information. The next chapter presents and discusses a model for personal knowledge that this system uses and describes in detail the different problems addressed in this thesis.

Chapter 2

Personal knowledge

In this chapter, we describe a model for *personal knowledge* that we use throughout this thesis. We begin by describing the abstract nature of personal knowledge, its different dimensions, and what it represents (Section 2.1). Then we present an ontology for personal knowledge (Section 2.2). We conclude the chapter by introducing the different points addressed in this thesis involving personal knowledge (Section 2.3).

2.1 The nature of personal knowledge

The Five Ws and one H

Journalism students have long been taught a principle, known as the *Five Ws*, by which information about a story should be collected (Flint 1917). By this principle, a story can only be considered complete if it provides an answer to the following questions: what happened? who is involved? where did it take place? when did it take place? and why did it happen? A sixth question is sometimes mentioned: how did it happen? As highlighted by Vianna et al. (2014), answers to these questions can help and guide Alice when searching for information that she remembers having stored and accessed in the past. For instance, when searching for an important document that she remembers having seen not so long ago, she might recall and exploit information about how she came across this document (read in an inbound email message), who is involved (Bob sent it to Alice), what it is about (project Charlie), or when she saw it (last month). These pieces of information provide hints to a system helping Alice find this document. In the rest of this section, we describe the nature of personal knowledge with respect to the dimensions captured by the Five Ws.

An extension to a person's memory

In Section 1.4, we mentioned the vision of a PIMS as a supplement to the user's memory. Following such a vision, we view personal knowledge as extending the memory of Alice. More specifically, it extends her *explicit memory*, which is a type of *long-term memory* (Atkinson and Shiffrin 1968) responsible for keeping and

consciously recalling facts and events. Explicit memory can be further decomposed into *episodic memory* and *semantic memory* (Tulving 1972). Analogously, we view personal knowledge as being composed of two parts — *episodic knowledge* and *semantic knowledge* — respectively extending the two aspects of Alice's explicit memory.

Episodic knowledge

Alice's episodic memory represents her memory of personal experiences and autobiographical events that occurred at particular moments in her life. Episodic knowledge intends to capture and extend this memory. By the definition given in Section 1.1, information about such experiences and events is personal. A great source of event information is Alice's calendar. Different from her episodic memory however, her calendar mostly contains information about planned events: she uses her calendar to plan ahead how she spends her time. Planned events may be past or future and may or may not end up being realized. Experienced events, on the other hand, are information about past events that did in fact happen. An example of a planned event is a doctor appointment scheduled for the next day. An experienced event may for instance be the knowledge, available the next day, of whether Alice attended this doctor appointment, if she was late to it or not. A planned event can thus be linked to an experienced event once it realizes. However, not all of Alice's personal experiences can be planned: she might spontaneously stop at a coffee shop on her way to her appointment. Additionally, if a planned event does not end up being realized (e.g., if Alice misses her appointment), knowledge about non realization is a form of experienced event. While experienced events more accurately reflect Alice's episodic memory, planned events are useful for planning and prediction tasks, which are important for managing one's life and are at the heart of many information management applications. Finally, both kinds of events have a similar structure. For these reasons, we define episodic knowledge as the representation of both experienced and planned *events*.

In the context of the Five Ws and one H, events are particular in that they invariably capture the *when* dimension. In the case of experienced events, the *where* is very often captured, as Alice is always somewhere, but this information may not always be known to Alice or available to the PIM system. Alice is always among the persons involved in an experienced event, i.e., the *who*. *What* an event is about varies a lot, e.g.: the user spent $30 (in a supermarket) to buy some groceries, she has to attend a work meeting, she was out jogging (along the riverbank), or she has to pick her kids up (from school). The most difficult dimension to capture is the *why*, which would actually allow us to understand why for instance Alice had to leave a work meeting early: to pick up her kids.

Semantic knowledge

Alice's semantic memory represents a record of facts, meanings, and concepts that she has acquired over her lifetime. Semantic memory refers to factual knowledge that holds independent of any particular personal experience or event in which

she may have acquired it. Her and her friends' birthdates, the place where she works at, and the kind of music that she likes are examples of semantic memory. Semantic knowledge intends to capture and extend this memory. Parts of this knowledge are immutable (e.g., her birthdate) and some others may change over time (e.g., her current workplace). Such changing knowledge can also be captured by episodic knowledge of events spanning long periods of times (e.g., her workplace over the years). Semantic knowledge is an important concept of personal knowledge as it helps defining and understanding the contents of events, e.g.: what do we know about the event's organizer? how far is the event from Alice's workplace? or what is so particular about the event's date (Bob's birthday)? By doing so, semantic knowledge clarifies the connections between the elements of Alice's episodic knowledge (e.g., Bob's birthday parties over the years).

Observed vs. inferred knowledge

We previously put forward the idea of personal knowledge as an extension to Alice's explicit memory. However, we must make clear that a PIMS, which is an agent that is external to Alice, cannot read Alice's mind, and thus cannot replicate the exact contents of her memory. To model this mismatch, we distinguish in the context of an external agent between *observed* and *inferred* knowledge. Observed knowledge refers to knowledge that the agent can observe directly from a trusted source, e.g., a birthday marked in Alice's calendar. Inferred knowledge is knowledge deduced by the external agent from observed knowledge. Episodic knowledge can be inferred this way, in particular knowledge about experienced events. For instance, knowledge that Alice is currently at work can be inferred from the observation that her location, as given by the GPS of her mobile device, comes close to her workplace. An external agent can also infer semantic knowledge, for example from the analysis of episodic knowledge. For instance, Alice's preferences can be deduced from repeated occurrences of similar events: if Alice listens to a lot of rock music, the agent could deduce that she has a preference for the genre. Or, the fact that she loves this particular fish restaurant can be inferred from the many times she's been there. In the model that we use to represent personal knowledge and that we describe in the next section, *provenance* is used to distinguish observed from inferred knowledge.

2.2 A model for personal knowledge representation

In this section, we describe the model that we use to represent personal knowledge. This model intends to capture both Alice's episodic and semantic personal knowledge, and distinguishes, from the point of view of an external agent (the PIMS), between observed and inferred knowledge. For knowledge representation, we use the Resource Description Framework (RDF) standard (Carroll and Klyne 2004; Wood, Cyganiak, and Lanthaler 2014), which we describe next. After that,

we present the *personal knowledge ontology*, an ontology that we designed to represent personal knowledge.

The Resource Description Framework

Knowledge representation and reasoning is concerned with how knowledge about a domain can be formally represented and used by a machine for reasoning about this domain (Levesque 1986) The concepts and relationships captured by a knowledge model are formally defined in an *ontology*. In following the vision of the Semantic Web (Berners-Lee, Hendler, and Lassila 2001), the World Wide Web Consortium (W3C) has proposed standards for sharing knowledge and ontologies on the Web, most fundamentally RDF.

RDF uses *Internationalized Resource Identifiers* (IRIs) to identify *entities*, e.g., physical things, documents, and abstract concepts. (The IRI specification is an extension to the *Uniform Resource Identifier* (URI) specification that allows greater set of characters.) For example, Wikidata (Vrandečić and Krötzsch 2014), a collaboratively edited knowledge base, uses the IRI `http://www.wikidata.org/entity/Q76` to identify the *Barack Obama* entity. To abbreviate IRIs, it is common to define *namespace prefixes*. A namespace prefix is the abbreviation of a *namespace IRI*. For instance, Wikidata uses `wd` as the namespace prefix for `http://www.wikidata.org/entity/`. The IRI of the *Barack Obama* entity can then be written as `wd:Q76`. To represent values such as strings, numbers, or dates, RDF uses *literals*. A literal consists of a *lexical form*, i.e., a string, and a *datatype IRI*, i.e., the identifier of a *datatype* that determines how to map the lexical form to a value. RDF reuses the datatypes defined in XML Schema (Biron and Malhotra 2004; Peterson et al. 2012), e.g., `xsd:string`, `xsd:integer`, and `xsd:date` (`xsd` is the conventional namespace prefix for `https://www.w3.org/2001/XMLSchema#`). Some examples of literals are (`"John Doe"`, `xsd:string`), `"42"`, `xsd:integer`), and (`"1990-01-01"`, `xsd:date`). A *blank node*, which is neither an IRI nor a literal, is an identifier that serves as a local existentially quantified variable. IRIs, blank nodes, and literals are called *resources*. In RDF, a *statement* (or *triple*) consists of a *subject*, which is an IRI or a blank node, a *predicate*, which is an IRI, and an *object*, which is a resource. The predicate denotes a *property*, i.e., a resource that can be thought as a binary relation. A RDF statement is written as $\langle s, p, o \rangle$, where s is the subject, p the predicate, and o the object. To assert $\langle s, p, o \rangle$ is to say that the relationship p holds between the resources s and o. For example, assuming that the `example:familyName` represents the *family name* property (where `example` is the namespace prefix for `http://example.org/`), the family name of Barack Obama can be stated as: $\langle$`wd:Q76`, `example:familyName`, (`"Obama"`, `xsd:string`)$\rangle$, An *RDF graph* is a set of statements. A *named graph* is an RDF graph that has an associated IRI or blank node (the *graph name*).

RDF Schema RDF Schema (RDFS) (R. Guha and Brickley 2004; Brickley and R. Guha 2014) enriches RDF by providing basic elements for the description of ontologies (also called *vocabularies*), including constructs for the definition of classes, class hierarchies, and property constraints. RDFS con-

structs are based on RDF resources that are conventionally abbreviated using `rdf` and `rdfs` as the namespace prefixes for `http://www.w3.org/1999/02/22-rdf-syntax-ns#` and `http://www.w3.org/2000/01/rdf-schema#`, respectively. In RDFS, a *class* is a collection of resources such as *Person* and *City*. The members of a class are known as the *instances* of the class. Classes are themselves resources and are often identified by IRIs. For instance, *Person* may be identified by `example:Person`. Classes are instances of the class `rdfs:Class`. The `rdf:type` property is used to state that an IRI or a blank node is an instance of a class, as in ⟨`schema:Person`, `rdf:type`, `rdfs:Class`⟩ and ⟨`wd:Q76`, `rdf:type`, `schema:Person`⟩. The property `rdfs:subClassOf` is used to state that a class is a subclass of another class, i.e., that any instance of the first class is an instance of the second one, as in ⟨`schema:Person`, `rdfs:subClassOf`, `schema:Thing`⟩. Properties are instances of the class `rdf:Property` and literals are instances of the class `rdfs:Literal`. Datatypes, assimilated to their IRIs, are both classes and instances of the class `rdfs:Datatype`. Every literal is an instance of its datatype. For example, `xsd:integer` is an instance of `rdfs:Class` and `rdfs:Datatype`, and the literal (`"42"`, `xsd:integer`) is an instance of `xsd:integer`. Each instance of `rdfs:Datatype` is a subclass of `rdfs:Literal`.

The personal knowledge ontology

For representing personal knowledge, we use the `schema.org` vocabulary wherever possible. This vocabulary is supported by Google, Yandex, Microsoft, and Yahoo, and was initially developed as a set of schemas "for structured data markup on web pages" (Seth 2011). `Schema.org` uses `schema` as the namespace prefix for `https://schema.org`. Wherever this vocabulary is not fine-grained enough for personal knowledge representation, we complement it with a vocabulary that we designed, which lives in the namespace `http://thymeflow.com/personal#` associated with the prefix `personal`.

The *personal knowledge ontology* is the resulting ontology. Next, we describe the concepts captured by this ontology and put them into perspective with respect to the different dimensions of personal knowledge that they capture: *what*, *who*, *where*, and *when*. Figure 2.1 illustrates the classes and properties in this ontology.

Agents: persons and organizations

An important dimension of personal knowledge is the *who*. The `schema:Person` and `schema:Organization` classes are used to model this dimension. Instances of these classes represent the persons and organizations that Alice knows, has heard of, or has interacted with. Alice interacts with persons including her family, friends, colleagues, and all kinds of acquaintances in general, e.g., clients, her car dealer, her hairdresser, or her lawyer. She interacts with organizations either directly, e.g., when she receives a quote from her car dealer John, or indirectly, e.g., when she visits the website of the dealership business that employs John. These interactions are associated with episodic memories. To express the relationship between John and his business, we use the `schema:affiliation` relation. When

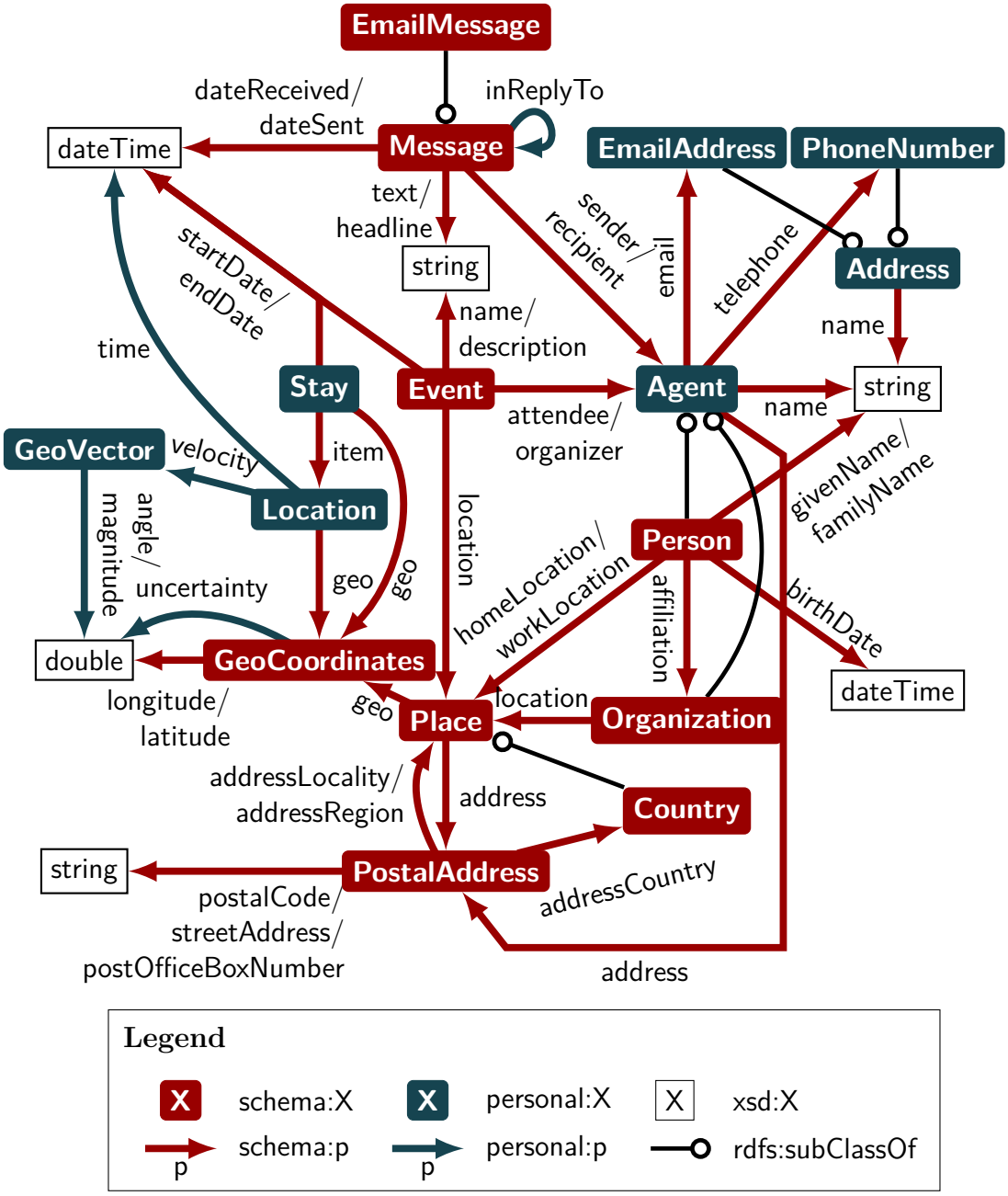


Figure 2.1: The personal knowledge ontology. Rounded nodes are non-literal classes while rectangular ones are literal classes (datatypes). An arrow from a non-literal class X to a class Y with label p means that the object of a statement whose predicate is p and whose subject is an instance of X is an instance of Y. A line between two non-literal classes X and Y with a circle near Y means that X is a subclass of Y.

it is not known whether an entity represents an individual or an organization, we use the class `personal:Agent`, which subsumes the `schema:Person` and `schema:Organization` classes. The name of an agent is represented via the `schema:name` property. The given name, family name, and birth date of a person are represented via the `schema:givenName`, `schema:familyName`, and `schema:birthDate` properties.

Places

For modeling the *where* dimension of personal knowledge, *places* are used (`schema:Place`). Simple places are defined either by a postal address, via the `schema:address` property, or by a geographic point, via the `schema:geo` property. A postal address usually has a country, a locality, a region, a postal code, and a street address. Countries, localities, and regions are places themselves that are associated with a geographic area and are organized in an inclusion hierarchy (via the `schema:containsPlace` relation). A geographic point is defined by a set of geographic coordinates (`schema:GeoCoordinates`): a longitude (`schema:longitude`) and a latitude (`schema:latitude`). We use the geo URI standard (Mayrhofer and Spanring 2010) to identify geographic coordinates. Agents are associated with places via the `schema:location`, `schema:homeLocation`, and `schema:workLocation` relations.

Events

The `schema:Event` class is used to represent the experienced and planned events that belong to Alice's episodic knowledge. The properties of an event partially answer the Five Ws: `schema:name` and `schema:description` define the *what*, `schema:location` defines the *where*, `schema:startDate` and `schema:endDate` define the *when*, and `schema:attendee` and `schema:organizer` describe the *who*. The *who* is of course represented by an instance of `personal:Agent` and the *where* is represented by an instance of `schema:Place`. To represent the *when*, we use the standard XML Schema `xsd:dateTime` class. The name and description of an event are typically text strings.

Messages

For describing communications between agents, the `Message` class is used. Messages are associated with single points in time via the `schema:dateSent` and `schema:dateReceived` properties (the *when*).

The `schema:sender` and `schema:recipient` relations are used to represent the `who` of a message. The object of a statement of such relation is an agent. Technically, however, the sender and recipients of a message are the *addresses* to which the message is delivered. An address identifies the medium via and the location to which a message is delivered, e.g., an email address for an email box, a telephone number for a telephone line, or a postal address for a mailbox. An agent may own one or multiple agents. Using addresses (`personal:Address`) instead of agents to represent the sender and recipients of a message would thus have

been a legitimate alternative. Nevertheless, for modeling personal knowledge, we prefer the use of agents for this purpose as a message is ultimately addressed to a person or an organization. The `schema.org` ontology has made this choice as well, which brings *messaging* closer to the semantics of `schema:Action` that is used for expressing general interactions between agents. In this direction, it should be noted that the addresses used for email or mail usually include enough information to identify a person or an organization, which is necessary for avoid ambiguity, as single mail or email box may be shared among multiple individuals of the same family or organization. However, using agents instead of addresses to represent the sender and recipients of a message makes it impossible to express the address to which the message was sent unless such agents are restricted to having only one address. The address may contain important information about a message, e.g., knowing that a message was sent to Alice's professional address may tell us that the message is about work. This is why the personal knowledge ontology enforces this restriction. To represent a real-world person or organization that is reachable via multiple addresses, multiple `personal:Agent` instances are used. As we see later in this section, these instances are said to be *facets* of the same real-world entity. The address of an agent is represented via the `schema:telephone`, `schema:email`, and `schema:address` relations, whose objects are respectively instances of the `personal:PhoneNumber`, `personal:EmailAddress`, and `schema:PostalAddress` classes.

Messages also have a subject and a body, which are represented via the `schema:headline` and `schema:text` properties, respectively. For representing conversations or threads, the `personal:inReplyTo` relation is used. The statement $\langle s, \texttt{personal:inReplyTo}, o\rangle$ says that the message s is a reply to the message o.

Locations and Stays

Places are useful for representing the location of an event or the locations that are important to a person or an organization (e.g., the person's home or the organization's offices). For representing the geographical position of Alice over time (i.e., Alice's trajectory), the `personal:Location` class is used. Each instance of `personal:Location` consists of a time, represented via the `personal:time` property, and a geographic point, represented via the `schema:geo` property. Since Alice's location may only be known up to some uncertainty, the geographic point of a location is often associated with some confidence interval, which is defined by a radius in meters and is represented via the `personal:uncertainty` property. The geo URI specification used to define the IRIs of geographic coordinates conveniently includes an uncertainty parameter. Finally, a period of time during which Alice remained in the same place (e.g., a restaurant, the gym, or her office) is called a *stay*. Stays are instances of the `personal:Stay` class. A *stay* consists of a geographic point and a period of time. This period of time is represented by start and end times (`schema:startDate` and `schema:endDate`). Instances of `personal:Location` that correspond to a stay are associated with this stay via the `schema:item` relation (in such a statement, the stay is the subject and the location is the object).

Facets

In order to express that a group of instances represent the same real-world entity, the `personal:sameAs` equivalence relation is used. Agents, events, places, or messages that belong to the same equivalence class are called *facets* of the real-world entity that they represent. The `personal:sameAs` relation allows the representation of co-reference between different sources of knowledge. For instance, it may happen that Bob, a friend of Alice, appears both on the address book of Alice's mobile phone and on Alice's friend list in some social networking site. These two references of Bob, originating from different sources, may be represented in RDF as two distinct resources (IRIs). To express that these two resources refer to the person, the `personal:sameAs` relation is used to link one to the other. The representation of facets allows some flexibility that is useful when the knowledge of co-reference is uncertain and may change over time. In addition, as discussed during the description of the representation of messages, facets can be used to represent an agent entity that owns multiple addresses when, for modeling reasons, a single agent facet is only allowed to have one address.

Personal knowledge base

In our context, a *knowledge base* (KB) is a set of RDF named graphs. The *personal knowledge base* of the user is a knowledge base that represents the user's personal knowledge from the point of view of an external agent (the PIMS) using the personal knowledge ontology. In this knowledge base, named graphs are used to represent provenance. Each such graph represents a single unit of knowledge, i.e., the smallest collection of statements to be associated with a single provenance. We call *document* such a collection. The name of this document, i.e., the name of the graph that this document represents, is an instance of the `personal:Document` class. This instance is associated with the documents source via the `personal:documentOf` relation, whose object is an instance of `personal:Source`. To distinguish between observed and inferred knowledge, the `personal:ObservationSource` and `personal:InferenceSource` classes are used, which are subclasses of `personal:Source`.

2.3 Goals of this thesis

In this section, we introduce the specific tasks addressed in this thesis. As mentioned in Section 1.4, these tasks are generally concerned with collecting and enriching personal information.

Knowledge integration and querying

In building a knowledge base, one important task is to merge information from multiple sources with different schemas and representation models. The representation of personal information may vary from one source to another in the following ways:

- Different concepts: Alice's contacts in a business-oriented social networking site are her "business connections", her contacts in a general social networking site are her "friends", and her contacts in an online dating site are her "possible dates".

- Different contexts: Alice may use an alias instead of her real name for her social network profile (e.g., for privacy reasons), but use her real name when sending email messages.

- Different granularity and structure: An email message that describes an event contains mostly unstructured (textual) information about this event, whereas a calendar application may use fields such as name, date, and location to describe this event.

- Different channels: In order to store a note or document, Alice may send an email to herself or use her note-taking application.

Each source may exhibit various degrees of uncertainty and redundancy within itself, due for the instance to:

- Duplicates: Alice's address book may contain duplicate entries for the same person (e.g., entries share the person's name but have different telephone numbers).

- Inaccuracies: The times for scheduled events are mostly inaccurate since they are estimates of future things to happen. Also, for reasons of simplicity, applications may be biased or limited in their representation of information. For instance, times are usually set in 30-minute increments in calendar applications and most of calendar applications require that users define a planned end date for each future event. For some events, such as a dinner appointment, Alice may not be able to provide an accurate end date in advance.

- Incompleteness: not all of Alice's real-life friends have an account on the social networking service that she uses.

- Errors: The names of contacts that Alice manually entered in her address book may contain typographical errors (e.g., "Johnson" instead of "Johnsson").

- Concealment: For privacy reasons, Alice may provide incorrect information on purpose to services that require them (e.g., her birth date or address).

- Outdatedness: She may also have responded "Going" to an event to which she never went.

- Incoherence: Alice may have signed up for two different events happening at the same time in distant places. Since it is clear that she may not possibly attend both, such information is incoherent.

When gathering information from multiple sources without reconciling them, duplicates naturally appear. For instance, there may be a contact entry for Bob in Alice's address book and in Alice's social networking site. In presence of uncertainty in the sources, errors in the sources may become incoherences. For instance, a typographical error in the name of Bob in Alice's address book may be incoherent with the name that appears on Bob's social network profile.

We could acknowledge the presence of uncertainty and representational differences in sources of personal information by using a personal information model in which knowledge is represented as close as possible to information sources: In such a model, we could state for instance that there exists an agent called "Bob" in source A and an agent called "Robert" in source B, and that we have more confidence in source B than in source A. However, with such a representation of knowledge, Alice would not be able to get an answer to queries that need the reconciliation of information from different sources: "What is the most recent message received from Bob?" Being capable of expressing that "Bob", as he calls himself on his social network profile, and "Robert", as he is known in email communications, represent the same person is necessary for properly answering this query. In our setting, we would like to integrate information into a representation model in which agents, events, messages, places, and locations representing the same real-world entities are connected as much as possible, as the user, Alice, would see them. In this model, it is not contradictory that Bob, Alice's friend, may be represented as both "Bob" and "Robert" in different sources. They can be seen as two distinct *facets* of the real-world entity Bob (cf. Section 2.2). When the connection between these facets is not explicit in the sources (e.g., via a common identifier), automatically identifying equivalent facets is necessary for reconciling these sources. Similarly, a calendar event about a dinner at *L'Arpège* at 7 p.m. could be automatically linked to a *stay*, derived from Alice's location history, representing the fact that Alice spent some time near that restaurant around that hour. In both situations, there is a certain degree of uncertainty associated with automatically inferring these connections. Though also important, we do not consider in this thesis the explicit general modeling of uncertainty within the knowledge base. Instead, to simplify, the inference tasks that we consider model uncertainty at the information level and output a non-probabilistic representation at the knowledge level. In our context, we denote by *knowledge integration* the process of merging information from different sources into a *personal knowledge base*, which includes identifying facets of the same real-world entity, such as described.

Another important task in building a personal knowledge base is querying it. The user should be able to query the knowledge base within and across its dimensions and aspects: data types (e.g., numeric, date, and text), entity types (e.g., agents, events, messages, stays, and places), provenance, facets, and real-world entities. For instance, Alice should be able to query real-world entities instead of facets, so as to retrieve all the messages from her friend Bob regardless of the communication medium.

Related work Providing a unified view of the data residing in different sources by using a mapping from queries over a global schema to queries over the source schemas, defined in some formal logic, is called *data integration*, which is a well known subset of database theory (Abiteboul, Manolescu, et al. 2011). When an entity may be represented in two different sources without a common identifier, *data matching* has to be performed prior to data integration. Data matching (also known as *record linkage*, *data linkage*, *entity resolution*, *object matching*, or *field matching*) is the task of identifying records that refer to the same entity across different sources (Christen 2012). Data matching may involve the use of a probabilistic model to represent the probability that two records represent the same entity. It is extensively utilized in data mining projects and in large-scale information systems by business, public bodies, and governments. Example application areas include national censuses, the health sector, fraud detection, online shopping, and genealogy (Christen 2012). More generally, when sources contain unstructured or semi-structured data and when information may be contained within text (e.g., the date of an event within the body of an email message) or rich media (e.g., the persons in a photo), the task of merging information from these sources is called *information integration*. In particular, *information extraction* (Cowie and Lehnert 1996) denotes the set of techniques to extract information from unstructured resources. Information integration, which is a superset of data matching, aims at reducing redundancy and uncertainty when combining information.

Knowledge integration can be understood as being one level above information integration, in accordance with the DIKW hierarchy's interpretations of knowledge and information (cf. Section 1.1). Knowledge integration is the process of deriving a common knowledge representation from different sources of information. When information sources are also knowledge sources (i.e., each source has its own ontology), data integration techniques can leverage this formal representation via *ontology-based integration* (Wache et al. 2001). The process of finding automatic correspondences between concepts in two ontologies is known as *ontology alignment* (Euzenat and Shvaiko 2013). Knowledge integration can also be seen as incorporating information into a body of existing knowledge that is larger in scope, possibly universal. In this direction, Berners-Lee (2006) outlined a set of best practices for publishing knowledge that is linked to or can be linked to from knowledge published by others. *Linked data* is the term used to refer to this publishing method. It is based on RDF for knowledge representation and the use of *dereferenceable* HTTP URIs to identify resources (i.e., entities, concepts, and relations). Issuing an HTTP GET request to the URI that identifies a resource should provide information about this resource (including links to related resources) in RDF. These practices have been widely adopted by an increasing number of providers, effectively forming a knowledge base out of a set of distributed heterogeneous sources (Bizer, Heath, and Berners-Lee 2009) — the *Web of data*. For querying RDF, SPARQL is the language of choice (Harris and Seaborne 2013).

In this thesis In Chapter 6, we present the description and the evaluation of a system that we designed and implemented to perform *personal knowledge integration* from different sources of personal information: the user's email messages, address books, calendars, and location history. The retrieval and transformation of information from each of these sources into *personal knowledge* (in RDF) is described in Chapter 3. The system integrates them into a *personal knowledge base* that Alice can query within and across dimensions. This integration involves entity resolution of agents and spatiotemporal alignment between events and stays.

Inferring a timeline of user activity

An important aspect of Alice's personal knowledge base is its ability to supplement her episodic memory (cf. Section 2.1). To this end, the knowledge base needs to maintain a history of Alice's activities, represented as *experienced* events. Information about Alice's activities can for instance be retrieved from her electronic diary, where she reports on her daily experiences.

Today, many applications allow Alice to keep a private electronic diary, but some users prefer to make their diaries public by writing a blog, as a form of social activity (Nardi, Schiano, and Gumbrecht 2004). Social networking sites, where users share their everyday experiences with a smaller group of people, fit somewhat in the middle. An alternative to diary-like sources for obtaining information about Alice's activities is any source that can provide systematic observations of her actions and her environment over time: where is she now? what is she doing? who is she with?, etc. In fact, these sources are complementary: While analysis of self-reported experiences is insightful, it has also been recognized that "diary studies", a research method where participants of an experiment are asked to write about their experiences and activities on a diary, benefits from objective systematic measures and vice versa (Bolger, Davis, and Rafaeli 2003).

As discussed in Section 1.2, observations of her actions and her environment can be produced either actively or passively. When Alice interacts with a machine that keeps a log of what it does, she passively produces a history of actions: she turned on her computer at 8:03 a.m., she sent an email at 11:40 a.m., she ordered a book online at 2:32 p.m., she entered in her calendar at 4:56 p.m. a new event scheduled for tomorrow. In doing so, she also actively produces the *what* of these experiences, when applicable: the email's content, the book's name, the event's description. Many machine applications do in fact keep logs, or at the very least record the creation and latest modification times of editable content. However, these logs do not necessarily tell anything about Alice's environment (e.g., who is she with? or is she in a business meeting?) nor about actions that do not involve interacting with a machine (e.g., is she on the way to work?).

Another way of obtaining information about Alice's actions and environment is through lifelogging, which lets Alice record the world that she sees using a wearable camera (cf. Section 1.4). Though lifelogging is directly useful for reminiscing and recollecting, the stream of images that it produces can also be automatically analyzed to infer a history of Alice's activities (Doherty, Moulin,

and Smeaton 2011; Doherty, Caprani, et al. 2011). However, such practice is costly. For instance, Gordon Bell, a lifelogging pioneer who started wearing a camera in 2000, recently stopped wearing his camera on the grounds that lifelogging "wasn't something that was bringing a lot of value to [his] life" (Regalado 2016). For Bell, the information collected by the sensors in Alice's smartphone, smartwatch, and fitness wearables (e.g., location and activity trackers), as well as what she posts on social media together constitute a lifelog.

One of our goals is on recreating Alice's history of visited places and itineraries followed from mobile device sensor data. The first goal is to be able to segment her day into two kinds of events: *stays* and *moves*. *Stays* are the moments and places where Alice has remained for some amount of time (cf. Section 2.2), and *moves* are the moments in-between. A *move* usually corresponds to a trip to go from one place to another, but it might also correspond to richer outdoor activity (e.g., jogging and sightseeing). The second goal is to describe for each move the *multimodal itinerary* that the user followed. The goal is to identify the different transportation modes used by Alice during each *move* for modes such as foot, bicycle, car, bus, train, and metro, and identify, for public transportation modes, the public transportation routes taken by Alice.

Related work Commercial applications already exist that collect and analyze mobile device sensor data in order to show Alice a history of her stays and moves: Moves (ProtoGeo 2013), SAGA (A.R.O., Inc. 2011), and Google Timeline (Google 2015b). However, they do not always distinguish among car, bus, and train, and when they do, they do not identify the public transportation route taken by Alice. These applications are in part the result of the rich location (e.g., satellite-based, Wi-Fi-based, and cellular-based positioning) and movement tracking (e.g., accelerometer) sensors of a mobile device combined with *activity recognition* techniques, which we discuss in Section 5.8.

In this thesis In Chapter 4, we describe how automatic *stay/move* segmentation can be performed. In Chapter 5, we show how multimodal itinerary recognition can be performed. Finally, in Chapter 6, we show that it is possible to connect the knowledge extracted from this segmentation by connecting stays to possible events in Alice's calendar, which results in an activity timeline that is connected to all of the dimensions of her personal knowledge base.

Synchronization of personal information

Information synchronization is the process of establishing and maintaining consistency among information between two or more sources of information. An example is the synchronization between the contacts in Alice's smartphone and those in her personal computer. Assuming that each contact possesses an identifier and that two contacts representing the same individual share a common identifier, a synchronization process can begin by finding the contacts in both sources with identifiers in common and merging their attributes. Then the contacts in one source that do not exist in the other (i.e., whose identifiers are not present in

this other source) are transferred to this other source and vice versa. A conflict occurs when a contact appears in both sources with different values for some attribute. In that case, it must be resolved, by either requiring user intervention (e.g., by asking Alice what to do) or using a predefined fully automatic rule (e.g., by keeping the attribute that was modified last, if this information is available). Once consistency has been established, the synchronization process can maintain it over time by propagating into one source each change that happens in the other. During this propagation, conflicts can however also happen, for instance if an attribute is modified in both sources before they can synchronize. Such a conflict may be resolved using the same rules as before. Propagating changes as soon as they happen alleviates the appearance of conflicts, which works well assuming that changes can be propagated and acknowledged faster than the rate at which information changes in each source.

When entries or instances (e.g., contacts) representing the same entity (e.g., individual) may appear with different identifiers in different sources, synchronization may lead to the creation of duplicates in each source. To prevent this, data matching should be performed before attempting synchronization.

We distinguish two kinds of information synchronization: one-way and two-way. In one-way information synchronization, changes are unidirectionally propagated from a source to a target information system. No changes in the target are ever propagated to the source. In two-way information synchronization, changes propagate both ways.

Related work Multiple information synchronization techniques have been successfully applied to different kinds of applications: file systems (Tridgell and Mackerras 1996), databases (Wiesmann et al. 2000), address books (Hansmann et al. 2002), and cloud storage (Drago et al. 2012). The use of multiple devices and systems by a single individual (cf. Section 1.2), which usually entails the replication of personal information, has made information synchronization an important aspect of personal information management. However, the centralized management of data, where only one information system, i.e., a central server, holds an exact and complete copy of the data and where content is accessed by thin clients (with no storage) and edited by synchronously pushing each modification to the central system, simplifies the requirements of information synchronization. Examples of such management include email or file storage web applications that Alice can use to consult her email or browse her documents without having to download a complete copy of the data to her device. Nevertheless, web application technology has also facilitated the development of *cloud collaboration* tools, which allow multiple users to co-author documents on the Web in near real-time, and have led to the development of new automatic conflict resolution algorithms (Sun and Ellis 1998; Fraser 2009).

In this thesis The personal knowledge management system that we present in Chapter 6 integrates different sources of personal information. We show how the system's knowledge base is kept synchronized by propagating changes in any of the sources into the knowledge base, while inferring possibly new knowledge

relative and according to these changes. Conversely, we show how changes in the knowledge base are propagated back into the sources. Finally, we show how the integration and two-way synchronization realized by this system can used to keep multiples sources of the same kind (e.g., the set of all address books) synchronized.

Personal analytics

Analytics is the discovery, interpretation, and communication of meaningful patterns in data. It uses statistical, machine learning, and data visualization techniques. Coined by Stephen Wolfram, *personal analytics* refers to applying analytics to the data of an individual for the individual's own benefit (Regalado 2013). For Alice, it is a way to explore and gain insight into matters such as: what can I do to be more active? how does the food I eat impact my mood? is weather linked to my productivity at work? and what aspects of my day correlate with good sleep? For performing personal analytics, personal knowledge integration is useful, as the quality of analytics depends on the coherence of the input information. Additionally, analytics relies on being able to query the data efficiently within and across dimensions.

A related task is the *prediction* of Alice's future events by analyzing everything that is currently known, including the history of past events. More generally, *predictive analytics* refers to making guesses about unknown events, whether past, current, or future (e.g., did Alice attend yesterday's event?). Since the knowledge integration and user activity inference tasks are already concerned with past and current events, we assimilate *prediction* to the foreseeing task, which does not have access to current observations of the predicted reality (e.g., Alice's location when the event supposedly takes place).

Related work Wolfram (2012) uses personal analytics to gain insight about his life using email, calendar, phone call, keystroke, file, and pedometer data. He learns how the changes that occurred throughout his life had an impact on his activities, and learns how "shockingly regular" many aspects of his life have been.

Quantified self applications usually involve some analytics applied to the data that they collect (e.g., activity, health, spending, mood, time management, or communication information), for instance by showing the user aggregate data over some period of time. However, more interesting are those analytics that combine multiple sources of information. Exist (Cooper and Sharp 2017) is a commercial application that provides analytics from multiple sources of personal information, including calendars, activity trackers, and social networking services. It finds correlations such as "Your weight is higher after you check-in to *this one restaurant* more". An open-source application that does similar things is Fluxtream (Wright 2016).

In this thesis In Section 6.5, we show some basic examples of personal analytics that are based on querying Alice's personal knowledge base. The good level

integration provided by the knowledge base combined with its querying capabilities make such queries possible.

We do not show in this thesis examples of predictive analytics. One thing that we experimented with was to guess the location of a future calendar event using the contents of its different other attributes (e.g., the event's time, description, and participants), however, our attempts did not yield results better than a blind guess using the datasets at our disposal.

2.4 Conclusion

In this chapter, we presented the *personal knowledge ontology* and introduced in particular the concepts of *stays*, *facets*, as well as *observed* and *inferred* knowledge. Stays, which may be derived from the user's location history, are particularly important when inferring a timeline of user activity (Chapters 4 and 5), while facets and both observed and inferred knowledge are important concepts in knowledge integration (Chapter 6). The next chapter describes the different sources of personal information that are used throughout this thesis and their representation using the personal knowledge ontology.

Chapter 3

From data to personal knowledge

In this chapter, we present the different sources of personal data specifically considered in this thesis, describe the information they hold, discuss their significance, and describe the process by which they are converted into personal knowledge (cf. Section 2.2).

We begin by describing standard sources of personal information: email messages, address books, and calendars. For these sources, there exist standard protocols and formats whose usage is widespread. Then we describe a particular social networking service that provides a non-standard web API to access user data. Finally, we consider mobile device sensors, for which we developed an application to record and retrieve different kinds of data.

3.1 Email messages

Electronic mail, or email, is a standardized method for exchanging messages between computer users. The email system relies on a decentralized network of servers: mail submission agents, mail transfer agents, and mail delivery agents. These servers exchange messages using the Simple Mail Transfer Protocol (SMTP), which specifies how a submitted email message should be processed in order to reach its intended destination. The recipient of such a message is specified by an email address. An email address (e.g., `alice@example.com`) identifies an email box and is defined by its *local-part* (e.g., `alice`) and its domain (e.g., `example.com`) An email box is usually associated with an individual (its user), who can access its contents (e.g., incoming messages) by authenticating to the mail server responsible for her email box. The email system relies on the Domain Name System (DNS) (Mockapetris 1987) to find the mail server responsible for accepting email messages on behalf of a recipient's domain. The email system is among the most successful and frequently used computer applications, and for many people the main channel for distributing information (Whittaker, Bellotti, and Cwizdka 2011). For this reason, it is essential for personal information management. In the implementation that we present in Chapter 6, email messages are the only kind of `schema:Message` we consider. Other messaging systems such as text and instant messaging are similar in content but less standard in general and much harder to retrieve.

The format of email messages is specified by the RFC 5322 (Resnick 2008). A message consists of a header and an optional body. The header is structured into fields, and each field has a name and a value. Header fields contain information such as the sender (the `From:` field), the recipients (`To:`), the subject (`Subject:`), and the creation date (`Date:`) of the message. The body of a message supports many formats, for both text (e.g., plain text and HTML) and binary content (e.g., media or applications). A globally unique identifier is usually attributed to an email message by its sender, which stores it in the *Message-ID:* header field. They are used to prevent multiple delivery and for referencing a message from another. For instance, when replying to a message, the *In-Reply-To:* header field should be filled with the *Message-ID* of the message this is a reply to. Email messages are particular in that they can include different kinds of recipients, the carbon copy recipients (*Cc:*), and the blind carbon copy recipients (*Bcc:*). We use the `personal:primaryRecipient`, `personal:copyRecipient` and `personal:blindCopyRecipient` properties, which are subproperties of `schema:recipient`, to distinguish them.

As discussed in Section 2.2, the sender or a recipient of an email message does not directly refer to a person. Instead, it refers to an email address, which identifies an email box. Each correspondent present in a *From:*, *To:*, *Cc:*, and *Bcc:* header field has an email address and an optional display-name (e.g., `bob@example.com` or `Bob <bob@example.com>`). These fields are important sources of knowledge. A simple email address such as `jane.doe@inria.fr` implicitly provides the given and family names of a person ("Jane Doe"), as well as her affiliation ("Inria"). Performing DNS and WHOIS (Daigle 2004) lookups on the domain or IP address of the domain of the email address can yield extra information about an organization, such as its full name, address, and country of origin. However, some email addresses provide by themselves little knowledge about its owner and some almost none, like for instance `j4569@gmail.com`, which is managed by a public email service provider. When a display-name is provided, as in `Jane Doe <j4569@gmail.com>`, the name of the person is usually revealed. Sometimes, the display-name contains some extra information, as in `Jane Doe - Human Resources <jane.doe@inria.fr>`. In this thesis, we do not consider the extraction of such additional information. Addresses representing organizations may include for instance `edbt-school-2013@imag.fr` or `fancy_pizza@gmail.com`. The correspondents of an email message are represented as `personal:Agent` instances. The `schema:email` and `schema:name` properties are used to represent the email address and name (if it is provided) of each correspondent. In the personal knowledge ontology, `personal:Agent` instances extracted from email messages with the same address and display-name are considered indistinguishable, and a unique IRI is generated for each distinct display-name–address pair.

Email messages can be retrieved from a mail server using a standard email retrieval protocol. Locally, an email message can be stored in a file with the extension `.eml`. In a standard email retrieval protocol such as IMAP (Crispin 2003), the server assigns a unique identifier to each message. The user can retrieve, delete, or create messages, but the contents of an email message cannot be modified. This gives the user significant control over the contents of her email

```
Date: Wed, 4 Jan 2017 10:25:25 0100
From: Alice S. <alice@thymeflow.com>
Message-ID: <60D86715.9060503@thymeflow.com>
Subject: Thank you for attending
To: Bob E. <bob@example.com>
Cc: carol@thymeflow.com

Dear Bob,

Thank you for participating in yesterday's event. It
was a pleasure to have you with us. We will take into
 account your suggestions for our next release!

Sincerely,
Alice S.
```

Figure 3.1: An example of an email message sent to Bob and Carol (in Cc) by Alice.

box, although in practice client implementations may restrict the operations that are allowed in order to fit common use cases. Synchronization between the server and the client is maintained using message identifiers and the guarantee of message immutability.

An example of an email message and its representation in the personal knowledge ontology are shown in Figure 3.1 and Figure 3.2.

3.2 Address books

Electronic address books are databases of contact information. Each entry represents a single contact and possesses fields such as a given name, family name, telephone number, an email address, address, and organization. Often associated with electronic communications, electronic address books are a common feature in email applications and telephone devices with memory capabilities. To allow the quick and reliable exchange, storage, and organization of contact information, a file format, known as vCard, was developed by the Versit Consortium, which was later adopted and standardized by the Internet Engineering Task Force (IETF) (Perreault 2011). However, since basic contact information management does not require the interaction between different computer systems, the standard has not been adopted as widely as email. While most widespread address book applications today support vCard, they sometimes only partially do so, do it incorrectly, or do not use the latest specification (Rossini 2012). Rossini (2012) notes that the abuse of grouped and custom properties, allowed by the vCard standard, prevents interoperability and promotes vendor lock-in.

For publishing contact information on the Web, there exists a serialization

```
@PREFIX schema: <http://schema.org/> .
@PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> .
@PREFIX personal: <http://thymeflow.com/personal#> .
@PREFIX service: <http://thymeflow.com/personal#Service/File/data> .
@PREFIX ctxt: <http://thymeflow.com/personal#Service/File/data/File?id=> .

<file:///data/message-1.eml> {
   <file:///data/message-1.eml> personal:documentOf service:File .
   <mid:60D86715.9060503@thymeflow.com>
      schema:dateSent "2017-01-04T11:25:25.000+01:00"^^xsd:dateTime ;
      schema:headline "Thank you for attending" ;
      schema:sender ctxt:edf1736e ;
      schema:text """Dear Bob [...] Alice S.""" ;
      personal:copyRecipient ctxt:337a73a0 ;
      personal:primaryRecipient ctxt:2ea82462 ;
      a schema:EmailMessage .
}
ctxt:2ea82462 {
   ctxt:2ea82462 schema:email <mailto:bob@example.com> ;
      schema:name "Bob E." ;
      personal:documentOf service:File ;
      a personal:Agent .
}
ctxt:337a73a0 {
   ctxt:337a73a0 schema:email <mailto:carol@thymeflow.com> ;
      personal:documentOf service:File ;
      a personal:Agent .
}
ctxt:edf1736e {
   ctxt:edf1736e schema:email <mailto:alice@thymeflow.com> ;
      schema:name "Alice S." ;
      personal:documentOf service:File ;
      a personal:Agent .
}
{
   <mailto:alice@thymeflow.com> schema:name "alice@thymeflow.com" ;
      personal:domain "thymeflow.com" ;
      personal:localPart "alice" ;
      a personal:EmailAddress .
   <mailto:bob@example.com> schema:name "bob@example.com" ;
      personal:domain "example.com" ;
      personal:localPart "bob" ;
      a personal:EmailAddress .
   <mailto:carol@thymeflow.com> schema:name "carol@thymeflow.com" ;
      personal:domain "thymeflow.com" ;
      personal:localPart "carol" ;
      a personal:EmailAddress .
}
```

Figure 3.2: The email message that was shown in Figure 3.1 represented in the personal knowledge ontology (TriG syntax (Carothers and Seaborne 2014)). The source is a file called`/data/message-1.eml`.

format of the vCard model that allows it to embedded inside an HTML page, known as the hCard microformat. There also exists RDF vocabularies, namely the Friend of a friend (FOAF) (Brickley and Miller 2014) and `schema.org` ontologies, which can be used for this purpose in its different serialization formats: microdata, RDFa, and JSON-LD (Bizer, Meusel, and Primpeli 2016). In applications, vCard is still however the dominant standalone file format, and this is what we use in the implementation that we present in Chapter 6, despite the aforementioned problems.

Electronic address books are a rich source of structured information. Since efficient communications depends on the accurate and efficient retrieval of contact information, the reliability, up-to-dateness and coherence of address books is high for contacts with whom the user communicates the most. In return, contacts may be represented from the point of view of the user, through the use of aliases and contextualized nouns (e.g., "Mom" and "Dad"). Regardless, we map each contact to an instance of `personal:Agent` and represent its fields using the `schema:name`, `schema:familyName`, `schema:telephone`, `schema:email`, `schema:address`, and `schema:affiliation` properties. When appropriate, the content of each field is normalized, e.g., telephone numbers are normalized to international format based on a country setting globally provided by the user.

VCards are usually stored in files with the extension `.vcf`, and are usually transmitted over email. For accessing and sharing vCard information on a server, CardDAV (Daboo 2011) can be used, which is an extension of WebDAV (Dusseault 2007), a protocol for authoring content on the Web. Based on HTTP, WebDAV allows users to create, delete, retrieve, and modify documents on a server. In WebDAV, documents are grouped in directories, and directory hierarchies are supported. HTTP URIs are used to identify documents and directories. The authority part of the URI identifies the server, and its path represents the location of the document or directory in the hierarchy. To track changes to documents or directories, WebDAV uses entity tags and last-modified dates, which are HTTP features. In CardDAV, each document is a vCard, and each vCard has an identifier (set on the vCard's UID field) that is guaranteed to be unique within a directory. UIDs, entity tags, and last-modified dates are useful for client–server synchronization. Given the appropriate permissions, the user has complete control over the information stored in a CardDAV server.

An example of a vCard and its representation in the personal knowledge ontology are shown in Figure 3.3 and Figure 3.4.

3.3 Calendars

Electronic calendars are databases of planned events that are useful for managing schedules. Events may be past, present, or future. Email software often incorporate electronic calendars, as the planning of appointments is an important part of electronic communications. Along with vCard, the standard format for address books (cf. Section 3.2), the Versit Consortium also developed vCalendar, a file format for the exchange of calendaring and scheduling information. The IETF later standardized iCalendar (Desruisseaux 2009), which is heavily based

```
BEGIN:VCARD
VERSION:3.0
UID:urn:uuid:a75c0fb9-ba2a-4bf9-8272-6beda5a258c7
N:S.;Alice;;;
FN:Alice S.
ORG:Thymeflow
TITLE:CEO
TEL;TYPE=WORK:01 99 99 99 99
TEL;TYPE=CELL:07 99 99 99 99
ADR;TYPE=WORK:;;15 Place Vendôme;Paris;;75001;France
URL:http://alice.thymeflow.com/
EMAIL;TYPE=INTERNET:alice@thymeflow.com
END:VCARD
```

Figure 3.3: Alice's own contact information in vCard format.

on vCalendar and the most widely used calendar format today. In particular, iCalendar defines a transport-independent protocol for implementing workflow for scheduling an event between multiple users using different calendaring systems. The protocol defines operations such as requesting for and replying with free/busy time information as well as requesting, replying to, modifying, and canceling an event (Daboo 2009). A transport in which this protocol is typically implemented is email. Besides event and free/busy time information, iCalendar allows the representation of to-dos and journal entries. The latter two are not supported by all providers, and we do not consider them in this thesis. The representation of events in iCalendar benefits from good interoperability between different email applications as it is a requirement in business environments for effective collaboration between users. For publishing calendar information on the Web, hCalendar and xCal can be used, which are respectively microformat and XML representations of iCalendar. An alternative is to use the `schema.org` (using the microdata, RDFa, or JSON-LD serialization format).

As discussed in Section 2.1, calendars capture the notion of *scheduled events*. An iCalendar event includes a summary, a description, an organizer, a start date, an end date, a location, and a list of attendees. Each such event is mapped to an instance of `schema:Event`, and its attributes are mapped to the appropriate `schema.org` properties (cf. Section 2.2). In particular, the summary of an event is represented using the `schema:name` property, and its location is represented by a instance of `schema:Place` that is defined by a postal address (some text) or some geographic coordinates.

ICalendars can be stored in files with the extension `.ics`. For accessing and sharing iCalendar information on a server, CalDAV can be used (Daboo, Desruisseaux, and Dusseault 2007), which is analogous to CardDAV in terms of vCard sharing. CalDAV benefits from the same authoring, control, and synchronization capabilities as CardDAV (cf. Section 3.2).

```
@PREFIX schema: <http://schema.org/> .
@PREFIX personal: <http://thymeflow.com/personal#> .

<file:///data/contacts.vcf> {
   <file:///data/contacts.vcf> personal:documentOf <http://thymeflow.com/
       personal#Service/File/data/File> .
   <urn:uuid:a75c0fb9-ba2a-4bf9-8272-6beda5a258c7>
      schema:address [
         schema:addressCountry _:a ;
         schema:addressLocality [
            schema:containedInPlace _:a ;
            schema:name "Paris" ;
            a schema:Place
         ] ;
         schema:postalCode "75001" ;
         schema:streetAddress "15 Place Vendôme" ;
         a schema:PostalAddress
      ] ;
      schema:email <mailto:alice@thymeflow.com> ;
      schema:familyName "S." ;
      schema:givenName "Alice" ;
      schema:jobTitle "CEO" ;
      schema:memberOf [
         schema:name "Thymeflow" ;
         a schema:Organization
      ] ;
      schema:name "Alice Smith" ;
      schema:telephone <tel:+33-1-99-99-99-99>, <tel:+33-7-99-99-99-99> ;
      schema:url <http://alice.thymeflow.com/> ;
      a personal:Agent .
   _:a
      schema:name "France" ;
      a schema:Country, schema:Place .
}

{
   <mailto:alice@thymeflow.com> schema:name "alice@thymeflow.com" ;
      personal:domain "thymeflow.com" ;
      personal:localPart "alice" ;
      a personal:EmailAddress .
   <tel:+33-1-99-99-99-99> schema:name "+33 1 99 99 99 99" ;
      a personal:PhoneNumber .
   <tel:+33-7-99-99-99-99> schema:name "+33 7 99 99 99 99" ;
      a personal:PhoneNumber .
}
```

Figure 3.4: The representation of Alice's own contact information (cf. Figure 3.3) in the personal knowledge ontology (TriG syntax). The source is a file called `/data/contacts.vcf`.

```
BEGIN:VCALENDAR
PRODID:-//Thymeflow//Thymeflow Calendar 1.0//EN
VERSION:2.0
CALSCALE:GREGORIAN
METHOD:PUBLISH
BEGIN:VEVENT
DTSTART:20170103T180000Z
DTEND:20170103T190000Z
DTSTAMP:20161222T131058Z
UID:20161222T131058Z-ba2a-4b9-8272@thymeflow.com
CLASS:PUBLIC
DESCRIPTION:Demo & Discussion & Drinks
LOCATION:Le Bristol Paris
SEQUENCE:0
STATUS:CONFIRMED
SUMMARY:Thymeflow meetup
TRANSP:OPAQUE
ORGANIZER;CN=Alice S.:mailto:alice@thymeflow.com
ATTENDEE;CN=Bob E.:mailto:bob@example.com
ATTENDEE:mailto:carol@thymeflow.com
END:VEVENT
END:VCALENDAR
```

Figure 3.5: Alice's calendar in iCalendar format.

An example of an iCalendar and its representation in the personal knowledge ontology are shown in Figure 3.5 and Figure 3.6.

3.4 Social networking services

Social networking services are online platforms where users publish service-specific profiles, build social networks by connecting with other users, and submit content such as photos, posts, and comments, which they share with their networks. Notable examples include Facebook, Google+, Twitter, LinkedIn, Qzone, Instagram, and VK. Unlike email, which is decentralized and whose implementation is based on open standards, major social network services centralize information from their users in a single proprietary platform and provide web-based and/or mobile-based user interfaces with this platform. These interfaces are intended to be used by the end-user and are not suited for the automated extraction of user information. However, most major social networking services also provide documented and publicly accessible programmatic web interfaces (web APIs) that provide access to the user information on their platforms and are meant to be used by third-party developers to develop all kinds of applications. Facebook is

```
@PREFIX schema: <http://schema.org/> .
@PREFIX personal: <http://thymeflow.com/personal#> .

<file:///data/calendar.ics> {
   <file:///data/calendar.ics> personal:documentOf <http://thymeflow.com/
       personal#Service/File/data/File> .

   <urn:uuid:5a9b71b0-9d6a-38bb-a22b-926328808107>
      schema:attendee <urn:uuid:28691712-a54f-3f2f-ae59-4fd3adddbc87> ;
      schema:description "Demo & Discussion & Drinks" ;
      schema:endDate "2017-01-03T20:00:00.000+01:00"^^xsd:dateTime ;
      schema:location [
         schema:name "Paris" ;
         a schema:Place
      ] ;
      schema:name "Thymeflow meetup" ;
      schema:organizer <urn:uuid:fa6b49fd-9b1f-32ce-b05f-ffcddf64c353> ;
      schema:startDate "2017-01-03T19:00:00.000+01:00"^^xsd:dateTime ;
      a schema:Event .

   <urn:uuid:fa6b49fd-9b1f-32ce-b05f-ffcddf64c353> schema:email <mailto:
       alice@thymeflow.com> ;
      schema:name "Alice S." ;
      a personal:Agent .

   <urn:uuid:28691712-a54f-3f2f-ae59-4fd3adddbc87> schema:email <mailto:
       bob@example.com> ;
      schema:name "Bob E." ;
      a personal:Agent .
}

{
   <mailto:alice@thymeflow.com> schema:name "alice@thymeflow.com" ;
      personal:domain "thymeflow.com" ;
      personal:localPart "alice" ;
      a personal:EmailAddress .

   <mailto:bob@example.com> schema:name "bob@example.com" ;
      personal:domain "example.com" ;
      personal:localPart "bob" ;
      a personal:EmailAddress .
}
```

Figure 3.6: The representation of Alice's calendar (cf. Figure 3.5) in the personal knowledge ontology (TriG syntax). The source is a file called `/data/calendar.ics`.

one such service that we consider in our implementation.

With 1.79 billion monthly active users worldwide in September 2016, Facebook is one of the largest social networking services (Facebook 2016a). The web API that Facebook provides is called the Facebook Graph API (Facebook 2016[b]). To use it, the developer of an application needs to register the application with Facebook. Then the application has to implement an OAuth authorization flow (Mayrhofer and Spanring 2010), which allows Alice to grant the application access to her information on Facebook without giving her account credentials to the application (e.g., her Facebook's password). This authorization allows the application to access the user's profile information (e.g., her name, date of birth, or hometown), events that she is interested in, is attending, or has attended, and her photo albums, posts, status updates, and conversations, as well as any other information that is connected (e.g., places, likes, photos, or videos). The API has its limitations, however. First, due to privacy concerns, Facebook no longer allows an application to access certain parts of Alice's personal information that is about users that have not explicitly authorized the application (Constine 2015). For example, the application cannot read the contents of a photo album or a status update posted by a friend of Alice who has not authorized the application. These elements, which are accessible to Alice via the human web interface, may be considered to be part of Alice's personal information, as per the definition given in Section 1.1, since Alice may *experience* (view) them at some point. Second, as of July 2015, the API no longer allows access to the user's message inbox (Facebook 2017), which is definitely information *directed to* the user.

Despite these limitations, the API can be used to extract some useful information, such as the list of events that the user is interested in, is attending, or has attended, and include the description, location, dates, and list of guests of each event. Each Facebook event is modeled by an instance of `schema:Event`, similarly to how it is done for calendar events (cf. Section 3.3). Facebook events have the advantage over iCalendar events in that they usually include more accurate and richer information (e.g., the event's location, list of guests, and guest responses). Each guest of a Facebook event is represented by an application-specific identifier and a name. Each Facebook event possesses a unique identifier and a last-modification date, which can are useful to track changes. The API allows events and other objects to be retrieved and created by the application, but only those created this way can be modified or deleted. Many fields of the user's profile, including her profile picture cannot be changed as well. As a result, the API does not grant complete control over the user's information. The user can still use the web-based user interface to perform changes that are otherwise impossible via the API, except for changes involving information created by other users.

An example of a Facebook event represented in the personal knowledge ontology is shown in Figure 3.7.

```
@PREFIX schema: <http://schema.org/> .
@PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> .
@PREFIX facebook: <https://graph.facebook.com/> .

<http://graph.facebook.com> {
   <http://graph.facebook.com> personal:documentOf <http://thymeflow.com/
       personal#Service/Facebook/alice@thymeflow.com/Facebook> .

   facebook:165465164
      schema:attendee facebook:101983248 facebook:101534565
         facebook:101541549 facebook:101953327 facebook:101541541
         facebook:101541568 facebook:101541237 facebook:101543035 ;
      schema:name "Alice's 25th Birthday Party" ;
      schema:description "Drinks and dancing for my 25th birthday!" ;
      schema:image <https://scontent.xx.fbcdn.net/1100.jpg?oh=ec30> ;
      schema:startDate "2017-02-14T21:00:00+01:00"^^xsd:dateTime ;
      schema:location facebook:105863085 ;
      a schema:Event .

   facebook:105863085
      schema:address [
         schema:addressCountry _:b ;
         schema:addressLocality [
            schema:containedInPlace _:b ;
            schema:name "Paris" ;
            a schema:Place
         ] ;
         schema:postalCode "75012" ;
         schema:streetAddress "46 Rue du Faubourg Saint-Antoine" ;
         a schema:PostalAddress
      ] ;
      schema:geo <geo:48.85219472,2.37266230> ;
      schema:name "Barrio Latino" ;
      a schema:Place .
   _:b schema:name "France" ;
      a schema:Country .

   facebook:101983248
      schema:name "Alice S." ;
      schema:givenName "Alice" ;
      schema:familyName "S." ;
      schema:gender "female" ;
      schema:image <https://scontent.xx.fbcdn.net/p50x50/134.jpg?oh=dee5> ;
      a schema:Person .
}
{
   <geo:48.85219472,2.37266230> schema:latitude 48.85219472 ;
      schema:longitude 2.37266230 ;
      a schema:GeoCoordinates .
}
```

Figure 3.7: The representation of a Facebook event that Alice is attending in the personal knowledge ontology (TriG syntax).

3.5 Mobile device sensors

As mentioned in Section 2.3, information about the mobile device's physical environment is useful for reconstructing a history of the user's activities. Mobile devices can gather information from different kinds of radio-based technologies: cellular networks (e.g., GSM, UMTS, CDMA, and LTE), Bluetooth, Wi-Fi, NFC, and satellite-based navigation systems (e.g., GPS, GLONASS, and BDS). Most of these technologies are designed for two-way communications (e.g., for exchanging voice, text, or media), with the exception of satellite-based navigation systems, which are designed for (geographic) positioning. Mobile devices also embark sensors capable of tracking motion (e.g., an accelerometer and a gyroscope), orientation (e.g., a magnetometer), ambient sound (e.g., a microphone), ambient air parameters (e.g., a barometer, thermometer, and an air humidity sensor), ambient light, and user proximity (e.g., infrared and hall sensors). For collecting, exchanging, and storing information captured by these sensors and radio-based technologies, no uniform methods exist. However, the operating systems of most modern mobile devices provide APIs to uniformly access data from sensors and radio-based technologies, regardless of the underlying hardware and implementation specifics. Notable operating systems of this kind are the Android (Google 2017a), iOS (Apple Inc. 2017b), and Windows Phone (Microsoft 2017) families.

Sensor information

In this thesis, we are interested in sensors capable of tracking the user's location: satellite-based navigation systems and network-based positioning systems (Wi-Fi or cellular-based); tracking her motion: accelerometers and gyroscopes; and describing the user's environment (e.g., indoors, outdoors, or underground) by measuring the signal strengths of wireless network devices in range: Bluetooth devices, Wi-Fi access points, cellular sites (towers), and satellites. We refer to these categories of sensor information by the names *locational*, *dynamic*, and *contextual*, respectively.

Locational

Mobile devices are capable of tracking the user's (device's) geographic location over time using different radio-based technologies: satellite-based navigation systems (e.g., GPS, GLONASS, and BDS), Wi-Fi, and cellular network technologies, with varying degrees of accuracy. Satellite-based navigation systems are the most accurate, work all over the globe, and are capable of measuring the velocity of the device in addition to finding its position. The device's velocity is defined by a magnitude and a bearing, which defines the direction of movement as an angle relative to the direction of the north-pole. The accuracy of satellite-based positioning can be affected in areas with a partial line of sight of the sky, e.g., inside buildings and especially underground. Wi-Fi and cellular-based positioning are based on the identification of Wi-Fi access points and cellular sites and a mapping of their identifiers to geographic locations. Contrary to satellite-based positioning, Wi-Fi and cellular-based positioning can work indoors or even underground, such

as inside transit tunnels. However, Wi-Fi and cellular-based positioning systems depend on the local Wi-Fi and network infrastructure. For this reason, they are not available worldwide and their positioning accuracy varies depending for instance on the density of Wi-Fi access points and cell sites. The accuracy of satellite-based positioning is in order of 10 meters, for Wi-Fi it is in the order of 100 meters, and for network-based positioning it is in the order of 1 kilometer.

The location APIs of most mobile platforms do not distinguish between Wi-Fi and cellular-based positioning and instead only provide a network-based positioning service which internally combines both sources of information. This is case for the Android, iOS, and Windows Phone platforms. An application developed for these platforms can request the current location or register to location updates (either periodic or triggering after the device has moved a certain distance from the previous location). Occasionally, the device may be unable to determine its position, in which case updates may not come as often as requested. A location has a timestamp, a set of geographic coordinates (i.e., a latitude, longitude, and an optional altitude), an accuracy and an optional velocity (defined by a magnitude and a bearing). The accuracy of a location is defined by a radius, in meters, within which the system has a 68% confidence that the device is located. Assuming that the error is normally distributed, this radius roughly corresponds to one standard deviation.

Dynamic

The accelerometer of a mobile device measures the device's *proper acceleration*, that is, the physical acceleration experienced by the device relative to free fall (free fall is the device's motion where gravity is the only force acting upon it). Because of this, when the device is at rest on the surface of the Earth, the accelerometer approximately indicates an upward acceleration that is equal in magnitude to Earth's gravity. The measured acceleration is expressed as a vector in a three-dimensional Cartesian coordinate system with axes fixed to the device's frame. The gyroscope of a mobile device measures the device's angular velocity. The measured angular velocity is a three-dimensional vector indicating the device's rate of rotation around each of the axes in the acceleration's coordinate system. The measurement frequencies of the accelerometer and the gyroscope range from tens to hundreds of hertz. These sensors are useful in applications where the user interacts through motions and gestures, e.g., by tilting, shaking, or rotating the device. Also, since these sensors indirectly measure the user's motion, they can also be used to recognize the user's activity, as we will see in Section 5.8. Some mobile devices also embed a magnetometer, which is a sensor for measuring the magnetic field. Measurements by the accelerometer, gyroscope, and magnetometer can be combined to obtain an accurate estimation of the device's orientation with respect to the Earth's surface. Combining sensor data in such way is an example of *sensor fusion*, which is a subset of information fusion (cf. Section 2.3). The estimated orientation can then be used to represent the proper acceleration of the device with respect to the ground.

By design, the personal knowledge ontology presented in Section 2.2 is not able to represent sensor measurements by an accelerometer, a gyroscope, or a

magnetometer. Such measurements are too low-level to be directly related to the dimensions that the ontology aims to capture (i.e., who? what? when? where? etc.). In Chapter 5, however, we show how these sensor measurements can be used to derive the user's mode of transportation, which is closer to what the ontology aims to capture.

Contextual

Some information about the user's environment can be inferred from the measured signal strengths of in-range devices for the different radio-based technologies listed at the beginning of this section. Signal reception depends on whether the user is in a building, underground, or outside in an open-field, and varies depending on the technology. For instance, satellite-based navigation reception is low or inexistent underground and Wi-Fi is abundant indoors. For some technologies, the measured signal strength may vary over the course of the day in a single place. For example, persons may carry Bluetooth-enabled devices with them (e.g., smartwatches and smartphones), and so the number of detectable Bluetooth devices at any given point may be used to estimate the number of people around it.

Similarly to the modeling of *dynamic* sensor information, *contextual* information was intentionally not captured by the personal knowledge ontology. In Chapter 5, we show how to use it to derive the user's mode of transportation.

Applications

To collect the different kinds of sensor information previously presented, we developed two mobile applications: *Thymeflow mobile* and *Hup-me mobile*. We now present them.

Thymeflow mobile

Thymeflow mobile is an Android and iOS application that we developed to collect a history of the user's location. The resulting location history is a time-ordered sequence of locations visited by the user over the course of time. A location history application typically runs forever in the background, periodically collecting and storing the location either locally (on the device) or on a distant server. By default, *Thymeflow mobile* collects the user's location every minute, stores it locally, and uploads every hour the new locations to a distant server that the user controls. The distant server runs Cozy (Cozy Cloud 2016), an open-source personal cloud platform. The application generates a unique identifier to the device it runs on, which it attaches to every collected location. Locations are formatted as JSON objects and uploaded to Cozy, which stores them in a CouchDB (The Apache Software Foundation 2017a) database, which is a JSON-based document store. The user can then query the database to retrieve the locations in a given time-range, and in particular visualize the locations for a given day.

For managing a location history, as for mobile device sensor data in general, no standardized method exists. However, there exist other location history applica-

tions for Android, such as Google Timeline and Location History (Google 2015b), GPS Logger (Mendhak 2011), Moves (ProtoGeo 2013), and OpenPaths (The New York Times Company 2012). Google Timeline is usually installed with the Android platform by default. Locations are stored on Google servers and attached to the user's Google account and the user can browse her location history using the web and mobile interfaces. However, Google does not provide a public API to retrieve the user's location history. The user can only manually download a full dump of her location history, which is not efficient for managing updates. Moves is similar to Google Timeline except that Moves provides a public API to query the user's history of locations and activities. The OpenPaths's application functions likes Moves, except that it does minimal processing and does not provide a history of the user's activities. GPS Logger, on the other hand, directly stores the locations on the device in GPX, KML, or CSV format. GPX and KML are XML-based formats for exchanging GPS data and generic geographic information, respectively. GPS Logger can periodically send the saved files to the user's email address, or upload them to a FTP server or a cloud storage provider such as DropBox, OwnCloud, or Google Docs. Locations can also be directly logged by issuing an HTTP request to a custom URL or a OpenGTS (GeoTelematic Solutions, Inc. 2017) server.

Thymeflow mobile is different from the applications above in the following aspects:

1. Contrary to Google Timeline, locations are stored in a database which the user can freely query.

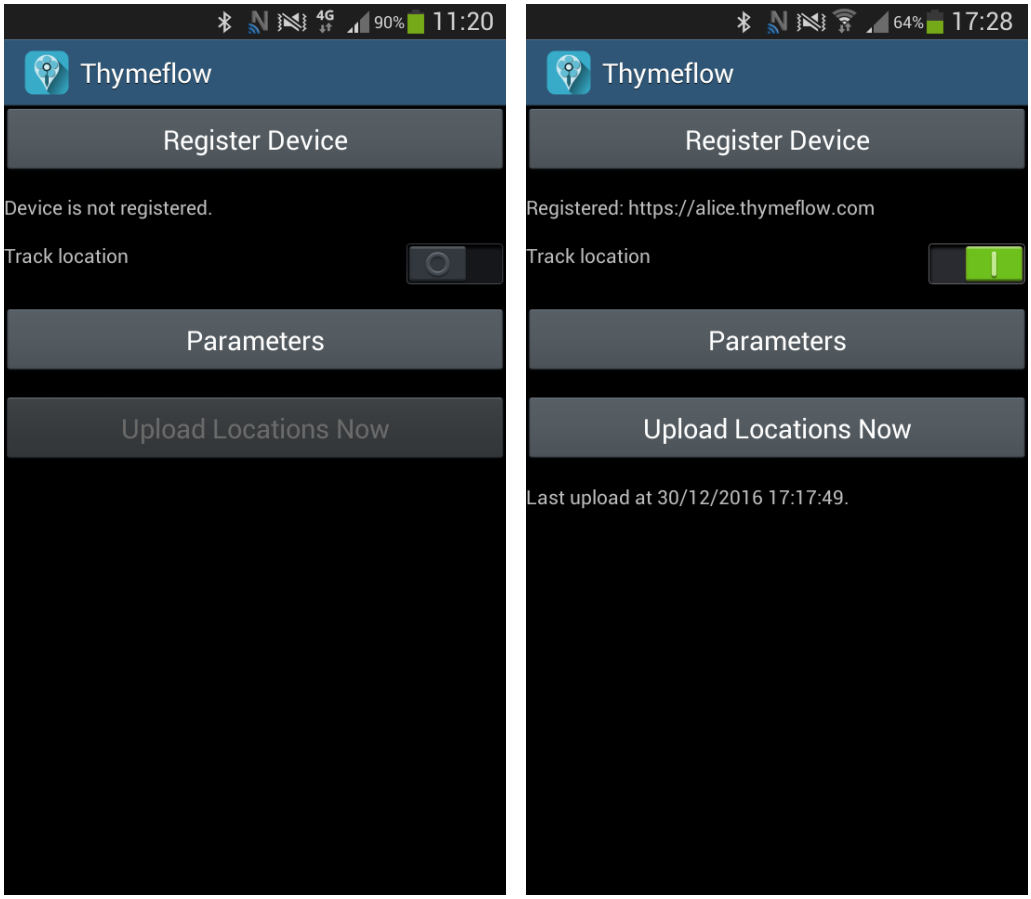
2. Google Timeline can be installed on multiple devices with the same Google account. However, when the user exports her Google's location history, locations are not associated with devices, which makes it difficult to process the location history when the user has multiple devices and does not always travel with all of them (e.g., when she leaves her tablet at home). In *Thymeflow mobile*, device identifiers make it possible to distinguish between the different device traces.

3. Different from Moves and OpenPaths, locations are stored in a platform that the user fully controls.

4. Different from OpenPaths, our application also registers the reported accuracy of a location.

5. Different from GPS Logger, we used an implementation of a location request manager which trades-off some accuracy in exchange of improved battery life by automatically switching between satellite and network location providers depending on various circumstances.

Implementation *Thymeflow mobile* was developed in C# using the Xamarin framework (Xamarin Inc. 2017). Xamarin is an open-source framework for developing cross-platform mobile applications while sharing code across multiple platforms. The application was developed for the Android and iOS platforms.



(a) Unregistered. Tracking is off. (b) Registered. Tracking is on.

Figure 3.8: *Thymeflow mobile* possible states: Initially, the application is not yet registered to the Cozy server and tracking is off. Once registered, tracking can be either on or off.

The shared code base includes the user interface, configuration manager, location storage (in a SQLite databse), and location uploader (as a Cozy client). Specific to each platform is the code to collect locations: periodically waking up the device and issuing a location request to the appropriate sensor. For Android, we use Google Play Services's location manager (Google 2017c) For iOS, we use iOS's native location manager (Apple Inc. 2017a). The requested location accuracy is about 100 meters.

The application has three states (Figure 3.8). Initially, no Cozy server is registered and tracking is off. Once registered, tracking can be turned on or off. The location request period and the upload period can be changed (Figure 3.9). The location request period is ignored in the iOS version of the application since the frequency of background location updates is subject to platform constraints.

Hup-me mobile

Hup-me mobile is an Android application that we developed to collect *locational*, *dynamic*, and *contextual* sensor information from traveling users. Contrary to

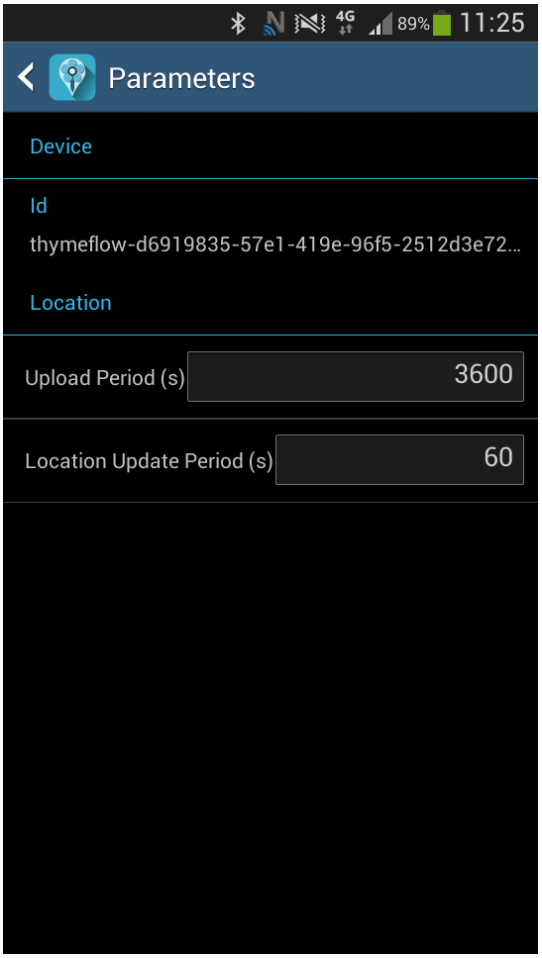


Figure 3.9: The configuration panel of *Thymeflow mobile*. The device identifier is displayed. The location request period and upload period can be configured.

Thymeflow mobile, the application was not designed for the end-user. Instead, it was designed to collect mobile sensor data from multiple users willing to participate in the activity and transportation mode recognition experiments that we present in Chapter 5. After the application has collected data about some trip and stored it on the device, a user that chooses to participate can anonymously upload this data to a server that we provided for these experiments. Because *Hup-me mobile* collects information from many more sensors and does so at a higher frequency (e.g., satellite location updates are requested every second) than *Thymeflow mobile*, *Hup-me mobile* consumes considerably more power than *Thymeflow mobile*. However, contrary to *Thymeflow mobile*, *Hup-me mobile*'s data collection is limited to certain periods of time over the course of the user's day: moments where she travels from one place to another. To do so, users have to manually trigger the collection of data when they start traveling, and stop it when reach their final destination. Besides preserving battery, this has the advantage of reducing the amount of data that is uploaded and that we need to process. Another advantage is that users participating in our experiment have control over what information is collected about them.

A period of time over which sensor data was collect for a user is called a *journey*. Sensor data for a journey is stored in tabular format in the device's local file-system as a set of CSV files. Each sensor generates its own CSV file, where each row corresponds to one measurement. Every CSV file has a "time" column that represents the time of each measurement. Each *journey* is stored in its own folder. Along with CSV files, a JSON file is also saved that contains information about the collection and the device: the device's model, operating system's version, sensors' hardware specifications, application version, collected sensors, journey start and end times. Finally, in addition to the sensors previously presented, the application also logs the battery status of the device, i.e., whether the device is charging and the remaining battery percentage, as well as whether the screen is on. The screen status sensor (called the *interactivity* sensor) helps us know whether the user is currently using the device at any given moment.

Clock synchronization One difficulty when collecting information from different kinds of sensors is dealing with clock synchronization. Low-level sensors such as the accelerometer or the gyroscope operate on high resolution clocks that correctly measure the elapsed time from the beginning of a journey (from the beginning of data collection) but do not necessarily relate to a standard time such as Coordinated Universal Time (UTC). The exact specifications of these clocks are implementation dependent. On the other hand, the system clock of the device is synchronized to UTC but does not have as much resolution. Also, the system clock may occasionally jump backwards or forwards when the clock goes out of sync with UTC and it is resynchronized using some clock synchronization networking protocol (e.g., the Network Time Protocol). Finally, satellite-based navigation systems keep their own clock, which may be used to represent the times associated with location measurements. The times associated with these locations have the advantage of being highly accurate (with respect to the navigation system's clock), since clock synchronization is a requirement for proper satellite-based positioning.

During the development of *Hup-me mobile*, we progressively realized that the best way to have comparable measurements between different sensors is to collect for each sensor the time from the clock that measures elapsed time the most accurately for this sensor. Elapsed time is guaranteed to be monotonic and to continue ticking even when the device is idle. To be able to synchronize the times between different types of measurements, the average time difference between the elapsed time clock of each sensor and a universally available elapsed time clock has to be computed. Android implements one such clock that measures the elapsed time since boot in nanoseconds. Finally, UTC time should be measured at the beginning of data collection so as to be able to relate all sensor time measurements to UTC. When the goal is infer the user's transportation mode from these sensor measurements, this synchronization is needed for making use of transit schedules. For a high-level sensor measurement for which no elapsed time clock is provided for the sensor by the system (e.g., the signal strengths of Wi-Fi devices in range or the signal strength of the cellular network), the device's elapsed time since boot can be retrieved at the moment the application collects

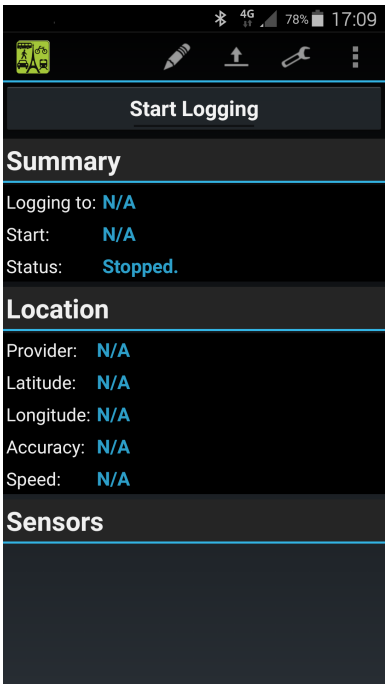
the measurement. Since location updates can be received by the application some time after they were acquired, it is important to use the (sensor) time provided with the measurement. Additionally, since version 4.2, Android's API allows us to reliably retrieve the elapsed time since boot of a location measurement. For older versions, the time is given by the internal clock of the positioning system. For satellite-based positioning, this is accurate, while for network it is not. For low-level sensor measurements (e.g., accelerometer or gyroscope measurements), which are of high frequency, the sensor's internal clock should be used.

In *Hup-me mobile* we did not initially implement all the of the above suggested practices. We did so for low-level sensor measurements, but did not for location data, and most of the other measurements were done using the system's UTC clock. Therefore, we later designed and implemented a procedure to synchronize the times of satellite-based location measurements collected this way with the other sensor measurements, and manually verified the result of this synchronization during our experiments.

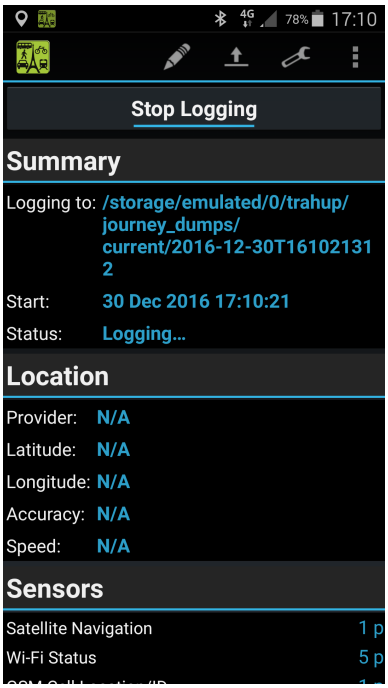
Implementation The application was developed in the Scala (EPFL 2017) programming language using the Android SDK. The user can start and stop the collection of data by tapping the "Start" and "Stop logging" buttons (Figure 3.10). When the application is logging a journey, the device is kept awake. The frequency of measurements varies from one sensor to another (Figure 3.11). Some sensors, such as "Phone Status", only pick up measurements when detecting a change of state. Others, such as "Satellite Navigation", pick up measurements at a fixed rate (e.g., once per second), regardless of the estimated state. The set of logged sensors can be configured (Figure 3.12).

3.6 Related work

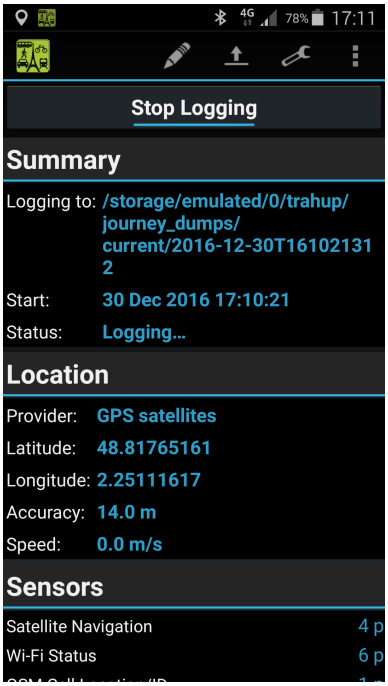
In data warehousing, the process by which data is moved and transformed from one or multiple data sources into a target data store is referred to as Extract-Transform-Load (ETL) (Vassiliadis 2009). ETL is typically performed in a periodic fashion, so that the target store keeps up with changes in the data sources. In data migration, i.e., where the sources are to be decommissioned and the target store is to bound to completely replace them, ETL tools can be used, although the full range of problems dealt by ETL is usually not encountered in migration projects. Data warehousing is one approach to data integration (cf. Section 2.3). The ETL process has three stages: extracting data from the sources (*data extraction*), transforming the data into the target schema and format (*data transformation*), loading of the transformed data into the appropriate location of the target data store (*data loading*). Each of these stages presents unique challenges. For instance, *data extraction* may entail the implementation of differential snapshots to reduce the communication overhead. *Data transformation* is concerned with modeling and implementing data mappings, managing lineage (i.e., identifying the origin and transformations of the target data), and data cleaning (i.e., removing inconsistencies, duplicates). *Data loading* has deal with



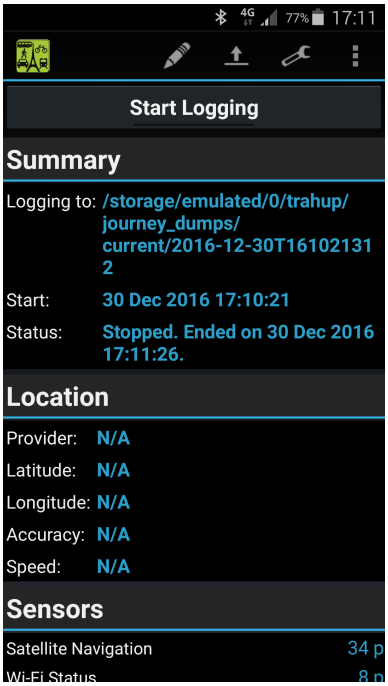
(a) Ready to start logging.



(b) Logging: no location has been acquired yet.



(c) Logging: a location has been acquired.



(d) Logging has stopped.

Figure 3.10: The main panel of *Hup-me mobile*: the user can manually start and stop the collection of data. A *journey* is created each time logging starts.

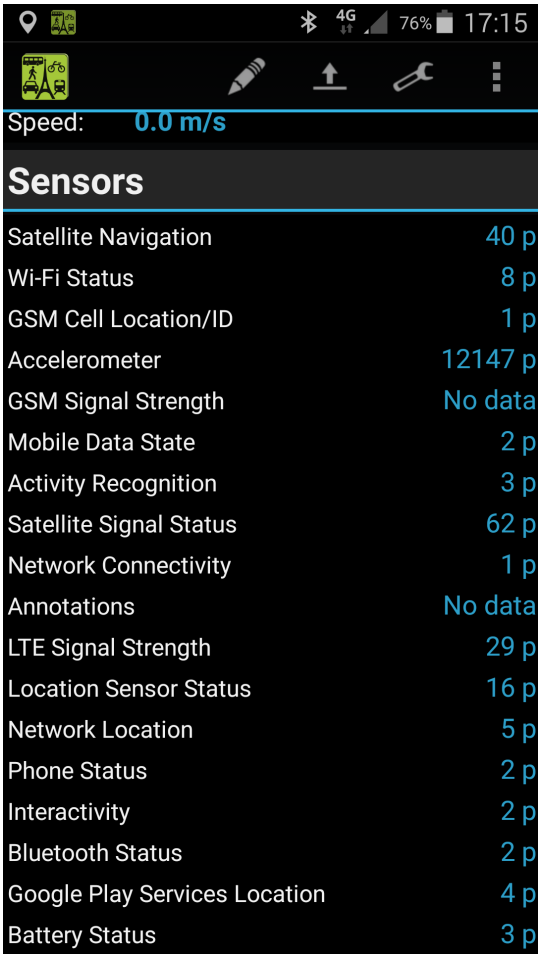


Figure 3.11: Sensor measurement metrics in *Hup-me mobile*: for each logged sensor the application displays the number of measurements performed by this sensor thus far.

performance issues, e.g., bulk loading and index maintenance. Overall, ETL processes have to be resilient: resumption after failure, and exception handling and reporting, while maintaining coherence in the target store.

In our setting, many tools exist for the extraction and transformation of Email, vCard, iCalendar, and Facebook data into RDF. The W3C maintains a list of such tools (W3C 2017). Some of these tools represent knowledge using the FOAF (Brickley and Miller 2014), Semantically-Interlinked Online Communities (SIOC) (Uldis and Breslin 2010), and `schema.org` ontologies. These tools do not commonly handle mobile sensor data, as sensor data collection tools and formats are less standard and are usually integrated within a larger application that only outputs a transformed result (Lane et al. 2010). Regardless, Open Geospatial Consortium's Sensor Web Enablement has produced a series of standards to describe and manage sensors, their observations, and sensing processes. One of such standards is SensorML (Mike and Robin 2014). The W3C's Semantic Sensor Network Incubator Group have extended these standards to the Semantic Web by proposing the Semantic Sensor Network ontology (Lefort, Henson, and Taylor

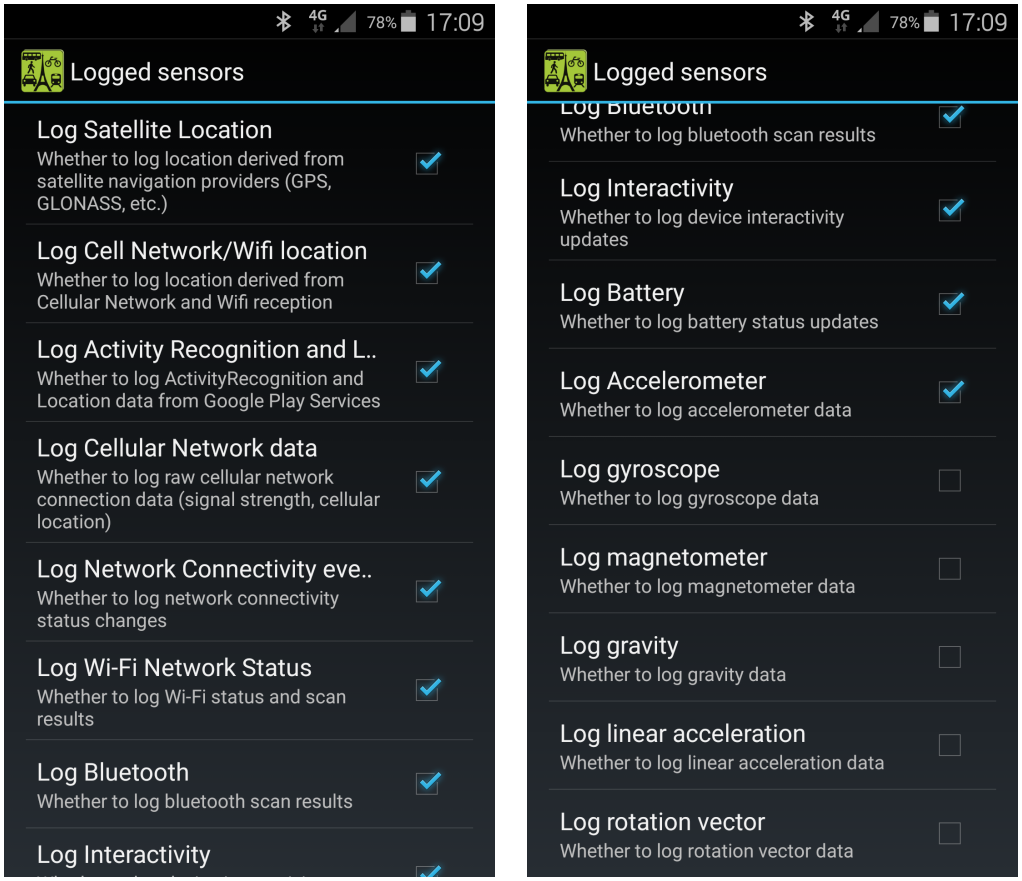


Figure 3.12: *Hup-me mobile*: The user can configure the list of logged sensors.

2011). However, these standards are primarily aimed at large-scale deployments of sensor networks, and we are not aware of a user-oriented mobile sensing application using them.

Chapter 4

Spatiotemporal knowledge: Stay extraction

The algorithm described in this chapter has been integrated into the personal information management system that we present in Chapter 6.

As presented in Section 2.3, one of our goals is to recreate Alice's history of activities from mobile device sensor data. In this chapter, we describe a process that uses this data to find Alice's *stays* and *moves* (Section 2.2). This process is called *stay extraction*.

4.1 Introduction

Stay extraction is the task of finding non-overlapping periods of time where Alice has remained in one place, and determine these places. Formally, the goal is to find, given a time interval $I = [start, end]$ and a set of observations O over this interval, a sequence of stays $(S_i)_{1 \leq i \leq n} = (s_i, e_i, \pi_i, \alpha_i)_{1 \leq i \leq n}$, where for each i, π_i is the center (a geographic point), α_i is the accuracy, and $[s_i, e_i] \subset I$ is the time span of the stay S_i. Additionally, the non-overlapping condition requires that for all $i \geq 2$, $e_{i-1} \leq s_i$.

To determine these periods, one could imagine using various sources of sensor information. A natural source for this information is her location history. Many users may already possess a location history, perhaps unknowingly, as they are very easy to collect and many applications exist to do so (cf. Section 3.5). For these reasons, we use Alice's location history for the stay extraction task.

This chapter is structured as follows. In Section 4.2, we formalize the notion of a location history, illustrate it with visualizations, and present different clustering strategies. In Section 4.3, we introduce *Thyme*, a *stay extraction* algorithm that we designed. Section 4.4 discusses an evaluation of this algorithm. We then present some related work and conclude this chapter.

4.2 Location history

As presented in Section 3.5, a location history is a time-ordered sequence of locations. Formally, it can be modeled as a sequence $(l_i)_{1 \leq i \leq T} = (t_i, p_i, u_i)_{1 \leq i \leq T}$, where for each i, t_i is the time, p_i is the geographic point and u_i is the accuracy of the location l_i. Additionally, for all $i > 2$, we have $t_i < t_{i+1}$.

Visualization

To understand the nature of location histories, we start by describing how they can be visualized. One simple way of visualizing a location history is to project its points on a map, possibly connecting time-consecutive locations with lines, as shown in Figure 4.1. To represent time, a third axis can be added, making the visualization three-dimensional. This is known as the "space-time cube" (Kraak 2003). Another way is to color points on the 2D visualization according to the time of the location that they represent, using for instance a gradient color scale based on the hue. To be meaningful, these visualizations rely on not having to represent too many locations, i.e., on the representation of a short period of time (e.g., a day). For the 3D visualization, the map is only shown at the bottom of the cube, making it hard to read the spatial dimension if the history is too complex. When using the flat visualizations to represent long histories, too many points may end-up being stacked over each other, masking the readability of the time dimension (color and/or connectivity). Being able to filter over the time dimension of the location history is thus necessary. While simple time range filters might work, more elaborate methods exist for navigating through time and space (Thudt, Baur, and Carpendale 2013).

Spatial clustering

When visually inspecting Alice's location history (Figure 4.1), we may notice certain clusters of points. We can make them explicit by agglomerating points that are close to each other into clusters and indicating the number of points forming each cluster on the map, as shown in Figure 4.2. Figure 4.3 shows a detailed view of the locations in the bottom-left part. Assuming that locations are uniformly distributed over the course of the analyzed period, the number of points around a certain area is indicative of the total amount of time that Alice spent in that area. Thus, to perform stay extraction, one could imagine performing spatial clustering on the geographic points of the set of analyzed locations. However, as shown by Kang et al. (2004), this does not work so well. To understand this, one could see spatial clustering as grouping close points together whereas stay extraction is more about separating points into two categories, i.e., distinguishing points belonging to *stays* from those belonging to *moves*. Because point density is related to the total amount of time Alice spent in an area, spatial clustering tends to detect places that Alice frequently visits or passes by rather than places where she stayed. Specifically, for spatial clustering to work, it would have to deal with:

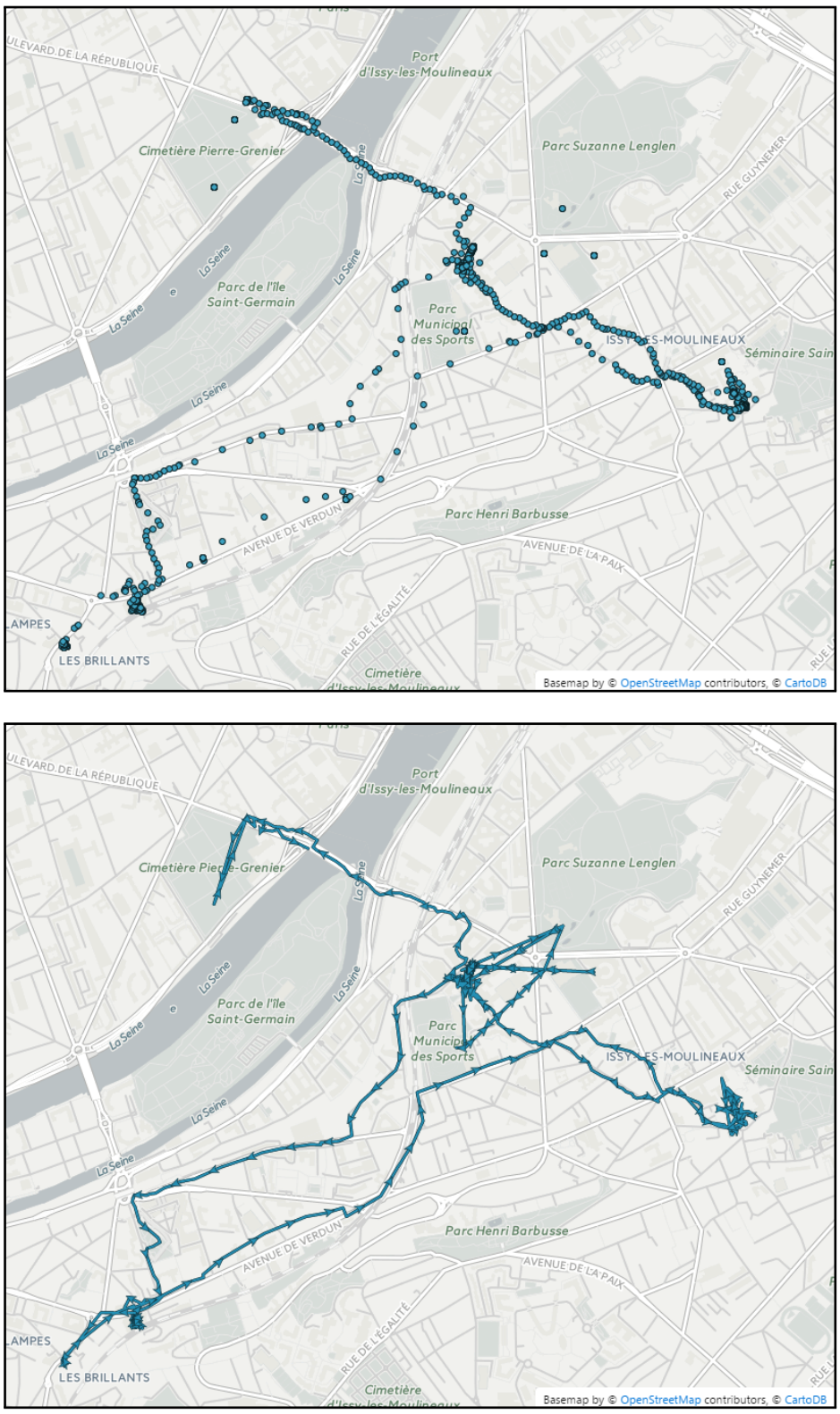


Figure 4.1: Points in Alice's location history for a particular day.

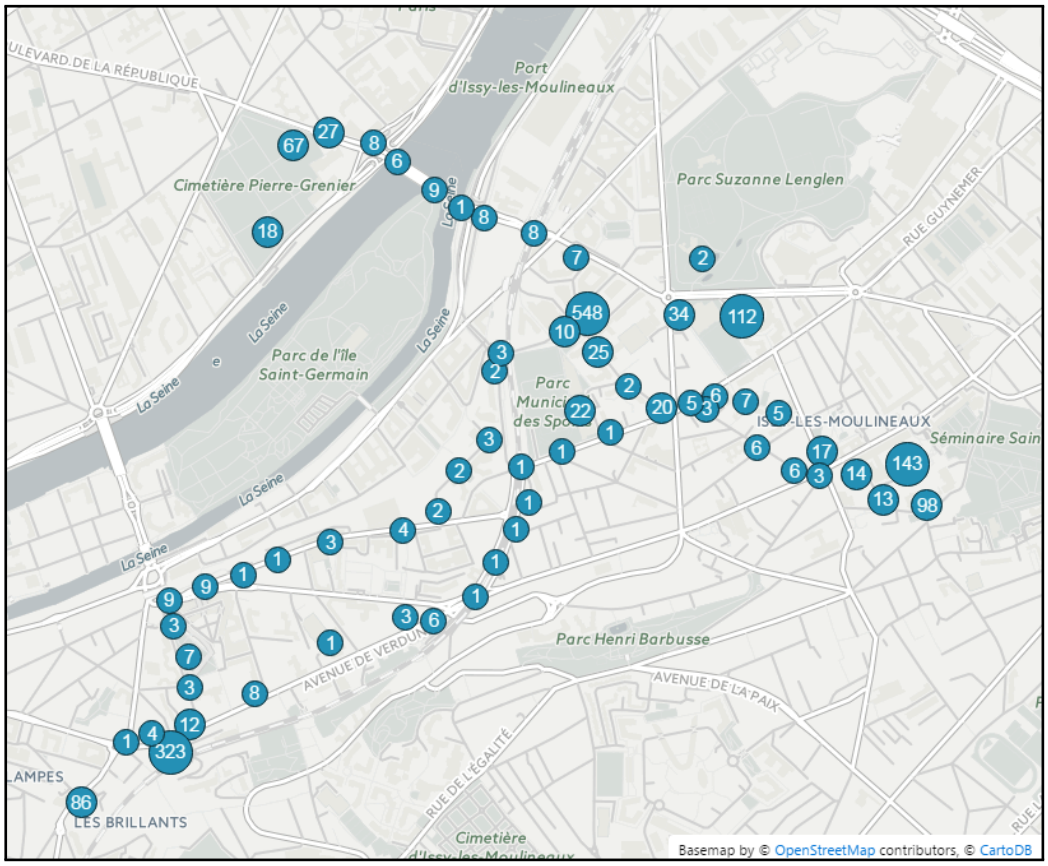


Figure 4.2: Spatial clusters detected in Alice's location history for a particular day, represented as blue circles. The number inside each circle indicates the number of points (locations) within the cluster.

Move clustering Points along routes that Alice frequently takes (e.g., the route she takes to go to her workplace every day) tend to form spatial clusters over time.

High connectivity The distance between time-consecutive points is roughly equal to the product of Alice's speed and the time between measurements. For relatively high measurement frequencies, *stays* are not spatially isolated points but appear to be "connected" via the *move* points in between them.

Varying densities There are significantly fewer points around a place that Alice visited only once (e.g., a restaurant in another town) than around a place where she goes all the time (e.g., home).

Missing locations The assumption that locations are uniformly distributed over the time-axis might not be verified when the device is unable to pinpoint its position over an extended period of time (e.g., when Alice is indoors or underground).

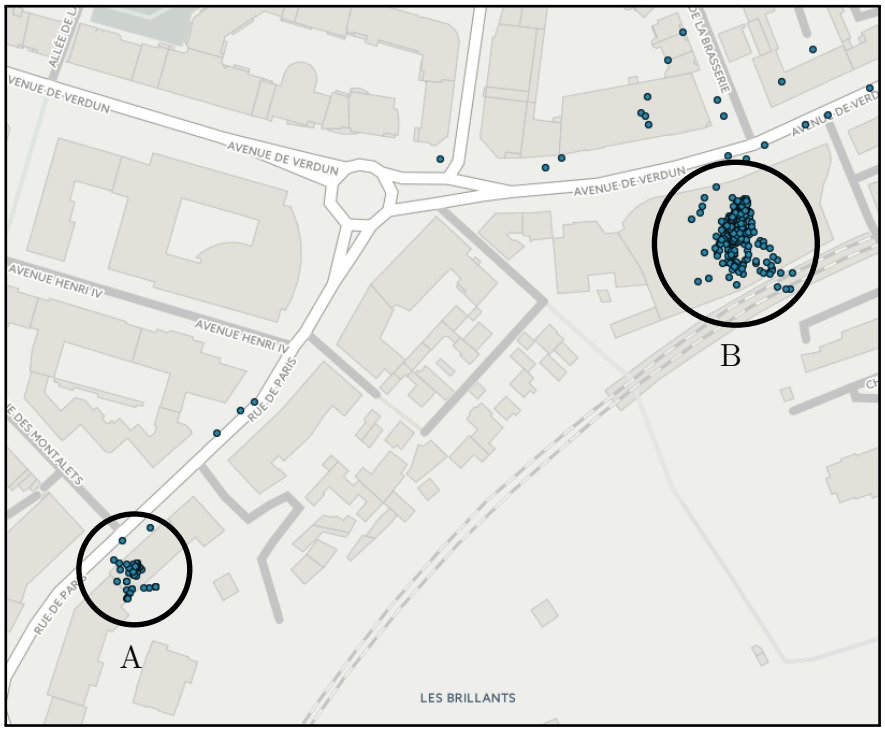


Figure 4.3: A detailed view of two spatial clusters in Alice's location history.

Time-based clustering

An alternative is to extract stays by sequentially scanning locations over the time axis. The idea is to keep track of Alice's position over time and detect when she starts moving. Such a method does not have to deal with the aforementioned difficulties encountered by spatial clustering. In particular, the time spent in a particular place can be monitored and used to counter the density and connectivity issues encountered by spatial clustering. The uniform distribution of locations over the time-axis is no longer a requirement. However, missing locations are a problem for detecting stays when the user is able to move significantly and stay for a long period of time in a place where the device cannot find its position (e.g., a shop inside a mall or a room in a large underground complex).

Other difficulties still remain:

Outliers The mobile device can report an erroneous distant location over a short period of time while the user stays in one place. The stay for such place should not be split.

Location uncertainty The accuracy of locations varies over a large range (cf. Section 3.5): not all positioning technologies are available at all times and the device might intentionally trade-off accuracy in exchange of improved battery life. For this reason, inaccurate locations should be used as much as possible when nothing else is available instead of simply dropping them.

To illustrate the extent of location inaccuracy, we show in Figure 4.4 the same points as in Figure 4.1, but we indicate the accuracy of each point by drawing a

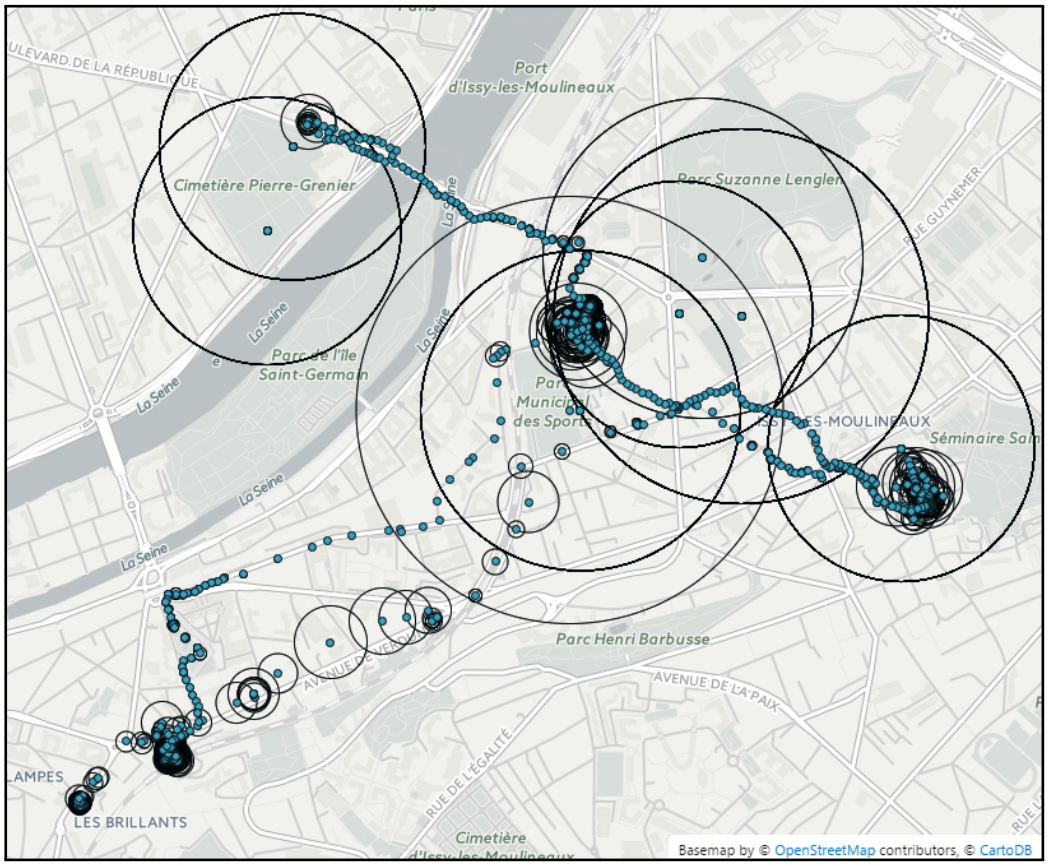


Figure 4.4: Point accuracies in Alice's location history for a particular day. The blue circles are the points, and the black circles indicate the accuracy of each point. The ground radius of a black circle represents the point's accuracy.

black circle whose ground radius is equal to the point's accuracy. In the following section, we present *Thyme*, a stay extraction algorithm that attempts to deal with these issues.

4.3 *Thyme*, the stay extraction algorithm

In this section, we provide a detailed description of *Thyme*, a time-based stay extraction algorithm. The algorithm sequentially scans the locations $(l_i)_{1 \leq i \leq T}$, while maintaining a set of clusters $\mathcal{C}$ in which locations are inserted. $\mathcal{C}$ is initially empty. Each location l_i is added to an existing cluster in $C \in \mathcal{C}$ if it is geographically close (i.e., $d(l_i, C) < \lambda$) and if there exists a location $l_j \in C$ that is recent (i.e., $j = i - 1$ or $t_i - t_j < \theta$). If multiple clusters verify this condition, the location is added to the most recent cluster (i.e., the cluster containing the location with the highest index in the sequence). If no clusters verify this condition, a new cluster $\{l_i\}$ is created and added to $\mathcal{C}$.

Location uncertainty is taken into account by the function d that measures the distance between a location and a cluster. To do so, we represent a location l_i as a two-dimensional unimodal normal distribution $P_i \sim \mathcal{N}(\mu_i, u_i^2)$ where the

expectation $\mu_i = (x_i, y_i)$ represents p_i in a projected coordinate system and the standard deviation is equal to u_i. The assumption of a normally distributed error is typical in the field of processing location data (cf. Section 3.5).

A cluster $C = \{l_{r_1}, \ldots, l_{r_n}\}$ is also represented by a normal distribution $P(C) \sim \mathcal{N}(\mu, \sigma^2)$. μ is the weighted arithmetic mean of the locations' projected coordinates[1] (x_{r_i}, y_{r_i}) weighted by their inverse accuracy squared $1/u_{r_i}^2$. The variance σ^2 is the harmonic mean of location accuracies squared $1/u_{r_i}^2$.

$$\mu = \sum_{i=1}^{n} \frac{(x_{r_i}, y_{r_i})}{u_{r_i}^2} \left(\sum_{i=1}^{n} \frac{1}{u_{r_i}^2} \right)^{-1} \qquad \sigma^2 = \left(\sum_{i=1}^{n} \frac{1}{u_{r_i}^2} \right)^{-1}$$

In particular, for a cluster consisting of a single location l_i, $P(\{l_i\})$ is equal to the location's normal distribution P_i.

To define d, we use the Hellinger distance (Hellinger 1909), a statistical distance used to quantify the similarity between two probability distributions. For two normal distributions $P_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $P_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$, the squared Hellinger distance is equal to:

$$H^2(P_1, P_2) = 1 - \sqrt{\frac{2\sigma_1\sigma_2}{\sigma_1^2 + \sigma_2^2}}\, e^{-\frac{1}{4}\frac{\delta(\mu_1,\mu_2)^2}{\sigma_1^2+\sigma_2^2}} \in [0, 1],$$

where $\delta(\mu_1, \mu_2)$ is the geographical distance between cluster centers. Finally, the distance $d(l, C)$ between a location l and a cluster C is defined by:

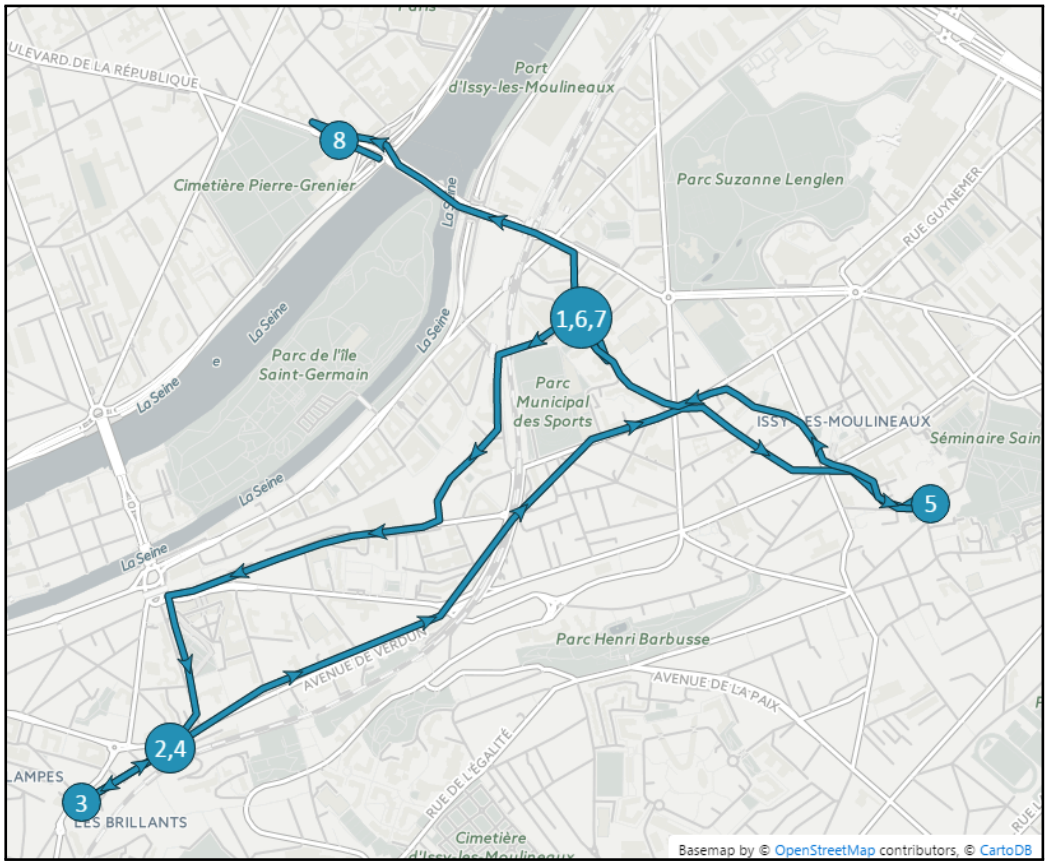
$$d(l, C) = \max_{k \in C \cup \{l\}} H(P(\{k\}), P(C \cup \{l\})).$$

The location l is added to C when $d(l, C)$ is below a certain threshold λ. This distance takes into account the overall uncertainty of points that form the cluster, making it tolerant to measurement variability and potential outliers. The time span of a cluster is defined by the time interval between the oldest and most recent location in the cluster: $T(C) = [\min_{l_i \in C} t_i, \max_{l_j \in C} t_j]$.

Output When $\theta = 0$, each iteration of the above procedure either merges l_i into the most recent cluster (i.e., the one with l_{i-1}), or creates a new singleton cluster $\{l_i\}$. Thus, when $\theta = 0$, the procedure creates a non-overlapping sequence of clusters over the time-axis. In this case, each cluster C that lasts more than a certain time threshold τ (i.e., whose time span length is greater than τ) defines a stay. The mean and standard deviation of $P(C)$ defines the center and uncertainty of the stay. The resulting sequence of stays is the output of *Thyme*. The result of running *Thyme* on Alice's location history is shown in Figure 4.5.

Filtering locations from multiple devices Sometimes, the user's location history may come from multiple devices, e.g., a telephone and a tablet. As discussed in Section 3.5, this may be problematic if each location is not associated with the device that generated it using for instance a device identifier. For

[1] The locations' geographic coordinates are projected on an Euclidian plane locally approximating distances and angles on Earth around some location in C (e.g., l_{r_1}).



1	00:01 - 10:18		5	17:45 - 19:15	
	12 minutes (12.03′) - 2 km (2.8 m/s)			23 minutes (22.81′) - 1.1 km (0.8 m/s)	
2	10:30 - 12:24		6	19:38 - 19:56	
	a minute (1.21′) - 261 m (3.6 m/s)			a minute (1.23′) - 39 m (0.5 m/s)	
3	12:25 - 13:52		7	19:57 - 20:41	
	a minute (0.98′) - 254 m (4.3 m/s)			13 minutes (12.71′) - 968 m (1.3 m/s)	
4	13:53 - 17:18		8	20:54 - 23:58	
	27 minutes (27.23′) - 2.3 km (1.4 m/s)			a minute (1.05′) - 107 m (1.7 m/s)	

Figure 4.5: The result of running *Thyme* on Alice's location history for a particular day ($\lambda = 0.95$, $\tau = 15\,\text{min}$). In the top figure, circles represent stays (i.e., clusters longer than 15 minutes), and the number inside each circle indicates the stay's position in chronological order. Consecutive stays are connected by a line formed by the sequence of clusters in-between, yielding simpler lines than those in Figure 4.1. The bottom figure gives the times and duration of each stay and move. For each move, the traveled distance and average speed are also given.

example, if the user leaves her tablet at home while she goes to her workplace, the location history may look as if she were "going back and forth" from home to her workplace multiple times over the course of the day. Google Timeline is subject to this issue. To counter it, we devised a location filtering algorithm that attempts to fix this problem by assuming that locations may come from up to two devices: one device that the user carries with her at all times, and one which may or may not be with her. The latter device is assumed to be static when the user is not carrying it.

To do so, we first run *Thyme* with $\theta = \theta_f > 0$, and filter out clusters shorter than a certain time threshold $\tau = \tau_f$. This results in possibly time-overlapping clusters. Some of these overlaps may represent moments where the user was at her workplace while her secondary device (e.g., her tablet) stayed at home. Then, a trellis[2] is generated from the sequence of locations. The n-th time slice in the trellis represents the n-th location in the sequence. The time slice representing location l consists of nodes representing one of the following three kinds of states:

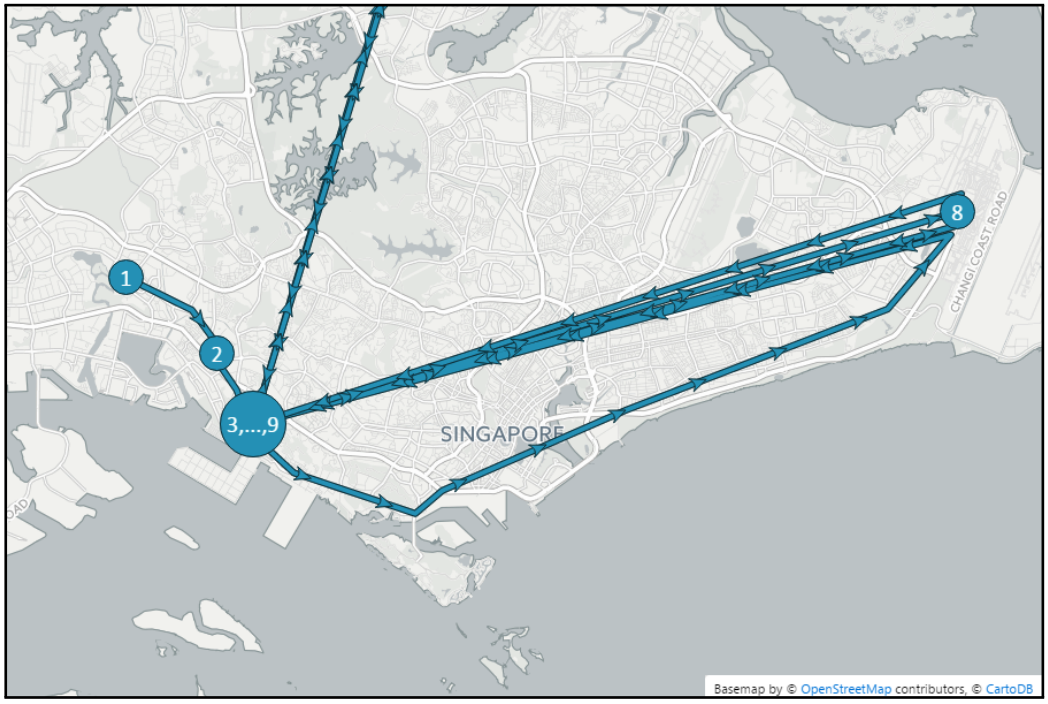
1. Both devices are in location l.
2. l is the location of the device carried by the user. The secondary device is located in a cluster C that spans over a period that includes l.
3. l belongs to a cluster C. The location of the secondary device is l. The latest location in which the device carried by the user was located is l' (l' is before l in the location sequence).

Edges are weighted in part by the distance that the user needs to travel to go from one state to another. For instance, when going into a state of type 3, the user does not move (she stays at a location l'). When going into a state of type 1 from a state of type 1, the distance between the locations of each state is the distance traveled. The minimum weight path from the earliest time slice to the latest represents the path traveled by the user. States of type 3 are filtered out, yielding a sequence of filtered locations. To extract stays, *Thyme* can be run once again on the filtered locations with $\theta = 0$. In our experiments we set θ_f to 600 min and τ_f to 15 min for the filtering part. An illustration of this problem, and the impact of this filter is shown in Figure Figure 4.6.

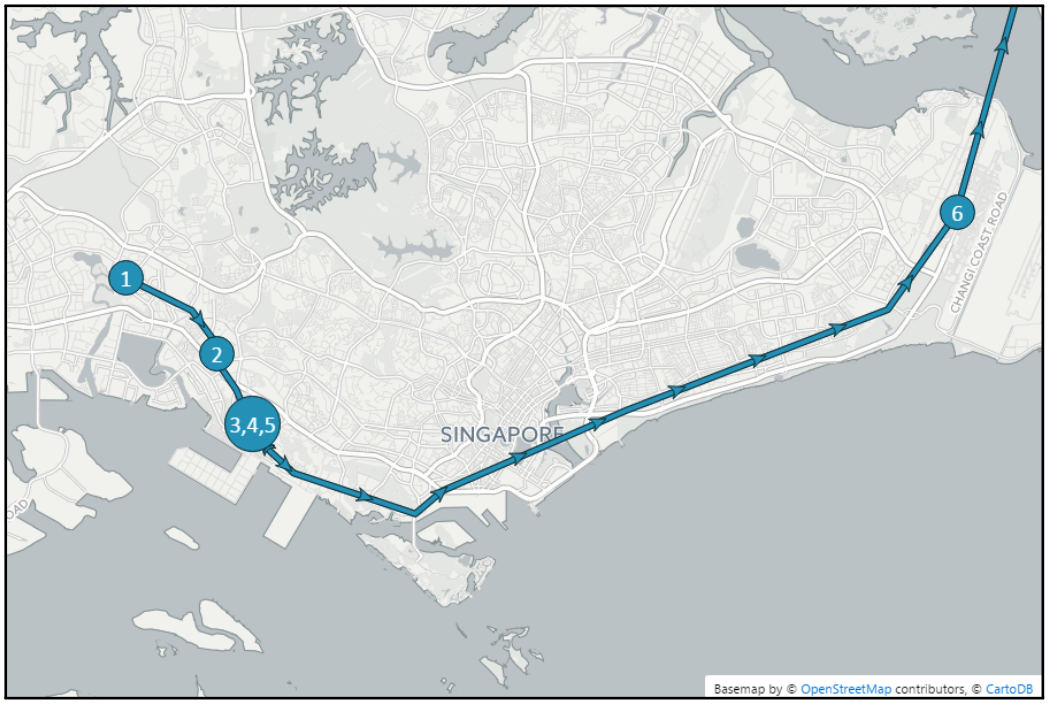
4.4 Evaluation

We evaluated *Thyme*'s stay recognition performance on the location history of a user, Bob, who had continuously collected his location over about 5 years. For the collection, he used the Google Timeline and Location History application presented in Section 3.5 and provided us a dump of his location history. Bob annotated 15 randomly chosen days (among 1676) in his location history. For each day, Bob was presented with an interface showing the raw locations of

[2] A trellis is a graph whose nodes are partitioned into a set of time slices, ordered by time, and whose edges are pairs of nodes in adjacent time slices. The earliest and latest time slices in the trellis only have one node.



(a) No multi-device filter has been applied. The movement is erratic and goes back and forth from Alice's workplace to some distant place (stays 3 to 9).



(b) The multi-device filter has been applied. The movement is smooth.

Figure 4.6: The stays extracted by *Thyme* from Alice's location history during a period when Alice had left her tablet at her workplace in Singapore (stay 5 in Figure 4.6b) while she traveled abroad.

Method	#D	DΘ	Prec.	Recall	F1
Thyme15	33 %	0.6 %	91 %	98 %	95 %
Thyme10	46 %	0.9 %	83 %	98 %	90 %
Thyme5	63 %	1.0 %	62 %	100 %	77 %
Google	18 %	N/A	88 %	89 %	88 %

Table 4.1: Stay extraction evaluation on Bob's dataset.

each day on a map, as well as the output of *Thyme* for a λ of 0.95, and for which he varied the stay duration threshold τ (5, 10, and 15 minutes). Since Bob locations come from a smartphone and tablet, we ran the location filtering procedure presented at the end of Section 4.3. For each day, Bob counted the number of stays that he thought were true, false, or missing. He also evaluated the stays extracted by his Google Timeline. The exact definition of an *actual stay* was left to Bob, and the reliability of his annotations were dependent on his recollection. In total, Bob found 64 *actual stays*. Among the true stays output by each method, some appeared multiple times. Consequently, we counted the number of true stay duplicates and their resulting move duration. For instance, an *actual stay* of 2 hours appearing as two stays of 29 and 88 minutes with a short 3-minute move in-between, would count as 1 true stay, 1 duplicate, and 3 minutes of duplicate duration. Table 4.1 shows the resulting precision and recall for each method, as well as the duplicate ratio #D (the ratio of the number of duplicates to the number of true stays) and the duplicate duration ratio DΘ (the ratio of the duplicate duration to the total duration of true stays). Figure 4.7 shows an example of a journey that Bob evaluated. Overall, our technique products results comparable to Google Timeline for stay extraction, with better precision and recall, but a tendency to produce duplicates.

4.5 Related work

Stay extraction The stay extraction problem is not new. While they are sometimes referred to as *visits* (Lv, L. Chen, and G. Chen 2012) or *places* (Ashbrook and Starner 2003a), the terms *stays* or *stay points* remain the most widely used ones (Kang et al. 2004; Hariharan and Toyama 2004; Li et al. 2008; Nishida, Toda, and Koike 2015). As noted by Nishida, Toda, and Koike (2015), state-of-the-art algorithms derive stays by scanning the location sequence in order. Among them, variations of *time-based clustering* (Kang et al. 2004) have been widely utilized (Hariharan and Toyama 2004; Li et al. 2008; Lv, L. Chen, and G. Chen 2012). Time-based clustering algorithms regard consecutive locations within a certain distance from each other as candidate stays, and consider for stays those longer than a certain duration threshold. Kang et al. (2004)'s algorithm keeps track of one cluster at a time, and either incrementally adds new locations to it or drops it to consider an entirely new cluster. In an attempt to alleviate the "discontinuous GPS sampling problem", Lv, L. Chen, and G. Chen (2012) extended Kang et al. (2004)'s algorithm by making it track a second cluster that

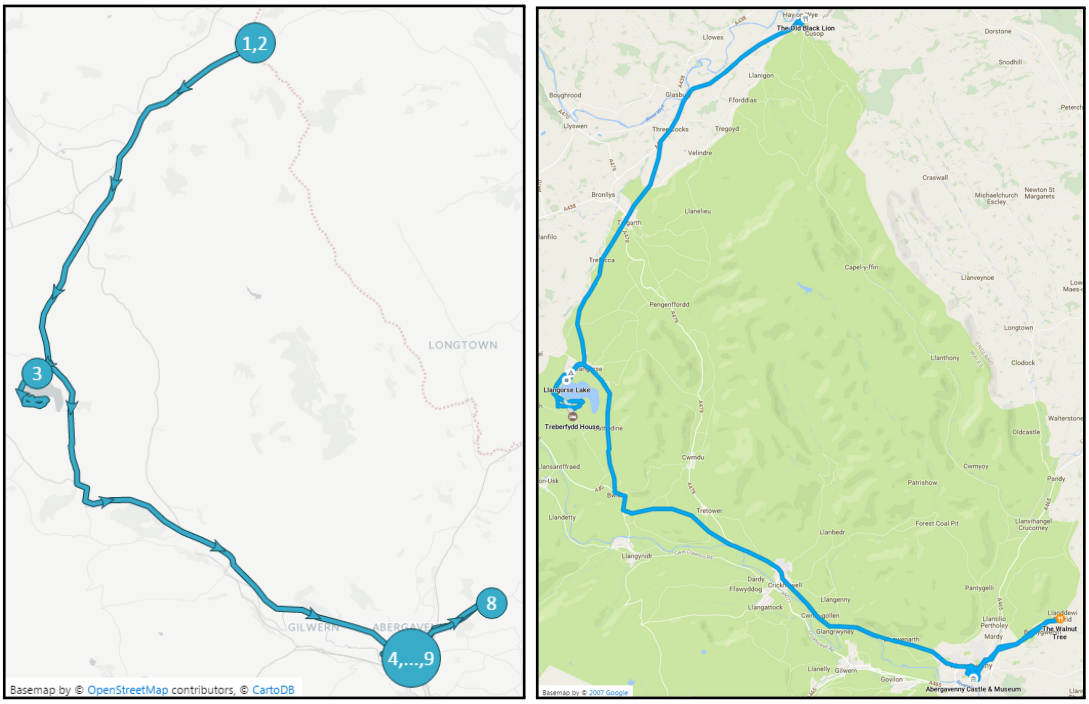


Figure 4.7: On the left, the result of running *Thyme* on a day in Bob's location history ($\lambda = 0.95$, $\tau = 15$ min). On the right, the result given by Google Timeline. Thyme found three duplicate stays (stays 2, 5, and 6), while Google only found one duplicate (stay 2). On the other hand, Google found two false stays (an 11-minute and a 22-minute stay) around the lake next to stay 3. When adjusting τ, Thyme also finds the 11-minute stay but never the 22-minute stay.

is possibly later merged with the first cluster based on the time and geographical distance between both clusters. Nishida, Toda, and Koike (2015)'s algorithm goes a little further: For each location l, a cluster is formed from the locations in l's temporal vicinity that are also within a small geographical distance of l. Time-overlapping clusters are merged, yielding *stay regions* instead of *stay points*. *Thyme* is similar to (Kang et al. 2004)'s algorithm, except that it uses a different location-cluster distance and different merge functions. *Thyme* has also been adapted to process locations collected from two devices without device identifiers.

Significant places Stays extracted from a location history can be used to derive *significant places*. A *significant place* is collection of *stays* representing the same place (e.g., home, the user's workplace, or the gym) (Ashbrook and Starner 2003a; Hightower et al. 2005; Adams, Phung, and Venkatesh 2006; Li et al. 2008; Lv, L. Chen, and G. Chen 2012). For deriving *significant places*, classic spatial clustering methods (cf. Section 4.2) can be applied on the set of stays. This derivation has been used to compare and recommend places to multiple users (Li et al. 2008), for modeling and prediction (Ashbrook and Starner 2003a; Adams, Phung, and Venkatesh 2006; Liao 2006), and for visualization (Otten et al. 2015).

4.6 Conclusion

We have proposed an algorithm for performing *stay extraction* from the user's location history. Our algorithm is the first, to our knowledge, to exploit the accuracy of locations. In summary, we provide the following contributions:

1. A distance function between a cluster and a location that takes into account location accuracy.

2. An incremental merge function between a cluster and a location that takes accuracy into account.

3. The ability to process a history of locations without device identifiers from up to two devices of the same user, assuming that no device moves unless the user is carrying it.

4. The implementation of this algorithm in a much larger system, an open-source personal information management system, that we present in Chapter 6.

Limitations The heterogeneity of positioning technologies and the algorithm's usage of all locations whatever the uncertainty helps detecting and positioning stays in places where satellite navigation is non-existent (e.g., inside buildings). However, it may also be a source of errors. This may happen when the user travels in a mode of transportation with poor network connectivity (e.g., a high speed train or underground transit). Network positioning technologies may report the same inaccurate location for a few minutes, leading to the detection of incorrect stays. Another difficulty is handling incorrectly split stays in presence of outliers. The algorithm by (Nishida, Toda, and Koike 2015) counters this by using *stay regions* instead and merging time-overlapping clusters, but at the expense of a new tunable parameter and the issue that highly frequent movements from one place to another may be incorrectly labeled as a stay. Finally, it is not yet clear how the λ parameter should be tuned, perhaps even automatically from user preferences and feedback.

Discussion and future directions Most documented stay extraction methods have been designed to deal with location histories generated from satellite-based positioning. Some of them even exploit the nature of this technology and consider stays to be the moments when satellite-based positioning is not available (e.g., inside a building) (Ashbrook and Starner 2003a). Recent developments in positioning technologies have greatly enhanced the availability of device positioning (e.g., indoors using Wi-Fi-based positioning) and improved the accuracy of existing ones (e.g., new satellite constellations in addition to GPS: GLONASS, BDS, and Galileo). These developments not only facilitate location history analysis by rendering issues such as the "discontinuous GPS sampling problem" less relevant, but also bring new opportunities. Detecting stays can use lower-level sensor information, to filter out location outliers through sensor fusion (e.g.,

when the accelerometer detects no movement), and to fine tune start and end times by monitoring changes in sensor information (change in network signal level, movement detection) (Kim et al. 2010). In fact, the problem boils down to whether the objective is to determine the geographic position of a stay at all costs, or if it is satisfactory to simply identify and distinguish them using all other available information (e.g., detecting that Alice is in the cafeteria even though we cannot tell her exact position). In the next chapter, we see how mobile device sensor information can be used to infer the itinerary followed by the user: the transportation modes, and the transit lines taken.

Chapter 5

Spatiotemporal knowledge: Itinerary recognition

Part of the contents of this chapter have been published in (Montoya and Abiteboul 2014; Montoya, Abiteboul, and Senellart 2015), where we demoed and presented a short evaluation of a first version of the system. In this chapter, we present the new version. Steve Samspon, from ENGIE, was a notable contributor in the development of this new version.

In the previous chapter, we saw how Alice's day could be segmented into *stays* and *moves*. In this chapter, we present *Movup*, a system that we designed that uses a wide range of mobile sensor information, geographic information about roads and railways, and public transportation schedule information (including associated geographic information) to infer the itinerary, possibly multimodal, followed by Alice during a *move* segment. *Movup* is a reworked version of *Hup-me* (Montoya and Abiteboul 2014; Montoya, Abiteboul, and Senellart 2015), the first version of the system. By design, the different aspects of the first version could not be evaluated separately, to measure for instance the impact of each sensor and in particular the impact of geographic information. Also, the system's execution time was poor due to the complexity of the continuous state space it worked with. To address this problem and some other shortcomings, we restructured and simplified the model and reimplemented the system to keep only the novel aspects of the problem it tries to solve. These are discussed in this chapter.

5.1 Introduction

The democratization of connected and sensor-rich personal mobile devices has increased the demand for context-aware commercial applications taking advantage of the information they are able to generate. Technology is still far from taking full advantage of mobile device sensors to understand the users' daily movements in urban environments. One requirement of transit navigation mobile applications is to provide real-time tracking within the network for the user, so as to alert her,

for instance, when she is on a bus, moments before the bus arrives at the stop she should get off at.

In our setting, we would like to recognize the different transportation modes used by Alice, including public transportation routes and transfers from one mode to another, in an offline fashion, i.e., after the whole trip has been observed, so as to an enrich her activity timeline in her knowledge base for later use. We call this task *itinerary recognition*. When transfers between different modes are possible, the recognized itinerary may be *multimodal*, which is an itinerary with a least one transfer between two distinct modes.

In previous works, the description of modes and routes has varied in form and complexity. A simple description is to find a mode among `foot`, `bicycle`, and `vehicle`. A more complex description may include more refined modes such as `car`, `metro`, and `airplane`, as well as extra dimensions for identifying the route (e.g., bus line 123, flight number XYZ130), the stops (e.g., Gare du Nord, Penn Station, Berlin Hauptbahnhof) and the roads (e.g., Lombard Street, Abbey Road, Avenue des Champs-Elysées).

Movup performs *multimodal itinerary recognition* using sensor data from Alice's mobile device (smartphone) for the set of modes `foot`, `bicycle`, `car`, `bus`, `metro`, `train`, and `tram`. For each unimodal segment of Alice's itinerary where Alice traveled on board of a public transportation vehicle (i.e., a bus, a metro, a train, or a tram), the system identifies the route as well as the stops where she boarded and alighted. To recognize such segments, *Movup* uses public transportation schedule information, from which it computes a set of candidate scheduled trips for various routes.

In contrast with its predecessor, *Movup* does not try to identify the roads and railways in Alice's itinerary, as doing so in our setting substantially increases the complexity of the problem and makes evaluating the system's performance much harder without that much benefit. Instead, the system only determines the locations (and times) where changes of transportation mode happen. When the system recognizes a segment in which Alice traveled on board of a public transportation vehicle, the roads or railways for that segment are implicitly determined by the route to which the vehicle belongs. However, for vehicles which can deviate from their normal route (e.g., buses in the case of road closures), *Movup* does not determine the exact deviation.

The system runs a supervised learning algorithm that looks for the itinerary of maximum probability given a set of mobile sensor observations. The algorithm works in stages that progressively lift uncertainty by deriving high-confidence estimates first, such as knowing whether the user is still or is moving, and use these estimates as facts in the next stage to guide the search within a smaller state space. In this setting, a critical aspect is that of the data that the system uses. First, the system has access to geographic data (roads and railways) obtained from OpenStreetMap (OSM) (Haklay and Weber 2008) that it uses to build a transportation network model for the different recognizable modes. Each network describes the roads or railways that a vehicle of the network's mode can travel on. Then the system uses public transportation data (routes and schedules) published online by transportation agencies in GTFS (Google 2015a) format.

Since geographic information about public transportation routes in GTFS format is usually limited to the locations of stops, the system uses the transportation networks built from OSM data to automatically infer the path of each route in the transportation network built for the route's mode. This enriches public transportation routes with information that is useful to the recognition task such as the position of underground passages and tunnels along the routes. Finally, the system processes mobile sensor data based on their kind (*locational*, *dynamic* and *contextual*, cf. Section 3.5) and their characteristics (e.g., frequency, accuracy). Accordingly, a preprocessing stage is used that reduces redundancy and noise, then the result is transformed into a sequence of feature vectors providing information in numerical form that is relevant to the recognition task, such as the user's distance to the nearest highway or her average speed over the previous few seconds at any given moment. Difficulties arise from lack of data (e.g., lack of positioning inside the metro system), from too much data (e.g., combination of possibly conflicting location information, overlapping public transportation lines), and inaccuracies or imprecisions in the data (e.g., map errors, imprecise location).

We evaluated the performance of the system using data recorded from users traveling in the Paris metropolitan area. For this, we provided them with the *Hup-me mobile* application described in Section 3.5 to collect sensor information from their smartphone. We manually annotated the journeys that had been recorded. Our annotations included the transportation modes as well as the public transportation routes that had been used. OSM data for the Paris area and GTFS data published by transportation agencies operating in the area were acquired. The Paris metropolitan area is rich in public transportation modes: sub-urban train, metro, bus, and tram. We compared the output of the algorithm to the annotations that we assumed to be correct.

This chapter is organized as follows: Section 5.2 introduces the notion of a transportation network and describes the construction of transportation networks from OSM data. Section 5.3 describes the representation of public transportation *routes* and their schedules, based on GTFS, and shows how the transportation networks obtained from OSM data can be used to enrich the representation of routes with information that is useful for itinerary recognition. Section 5.4 describes the preprocessing and extraction of features from mobile sensor data. Section 5.5 formally defines the *itinerary recognition* problem, and Section 5.6 describes the algorithm to solve it. Section 5.7 discusses an evaluation of the algorithm. Section 5.8 discusses the related work and Section 5.9 concludes.

5.2 Transportation networks

In this section we introduce the notion of a transportation network and describe its construction from OpenStreetMap (OSM) data (Haklay and Weber 2008). To do so, we first present standard definitions for geospatial information and formalize the notion of a *spatial network*.

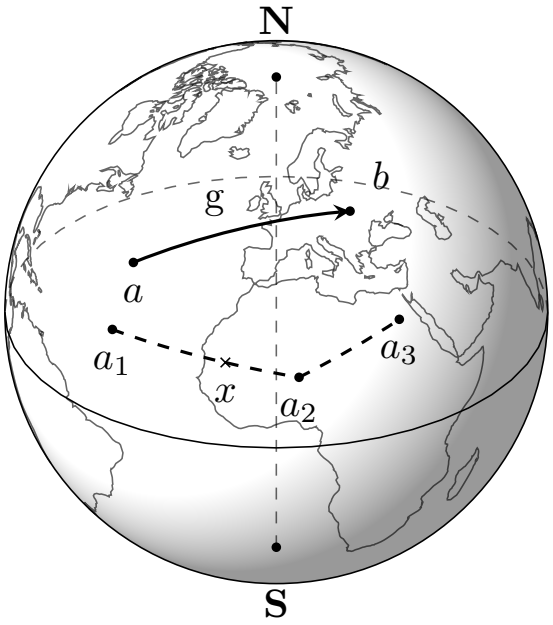


Figure 5.1: Geodesics and trails: a, b, a_1, a_2, a_3 are points, the curve denoted g represents the geodesic from a to b, and the two dashed curves represent the trail (a_1, a_2, a_3). x is the point on the geodesic from a_1 to a_2 at 3000 km from a_1 ($x = \mathcal{H}((a_1, a_2, a_3))(3000\,\text{km})$).

Geographic information

We model the shape of the Earth as a spheroid. A *point* on Earth is represented by a pair of geographic coordinates, called *latitude* and *longitude*, in **Point** $= (\pi, \pi] \times \left[\frac{\pi}{2}, \frac{\pi}{2}\right]$. A *trail* $(a_1, \ldots, a_r)$ is a finite sequence of 2 or more elements of **Point**, such that $a_i \neq a_{i+1}$ for all i. The set of trails is denoted **Trail**, and we call *geography* an element of **Geography** = **Point** $\cup$ **Trail**. Given two distinct points a, b in **Point**, the *geodesic* from a to b is the shortest curve from a to b on the spheroid. A *point on* a trail $(a_1, \ldots, a_r)$ is a point on the geodesic from a_i to a_{i+1} for some i. The length of the geodesic from a to b is called the *distance* between a and b and is denoted $\text{dist}(a, b)$. The length of a trail $(a_1, \ldots, a_r)$ is defined as:

$$\text{length}((a_1, \ldots, a_r)) = \sum_{i=1}^{r} \text{dist}(a_i, a_{i+1}).$$

The trail $(a_1, \ldots, a_r)$ defines a curve

$$\mathcal{H}((a_1, \ldots, a_r)) : [0, \text{length}((a_1, \ldots, a_r))] \to \textbf{Point}$$

as follows. For each $i \in \{1, \ldots, r-1\}$ and $l \leq \text{dist}(a_i, a_{i+1})$, $\mathcal{H}((a_1, \ldots, a_r))(\text{length}((a_1, \ldots, a_i)) + l)$ is the point at distance l from a_i on the geodesic from a_i to a_{i+1}.

The distance between points is extended to a distance between two *geographies* $g, g' \in$ **Geography** by:

$$\text{dist}(g, g') = \min\{\text{dist}(x, x') \mid x \text{ point on } \mathcal{H}(g), x' \text{ point on } \mathcal{H}(g')\}.$$

Figure 5.1 illustrates the concepts of geodesics and trails.

$\mathcal{E}_0$	$\texttt{from}_0$	$\texttt{to}_0$
5	1	4
6	1	2
7	1	2
8	2	3
9	3	4

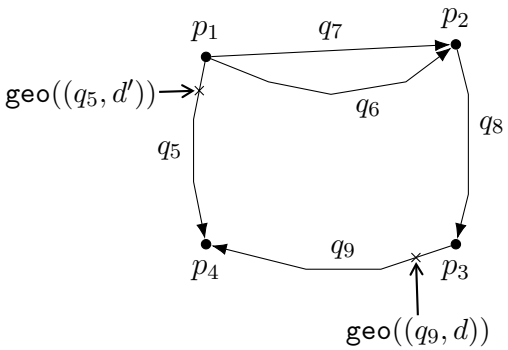


Figure 5.2: A spatial network example: $\mathcal{G}_0 = \{\mathcal{N}_0, \mathcal{E}_0, \texttt{geo}_0, \texttt{from}_0, \texttt{to}_0\}$. It has nodes $\mathcal{N}_0 = \{1, 2, 3, 4\}$ and edges $\mathcal{E}_0 = \{5, 6, 7, 8, 9\}$. The table on the left gives the functions $\texttt{from}_0$ and $\texttt{to}_0$. The figure on the right represents the geographies of nodes and edges projected into a plane: $p_i = \texttt{geo}_0(i)$ is the geography of node i, represented as a black disk, and q_j is the trail of edge j, represented as a line from $p_{\texttt{from}_0(j)}$ to $p_{\texttt{to}_0(j)}$ with an arrow oriented toward the geography of $\texttt{to}_0(j)$. d and d' are offsets over the trails q_9 and q_5: the geographies of locations $(9, d)$ and $(5, d')$ are represented on the figure.

Spatial networks

A *spatial network* is a tuple $\mathcal{G} = (\mathcal{N}, \mathcal{E}, \texttt{geo}, \texttt{from}, \texttt{to})$ s.t.

- $\mathcal{N}, \mathcal{E}$ are disjoint finite sets (called respectively the *nodes* and *edges*).
- For $n \in \mathcal{N}$, $\texttt{geo}(n) \in$ **Point**.
- For $e \in \mathcal{E}$, $\texttt{from}(e)$ and $\texttt{to}(e)$ belong to $\mathcal{N}$, and $\texttt{geo}(e) \in$ **Trail** is a trail from $\texttt{geo}(\texttt{from}(e))$ to $\texttt{geo}(\texttt{to}(e))$.

By extension, the length of an edge is defined as the length of its trail. Let $e \in \mathcal{E}$. For each $d \in [0, \text{length}(e)]$, we say that $l = (e, d)$ is a *location* in $\mathcal{G}$. The value d is called the *offset* of that location. By extension, we denote by $\texttt{geo}(l)$ the point on the curve of $\texttt{geo}(e)$ given by $\mathcal{H}(\texttt{geo}(e))(d)$. The set of locations is denoted **Location**$(\mathcal{G})$.

Figure 5.2 illustrates the notion of a spatial network.

Transportation networks

A spatial network captures the notion of "physical infrastructure": a spatial network can be used to model places on Earth (via spatial nodes) and their physical connections to one another (via spatial edges and their corresponding trails) such as metro platforms and metro line tunnels. However, it does not capture movement constraints with respect to a transportation mode. For instance, by foot, one may decide to do a U-turn between two spatial nodes, and reverse one's direction, whereas one cannot do that in a train or on a freeway. Also, turning left at an intersection might be forbidden for a car, while it may be

allowed for a bus. A *unimodal transportation network*, which we introduce next, specifies such constraints for a particular transportation mode.

We define the auxiliary notion of a *path* in a spatial network $\mathcal{G} = (\mathcal{N}, \mathcal{E}, \texttt{geo}, \texttt{from}, \texttt{to})$. It uses $\mathcal{D} = \{\blacktriangleleft, \blacktriangleright\}$, a set of two symbols, respectively called *backward* and *forward*, specifying the two directions on an edge. An edge $e \in \mathcal{E}$ that can be traversed in the orientation of the edge is denoted $(e, \blacktriangleright)$, and $(e, \blacktriangleleft)$ for the reverse orientation. Let $\hat{\mathcal{E}} = \{(e, \delta) \mid \delta \in \mathcal{D}\}$ be the set of *edge traversals*. We extend `from` and `to` to an edge traversal (e, δ) by:

$$\begin{aligned} \texttt{from}(e, \blacktriangleleft) &= \texttt{to}(e) & \texttt{to}(e, \blacktriangleleft) &= \texttt{from}(e) \\ \texttt{from}(e, \blacktriangleright) &= \texttt{from}(e) & \texttt{to}(e, \blacktriangleright) &= \texttt{to}(e). \end{aligned}$$

A *unimodal transportation network* over a spatial network $\mathcal{G}$ is a tuple $\mathcal{T} = (\mathcal{G}, M, U, V)$ such that:

- $M \subset \hat{\mathcal{E}}$ is the set of admissible edge traversals.
- U is a set of *transitions* of the form $(e_1, \delta_1), (e_2, \delta_2) \in M^2$, with $\texttt{to}(e_1, \delta_1) = \texttt{from}(e_2, \delta_2)$ and .
- V is a set of *in-edge transitions* of the form $(e, \delta_1), (e, \delta_2) \in M^2$.

In other words, U specifies possible transitions when switching from an edge to another at a node, and V specifies under which conditions one can reverse one's direction while staying on the same edge.

Example 5.2.1 *Consider the spatial network $\mathcal{G}_0$ defined in Figure 5.2. Let us define a transportation network $\mathcal{T}_0 = (\mathcal{G}_0, M_0, U_0, V_0)$ over $\mathcal{G}_0$. Let M_0 be defined by:*

$$M_0 = \{(5, \blacktriangleleft), (6, \blacktriangleright), (6, \blacktriangleleft), (7, \blacktriangleright), (8, \blacktriangleright), (9, \blacktriangleright), (9, \blacktriangleleft)\}.$$

Then the following statements:

$$\begin{gathered} (6, \blacktriangleright), (8, \blacktriangleright) \in U_0 \\ (7, \blacktriangleright), (8, \blacktriangleright) \notin U_0 \\ (9, \blacktriangleright), (5, \blacktriangleleft) \in U_0 \end{gathered}$$

ensure that the network allows transitions from edge 6 to edge 8 at node 2 but does not allow transitions from edge 7 to 8 at node 2. This is a turn restriction. Also, it ensures we are able to go from edge 9 to edge 5 at node 4. For edge 9, direction reversal is disallowed by ensuring:

$$\begin{gathered} (9, \blacktriangleright), (9, \blacktriangleleft) \notin V_0 \\ (9, \blacktriangleleft), (9, \blacktriangleright) \notin V_0. \end{gathered}$$

Edge 9 works like a highway.

Finally, a *location* in $\mathcal{T}$ is a pair (ϵ, d), where $\epsilon = (e, \delta) \in M$ is an admissible edge traversal and d is an offset in $[0, \text{length}(e)]$. The set of locations in $\mathcal{T}$ is denoted $\textbf{Location}(\mathcal{T})$. We extend `geo` to locations in $\mathcal{T}$ by $\texttt{geo}(((e, \delta), d)) = \texttt{geo}((e, d))$.

In Example 5.2.1, $((9, \blacktriangleright), \delta)$ is a location in $\mathcal{T}_0$.

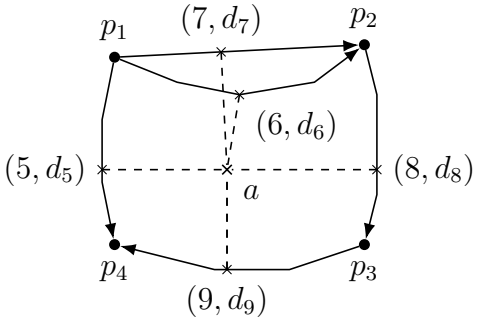


Figure 5.3: A point a and its projection $\{(5, d_5), (6, d_6), (7, d_7), (8, d_8), (9, d_9)\}$ on the spatial network $\mathcal{G}_0$ defined in Figure 5.2.

Paths in a transportation network For $r \geq 0$, a *path* in $\mathcal{T}$ is a sequence $\rho = ((\epsilon_0, d_0), (\epsilon_1, d_1), \ldots, (\epsilon_r, d_r))$ where for each i, (ϵ_i, d_i) is a location in $\mathcal{T}$ (with $\epsilon_i = (e_i, \delta_i)$), and for $i < r$ either:

(edge transition) $(\epsilon_i, \epsilon_{i+1}) \in U$ and

$$\begin{aligned} \delta_{i+1} = \blacktriangleright &\implies d_{i+1} = 0 \\ \delta_{i+1} = \blacktriangleleft &\implies d_{i+1} = \text{length}(e_{i+1}), \end{aligned}$$

(in-edge transition) $(\epsilon_i, \epsilon_{i+1}) \in V$ and $d_i = d_{i+1}$,

(in-edge move) $\epsilon_i = \epsilon_{i+1}$ and either

$(d_{i+1} \geq d_i$ and $\delta_i = \blacktriangleright)$, or $(d_i \geq d_{i+1}$ and $\delta_i = \blacktriangleleft)$.

The set of paths in $\mathcal{T}$ is denoted Paths($\mathcal{T}$).

In Example 5.2.1, a possible path is

$$((9, \blacktriangleright), d)((5, \blacktriangleleft), \text{length}(5))((5, \blacktriangleleft), \delta').$$

The following two paths are however disallowed:

$$\begin{aligned} &((9, \blacktriangleright), 0)((9, \blacktriangleright), d)((9, \blacktriangleleft), d)((9, \blacktriangleleft), 0). \\ &((7, \blacktriangleright), 0)((7, \blacktriangleright), \text{length}(7))((8, \blacktriangleright), 0). \end{aligned}$$

Projection on networks A *projection* on a spatial network is used to associate a point with a set of locations in the spatial network. We define the *orthogonal projection* of a point a on an edge e as the set

$$\begin{aligned} \text{proj}(a, e) = \{&(e, d) \in \textbf{Location}(\mathcal{G}) \mid \nexists\, d' \in [0, \text{length}(e)] \\ &\text{dist}(a, \texttt{geo}((e, d))) < \text{dist}(a, \texttt{geo}_0((e, d')))\}. \end{aligned}$$

To associate a with locations on a spatial network $\mathcal{G} = (\mathcal{N}, \mathcal{E}, \texttt{geo}, \texttt{from}, \texttt{to})$, we introduce the *orthogonal projection* of a on $\mathcal{G}$, defined as

$$\text{proj}(a, \mathcal{G}) = \bigcup_{e \in \mathcal{E}} \text{proj}(a, e).$$

Similarly, a *projection* of a point a on a transportation network $\mathcal{T} = (\mathcal{G}, M, U, V)$ is a set of locations in $\mathcal{T}$. The *orthogonal projection* of a on $\mathcal{T}$ is defined as:

$$\text{proj}(a, \mathcal{T}) = \{((e, \delta), d) \in \textbf{Location}(\mathcal{T}) \mid (e, d) \in \text{proj}(a, \mathcal{G})\}$$

Example 5.2.2 *Figure 5.3 shows the projection of a point a on the spatial network $\mathcal{G}_0$ defined in Figure 5.2. If we consider $\mathcal{T}_0$ as in Example 5.2.1, then some possible elements of* $\text{proj}(a, \mathcal{T}_0)$ *are*

$$((9, \blacktriangleright), d_9) \qquad ((9, \blacktriangleleft), d_9) \qquad ((7, \blacktriangleright), d_7) \qquad ((5, \blacktriangleleft), d_5)$$

On the other hand, $((5, \blacktriangleright), d_5) \notin \text{proj}(a, \mathcal{T}_0)$.

A spatial network built from OpenStreetMap data

In this part, we briefly describe the OpenStreetMap (OSM) model (Haklay and Weber 2008), then we explain how a spatial network is built from it. OSM data is composed of *nodes*, *ways*, and *relations*. Each such element can hold a set of textual key–value pairs, called *tags*.

A node represents a point. Each node is associated to a point in **Point** and can be used to represent road intersections, bus stops, building entrances, etc.

A way is a finite sequence of nodes. A way geographically represents either a trail or a polygon. As trails, they can be used to represent footpaths, roads, rail lines, etc. As polygons, they can represent building outlines, parks, etc.

A relation is a finite sequence of nodes, ways or other relations. A relation can be used to represent turn restrictions, administrative boundaries, routes such as bus routes or even major roads (trails).

Let *ways* that function as trails be called *trail-ways*, and let the order relation induced by a way w be denoted $\leq_w$. A spatial network $\mathcal{G} = (\mathcal{N}, \mathcal{E}, \texttt{geo}, \texttt{from}, \texttt{to})$ is built from OSM data by following these steps:

- $\mathcal{N}$ is defined as the set of OSM nodes that are either *extremities of a trail-way* or *are members of two or more trail-ways*. $\texttt{geo}(n)$ is defined as the point to which the node n is associated.
- $\mathcal{E}$ is defined as the set of $e = (n, n') \in \mathcal{N}^2$ such that:
  1. $n \neq n'$,
  2. both n and n' are members of a w that is not a polygon,
  3. $n \leq_w n'$,
  4. and there does not exist a $n'' \in \mathcal{N} \setminus \{n, n'\}$ such that $n \leq_w n'' \leq_w n'$.

  $\texttt{from}(e)$ (resp. $\texttt{to}(e)$) is set equal to n (resp. n'), and $\texttt{geo}(e)$ is defined as the substring of w from n to n'
- Similarly, an edge $e \in \mathcal{E}$ is added for each relation that may be interpreted as an OSM way.

Transportation networks built from OpenStreetMap data

We have previously shown how to build a spatial network $\mathcal{G}$ from OSM data. We now describe the construction of transportation networks $\mathcal{T}(m) = (\mathcal{G}_m, M_m, U_m, V_m)$ on top, for each transportation mode m among `foot`, `bicycle`,

`car`, `bus`, `tram`, `metro` and `train`. This construction uses the tags that are associated to edges in OSM data. These tags give information about the kind of vehicles that can access the different footpaths, roads, and railways represented in OSM: `highway=motorway`, `highway=cycleway`, `highway=footway`, `access=no`, `bus=yes`, etc.

OSM mode profiles To construct a transportation network from OSM data, we use an OSM mode profile. An OSM mode profile is a set of tag rules defining for a given transportation mode whether an edge is traversable. Several journey planners today use OSM data and base themselves on such profiles. Among the open-source ones, the most well-known (Karich and Schröder 2016; Byrd and Grégoire 2016) have already identified and published descriptions of OSM profiles for the modes `foot`, `bicycle`, and `car`. For each mode, there are even rules to extract information that is useful for associating costs to edges, such as the speed limit. These costs are used in journey planning to be able to compute the shortest path between two points. In our work, we based ourselves on their mode profile specifications. We simplified them so as to remove restrictive rules that were rare or overly specific and possibly highly inaccurate. These profiles are sufficient to build the sets M_m and U_t of possible edge traversals and transitions for the modes `foot`, `bicycle`, and `car`. In addition, we defined our own rules for `bus`, `metro`, `tram`, and `train` modes. Typically, the value of the `railway` tag on an edge is indicative of the kind of rail transportation that can traverse it. We did not impose any transition restrictions. For the set of in-edge transitions V_m, we disallowed U-turns for `car` and `bus` on roads such as highways.

5.3 Public transportation routes and schedules

In this section, we describe the representation of public transportation routes and schedules used by the system. This representation is based on the General Transit Feed Specification (GTFS) (Google 2015a). Many public transportation agencies all over the world now use the GTFS format to publish information about the routes and schedules they operate. GTFS defines the following concepts: *agencies*, *routes*, *trips*, *stops*, *stop-times*, *calendars*, *shapes*, and *transfers*.

Informally, an *agency* represents a transit operator that is in charge of several *routes*. (Routes are known as "lines" in some public transportation systems.) A *route* is a group of *trips* that are displayed to passengers as a single service. *Trips* are finite sequences of two or more *stops* traversed in order by a particular vehicle at a particular time of the day. A *stop* is geographically represented by a point and represents a location where a vehicle can pick up or drop off passengers. A *stop-time* represents the time when a particular *trip* arrives and departs from a *stop*. A `calendar` represents the precise dates an individual *trip* operates. Finally, a *transfer* is a pair of stops representing a foot connection between two routes. Optionally, a *trip* might be geographically referenced in the form of a *shape* (a trail).

A *trip pattern* is a set of *trips*, where two *trips* are said to belong to the

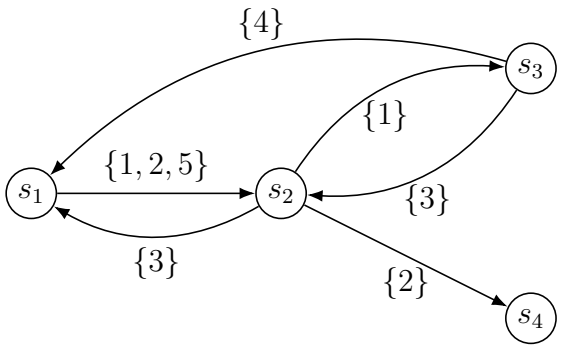


Figure 5.4: The graph representing the admissible moves of the trip pattern from Example 5.3.1. Each node in this graph represents a stop and each edge (s, s') represents the existence of a trip pattern p for which (s, s') is a pair of consecutive stops in p. Edges are labeled with the set of matching trip patterns.

same *trip pattern* if and only if they belong to the same *route* and have the same sequence of *stops*. The set of *trip patterns* for a given *route* forms a partition of the set *trips* of this *route*. A *route* models a real-world public transportation line. A *trip pattern* of this route corresponds to an particular order in which vehicles on this line can traverse its stations (e.g., an express service that only stops at selected stations). In GTFS, each *route* is associated with a transportation mode such as bus, tram, metro, or train. Example 5.3.1 gives an example of a route and its set of trip patterns.

Example 5.3.1 *Let $\{s_1, s_2, s_3, s_4\}$ be the set of stops of a bus route. The direction of travel is either eastwards or westwards. Five trip patterns compose this route. They are defined by the following sequences of stops:*

1. *(s_1, s_2, s_3), traveling eastwards;*
2. *(s_1, s_2, s_4), which is like 1. but with an alternative last stop;*
3. *(s_3, s_2, s_1), which is the reverse of 1., traveling westwards;*
4. *(s_3, s_1), which is like 3. but skips stop s_2;*
5. *(s_1, s_2), which is like 1. and 2. but stops earlier.*

Figure 5.4 shows a graph representing all admissible moves from one stop to another in any of these trip patterns.

Enriching *routes* using the transportation networks

An issue with GTFS data is that it often comes with missing or imprecise *shapes*, which are the geographical descriptions of the paths taken by public transportation vehicles. Moreover, the itinerary recognition task could benefit from knowledge of whether a stop or part of a trip is underground, which is never provided. To make route information more useful, we decided to enrich routes by exploiting the transportation networks built from OSM (cf. Section 5.2)

To do so, we associate each trip pattern with a path in the transportation network $\mathcal{T}(m)$ built from OSM using the procedure described in Section 5.2, where m is the trip pattern's mode. We developed an automatic alignment procedure to perform this association, that we briefly describe next.

The association that we want to perform can be formulated as a map-matching problem (Quddus, Ochieng, and Noland 2007), which consists in matching location observations to a path in a road network. In such setting, the sequence of stops $s_1, \ldots, s_T$ in a trip pattern p corresponds to the sequence of location observations to match. The position of the observation at index t is defined by the geographic point z_t of stop s_t. Observations cannot be bound to an absolute time. However, the duration between two consecutive observations can be defined as the average time a vehicle takes to go from one stop to the next (averaged over the set of trips). The road network in our case is the transportation network $\mathcal{T}$ built from OSM for the particular mode of p. The problem is to match the observations to a sequence of locations $(l_1, \ldots, l_T)$ and a path in $\mathcal{T}$ between these locations $l_1\pi_1 l_2\pi_2 \cdots l_{T-1}\pi_{T-1}l_T$ that corresponds to the path most likely followed by a vehicle in trip pattern p. The difficulty here is that consecutive observations are minutes apart.

To solve this problem, we compute for each stop s_t the set of its nearby orthogonal projections $\{l_{t,1}, \ldots, l_{t,n_t}\}$ on $\mathcal{T}$. Then, for each pair of orthogonal projections $(l_{t,i}, l_{t+1,j})$ obtained from consecutive stops, we compute the set of shortest paths from $l_{t,i}$ to $l_{t+1,j}$ in the transportation network $\mathcal{T}$. When this set is not empty, $\tilde{\pi}_{(t,i),(t+1,j)}$ denotes one of its elements. After that, we build a trellis[1] that has T slices in which the nodes of the t-th slice are the stop-projections $l_{t,1}, \ldots, l_{t,n_t}$ and in which there exists an edge between $l_{t,i}$ and $l_{t+1,j}$ if there exists a path between these locations in $\mathcal{T}$. We add two extra nodes s and e to the trellis, and connect them to rest of the nodes by adding an edge between s and each node in the first slice and an edge between e and each node in the T-th slice.

Weights are added to the trellis so that the weight of each acyclic path π from s to e is equal to the sum of shortest path lengths between consecutive stop-projections composing π and the sum of geographical distances between each stop-projection composing π and its associated stop. More precisely, the edge weight function w is defined as follows:

- for all i, $w(s, l_{1,i}) = \text{dist}(z_1, l_{1,i})$,
- for all t, i and j, $w(l_{t,i}, l_{t+1,j}) = \text{length}(\tilde{\pi}_{(t,i),(t+1,j)}) + \text{dist}(z_{t+1}, l_{t+1,j})$,
- and for all i, $w(l_{T,i}, e) = 0$.

Every acyclic path in the trellis from s to e yields an alternating sequence $l_1\pi_1 l_2 \cdots l_{T-1}\pi_{T-1}l_T$ of locations and paths in $\mathcal{T}$, which altogether defines a path in $\mathcal{T}$. One of these paths is among the shortest and is chosen as the path that is associated with the trip pattern p. An illustration of a trellis constructed from a trip pattern with five stops is given in Figure 5.5. In Section 5.7, we demonstrate

[1] A trellis is a graph whose nodes are partitioned into a set of time slices, ordered by time, and whose edges are pairs of nodes in adjacent time slices. The earliest and latest time slices in the trellis only have one node.

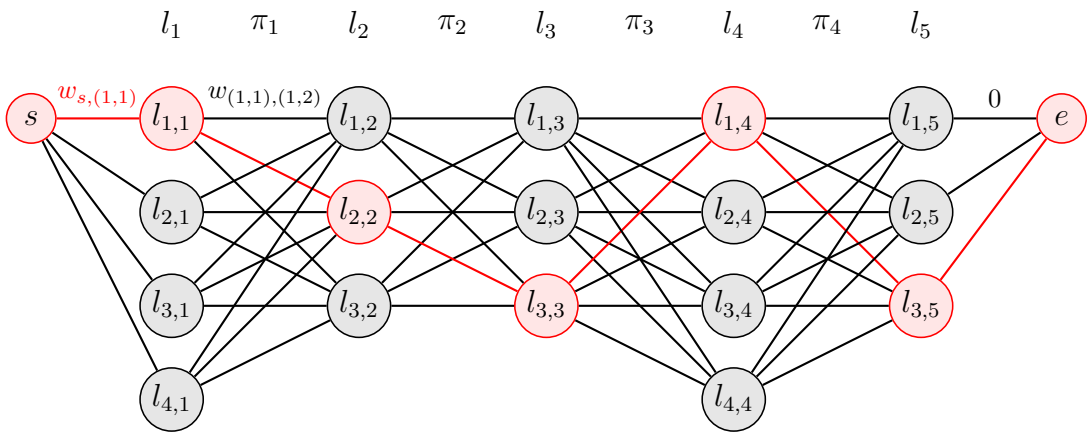


Figure 5.5: The trellis built during the matching of a trip pattern consisting of five stops to some path $l_1\pi_1 l_2\pi_2 l_3\pi_3 l_4\pi_4 l_5$ in $\mathcal{T}$ in the transportation network. In this example, the shortest path in the trellis is highlighted in red.

and briefly evaluate the result of this match. An alternative would have been to use the duration between each observation. Doing so would require being able to translate times into distances. Another possibility is using information of other kinds (found in OSM tags for instance), in which case we would perhaps need to use a probabilistic model. The disadvantage is that such model would require training.

5.4 Mobile sensor observations

In this section, we describe the observations made by the sensors of the user's mobile device over the course of her journey, and specify the preprocessing and extraction of features from these observations for use in itinerary recognition.

Raw sensor data is collected using the application described in Section 3.5 and is stored as a collection of time series. The application collects data over a given period of time $[\tau^s, \tau^e]$ from *locational*, *dynamic*, and *contextual* sensors. Sensors capture different kinds of knowledge. To perform itinerary recognition, sensor data is transformed into a set of informative values (features) relevant to the itinerary recognition task. Some of these features are computed using knowledge taken from the set transportation networks $\{\mathcal{T}(m)|m \in \mathcal{M}\}$ described in Section 5.2.

Raw sensor data do not form uniform time series. Most sensors, such as *locational* sensors, try to perform periodic measurements, but some measurements may be missed and measurements may not be completely regularly spaced in time. Other sensors, such as the sensor recording the signal strength of the cellular network, may only record state changes. To use these measurements for itinerary recognition, sensor data is converted into an evenly spaced time series of feature vectors $(\tau_t, v_t)_{1\leq t\leq T}$. The sampling frequency of this series is $1/\tau$ Hz, where τ is a strictly positive integer. We call this process *feature extraction*. The τ_t's are the

feature observation times in $[\tau^s, \tau^e]$. Each v_t is a real-valued multidimensional vector that is computed from sensor measurements taken before time τ_t. Each feature vector is defined as the concatenation of three vectors (its components), one for each kind of sensor: *locational*, *dynamic*, and *contextual*. Each component is the result of a specific process that is applied to a specific kind of sensor data. After defining the sequence of feature observation times, we present how each of these components is produced.

Observation times The definition of the sequence of feature observation times $\tau_{1:T}$ is based on *locational* sensor observation times. The idea is that location information, which is not always available, has the most impact on the size of the search space when trying to infer the user's itinerary. The location of the user's mobile device is requested every second from satellite and network location sensors. For each of these sensors, a measurement is obtained at most every second. The set of location measurements form a non-uniform time series spanning $[\rho^s, \rho^e] \subset [\tau^s, \tau^e]$. Let τ_1 be $\rho^s + \tau$, where τ is the desired sampling period of the feature vector time series. If $\rho^e \geq \tau_1$, then $T - 1$ is defined as the quotient in the integer division of $\rho^e - \tau_1$ by τ. Otherwise, T is defined as 0 and the resulting feature vector time series is empty. For $t \in \{1, \ldots, T\}$, τ_t is defined as $\tau_1 + \tau(t - 1)$.

Locational To obtain the *locational* component of the feature vector time series, the raw location measurements time series is resampled to 1 Hz by picking the sample with the highest accuracy (i.e., with the smallest radius) over a sliding 1-second window that starts at ρ^s. If no sample is available within the window, a placeholder value $\perp$ is picked. For each t, let L_t be the sequence of locations over the time interval $]\tau_t - \tau, \tau_t]$ in the resampled location series. Each L_t is of size τ. $(\tau_t, L_t)_{1 \leq t \leq T}$ is called the *location sequences* time series. The *locational* component of v_t is defined with respect to the sequence of locations L_t, and is the defined as the concatenation of the following vectors:

- Speed measurements included within L_t form a vector of size τ, assuming that missing speed measurements are replaced by a fixed large negative value.
- A vector of location accuracies and a Boolean vector indicating which locations in L_t are $\perp$ are also similarly formed. Missing location accuracies are replaced by a large positive value.
- Finally, for each transportation mode m in $\mathcal{M}$, a vector of size τ whose i-th coordinate is the minimum distance between the i-th location in L_t and its projection on the transportation network $\mathcal{T}(m)$ (when the i-th location is $\perp$, a fixed large positive value is used).

Dynamic *Dynamic* sensor information includes accelerometer measurements sampled at somewhere between 50 to 100 Hz. While we are aware that combining accelerometer data with gyroscope and magnetometer measurements could help

	Satellite	Cellular	Wi-Fi	Bluetooth
Emitters	Satellites	Towers	Access points	Discoverable devices
Level	SNR	RSSI	RSSI	RSSI
Sample rate	1 Hz	On-change	1/60 Hz	1/60 Hz

Table 5.1: Characteristics of radio-based technology contextual features. (SNR stands for Signal-to-Noise Ratio and RSSI stands for Received Signal Strength Indication.)

reduce the uncertainty and possibly even let us derive the orientation of the acceleration vector with respect to Earth, data from these two other sensors were not considered for this algorithm. As a consequence, only the norm of the acceleration vector is used, which forms an almost uniform time series. This series is interpolated and downsampled to 20 Hz and analyzed over overlapping sliding windows for which statistical features (e.g., mean, standard deviation, etc), time domain features (e.g., integral) and frequency domain features (discrete Fourier transform components) are computed. The *dynamic* component of v_t is the concatenation of these features computed over time windows inside $[\tau_t - \tau, \tau_t]$.

Contextual For each type of radio-based technology used by the mobile device, the measured signal levels of in-range emitters are considered. Statistics such as the emitter count and the mean, the maximum, and standard deviation of the signal level are used as features. Table 5.1 details the specifics. For each technology, the most recent measurement at time τ_t is used to define the features at time τ_t. When no measurement is available (e.g., at the beginning), statistical features are replaced with appropriate dummy values, and a Boolean feature is used to represent the unavailability of a measurement. For Bluetooth, additionally, a measure of the number of emitters that stay in range over two consecutive scans is used a feature. As discussed in Section 3.5, the number of discoverable Bluetooth peers in range may be indicative of the number of people around the user. In particular, if the user is traveling in a vehicle with other people, then some of these peers may stay in range multiple times over the course of the user's journey. This additional measure tries to capture this phenomenon.

5.5 Itinerary recognition

In this section, we define the *itinerary recognition* problem. We begin by presenting the *activity recognition* problem, which is more general, and *route recognition*, which is more specific.

Activity recognition Activity recognition is concerned with determining the actions and goals of one or several agents given a series of observations on their actions and the environment (L. Chen et al. 2011). In our setting, the agent of interest is Alice. Activity recognition is mostly concerned with determining

Alice's activity at any given time τ from observations before or immediately after τ, and particularly in an online fashion or in real-time. This is done by context-aware mobile applications that need to adapt quickly to a change in user activity.

In some cases, the problem is to build a timeline of Alice's activities, i.e., to determine Alice's activities during a given time interval $[\tau^s, \tau^e]$, after all observations over this interval have been collected. One difference in this case is the ability to take into account evidence of an activity that may come later on (e.g., distinguishing between equally likely "walking upstairs" and "walking downstairs" activities depending on whether Alice is later detected on the building's rooftop or in its underground parking lot). Formally, *activity sequence recognition* is the problem of determining a sequence of activities:

$$(\theta_0, a_1, \theta_1, a_2, \ldots, \theta_{i-1}, a_i, \theta_i \ldots a_n, \theta_n),$$

where the θ_i's are the transition times ($\tau^s = \theta_0 < \ldots < \theta_i < \theta_{i+1} < \ldots < \theta_n = \tau^e$), and each a_i is the user's activity in $\mathcal{A}$ over the time interval $[\theta_{i-1}, \theta_i]$.

Typically, the set of activities $\mathcal{A}$ to recognize from is finite, and, in *human activity recognition*, usually small (there is rarely more than ten classes) (Lara and Labrador 2013). Example activities are *tooth brushing*, *walking*, *running*, *sitting on a desk*, etc. Another special case of activity recognition is *transportation mode recognition*, in which activities to recognize are modes of transportation such as `foot`, `car`, `bicycle`, `bus`, etc.

Route recognition *Route recognition* or *route matching* is the problem of recognizing, given a set of observations over a time interval of unimodal transportation, whether Alice was on board a vehicle of public transportation, and identify the route it belongs to: an identifiable regular flight, a particular bus line, a train line. One difficulty is differentiating routes that only differ by subtle aspects (e.g., geographically overlapping routes with slightly different schedules). Another difficulty is recognizing a route over a short period of time, where the probability of overlapping with another route is higher, or, in the case of bus routes, where there may not be enough evidence to distinguish it from a non-public mode such as a car or taxi.

Itinerary recognition *Itinerary recognition* is a type of activity sequence recognition. The problem is to determine Alice's *itinerary*, i.e., the sequence of transportation modes and routes she used, given observations over a time interval $[\tau^s, \tau^e]$. An *itinerary* is defined as a sequence

$$I = (\theta_0)(m_1, r_1)(\theta_1, s_1)(m_2, r_2) \cdots (\theta_{i-1}, s_{i-1})(m_i, r_i)(\theta_i, s_i) \cdots (m_n, r_n)(\theta_n),$$

where the θ_i's are the transition times ($\tau^s = \theta_0 < \ldots < \theta_i < \theta_{i+1} < \ldots < \theta_n = \tau^e$), and each s_i is the stop (e.g., Gare du Nord, Heathrow Airport, Berlin Hauptbahnhof) at which the transition at time θ_i happens. Each m_i and r_i are respectively the user's transportation mode and route over the time interval $[\theta_{i-1}, \theta_i]$.

Itinerary recognition subsumes *transportation mode recognition* and *route recognition*. It is also a considerably more difficult problem. For instance, assuming that perfect solutions to these two other problems are available, itinerary recognition can in theory be solved by performing transportation mode recognition followed by route recognition over each recognized segment of unimodal vehicular transportation. In practice, however, transportation mode recognition cannot always be perfectly solved, so real solutions need to take into account that the assumption of unimodal transportation on the input of route recognition may not always be realized. Additionally, due to the difficulty of recognizing routes over short periods of time, an erroneously fragmented route may be harder to recognize. Thus, a combined solution may have to consider different segmentation hypotheses, run route recognition on each of them, and select the itinerary that best fits the whole observation sequence. Finally, if we simply formulated itinerary recognition as an activity sequence recognition problem, the set of activities to recognize would be much larger than in typical human activity or transportation mode recognition, as this set would have to contain the set of routes with their respective stops.

5.6 *Movup*'s itinerary recognition algorithm

In this section, we present the supervised learning algorithm for itinerary recognition that Movup uses. The algorithm takes as input:

- a set of observations $\mathcal{O}$ made by the sensors of the user's mobile device over the course of the time interval $[\tau^s, \tau^e]$;
- a set of vehicular modes $\mathcal{M}_V$, e.g., $\{\texttt{bicycle}, \texttt{car}, \texttt{bus}, \texttt{train}, \texttt{tram}, \texttt{metro}\}$;
- a set of public transportation modes $\mathcal{M}_P$ that is a subset of $\mathcal{M}_V$, e.g., $\{\texttt{bus}, \texttt{train}, \texttt{tram}, \texttt{metro}\}$;
- a set of transportation modes $\mathcal{M} = \{\texttt{foot}\} \cup \mathcal{M}_V$ that includes in addition to $\mathcal{M}_V$ a distinguishable foot mode `foot`;
- a description of public transportation routes and their schedules $\mathcal{P}$ (Each route r has a mode $m(r) \in \mathcal{M}_P$ and defines a set of trip patterns. The set of routes and stops are respectively denoted $\mathcal{R}$ and $\mathcal{S}$.);
- a mapping $\mathcal{T}$ assigning to each transportation mode $m \in \mathcal{M}$ a transportation network $\mathcal{T}(m)$.

The itinerary model

A pair of sequences

$$I = ((m_i, r_i)_{1 \leq i \leq n}, (\theta_i, s_i)_{1 \leq i \leq n-1}) \in (\mathcal{M} \times (\mathcal{R} \cup \{\perp\}))^n \times ([\tau^s, \tau^e] \times \mathcal{S})^n$$

is said to be an *itinerary* of length n if and only if:

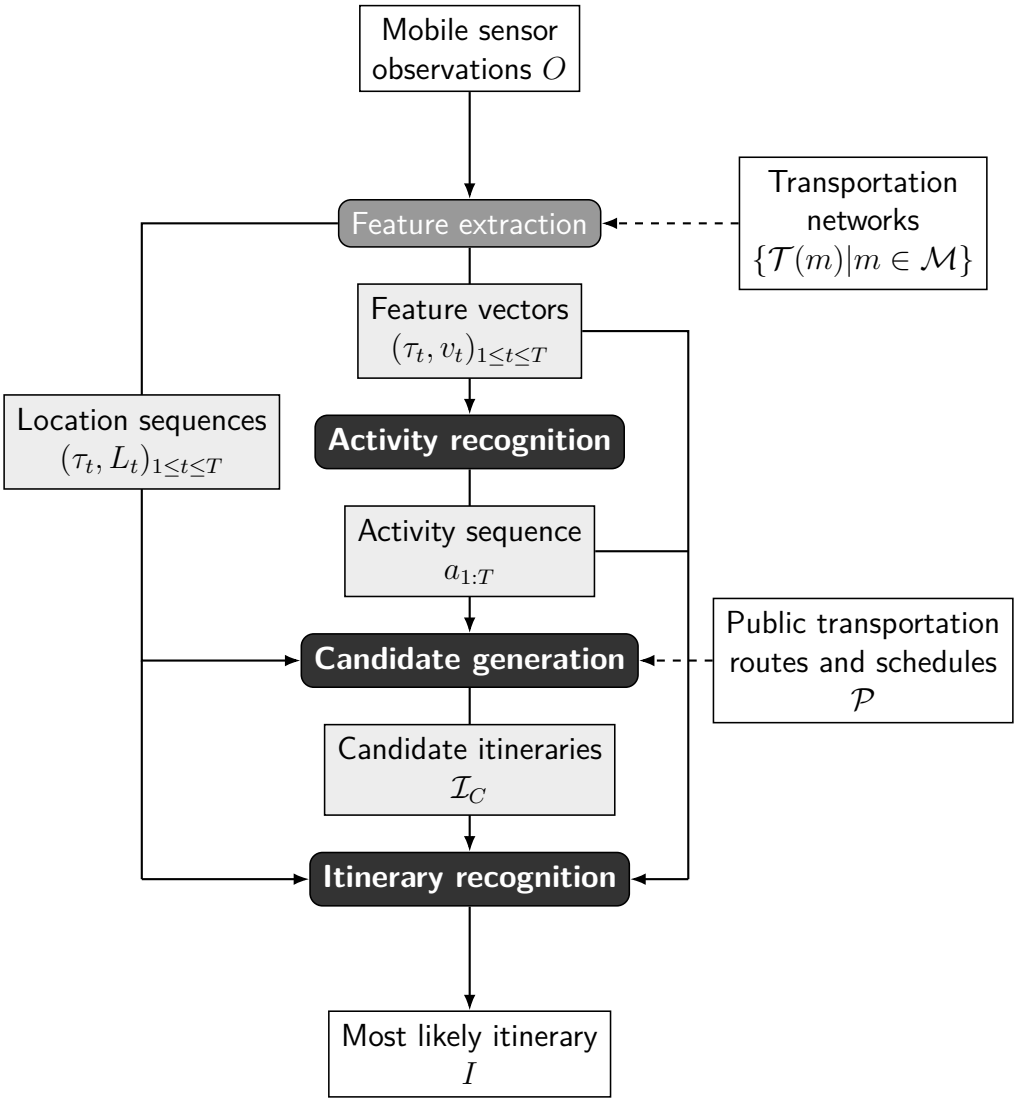


Figure 5.6: Overview of Movup's algorithm.

- for all i, $\theta_i < \theta_{i+1}$;
- for all i, either $r_i = \perp$ or $r_i \in \mathcal{R}$, $m_i \in \mathcal{M}_P$, and $m(r_i) = m_i$;
- for all $i > 1$, if $r_i \in \mathcal{R}$ then s_{i-1} and s_i appear in this order in a trip pattern of r;
- if $r_1 \neq \perp$ then s_1 belongs to a trip pattern of r_1.

We denote by $\mathcal{I}$ the set of itineraries.

Algorithm

The task of Movup's algorithm is to find the most likely itinerary $I \in \mathcal{I}$ with respect to the set of observations O. The algorithm is divided into stages that progressively derive more advanced knowledge: (1) *feature extraction*, (2) *activity*

recognition, (3) *candidate generation*, and (4) *itinerary recognition*. Figure 5.6 gives an overall picture of this decomposition. Next, we outline the inputs and outputs of each stage. After that, we provide a detailed description of each of them.

1. The first stage, denoted *feature extraction*, produces a time series of *feature vectors* $(\tau_t, v_t)_{1 \leq t \leq T}$ and *location sequences* $(\tau_t, L_T)_{1 \leq t \leq T}$ from the set of observations O and knowledge taken from the set of transportation networks $\{\mathcal{T}_m | m \in \mathcal{M}\}$.

2. The second stage, denoted *activity recognition*, infers the most likely activity sequence $a_{1:T}$ among `Stationary`, `MoveFoot`, and `MoveVehicle` using classic activity sequence recognition techniques. It takes as input the time series of feature vectors $(\tau_t, v_t)_{1 \leq t \leq T}$ output by the first stage.

3. The third stage, denoted *candidate generation*, produces a set of candidate itineraries $\mathcal{I}_C \subset \mathcal{I}$ that is encoded as a set of paths in a graph in which each node corresponds to a particular *itinerary state* over the course of the user's journey. This stage takes as input the activity sequence $a_{1:T}$ output by the second stage and the *location sequence* $(\tau_t, L_T)_{1 \leq t \leq T}$ output by the first stage.

4. The fourth and final stage, denoted *itinerary recognition*, uses a probabilistic model on top of the set of candidate itineraries $\mathcal{I}_C$ and looks for the itinerary I of maximum probability. The feature vector time series $(\tau_t, v_t)_{1 \leq t \leq T}$ and the recognized activity sequence $a_{1:T}$ is reused by this stage in the definition of the probability of each itinerary state with respect to the observations.

Activity recognition The *activity recognition* stage processes the sequence of feature vectors $v_{1:T}$ using a feedforward neural network (Goodfellow, Bengio, and Courville 2016a) that consists of multiple fully-connected layers, i.e., where each unit (neuron) in each layer is connected to every unit in the previous layer. The number of units comprising the input layer of this network is equal to the number of dimensions of each feature vector v_t. The output layer of the network forms a three-dimensional vector $A(v; \theta_N)$ representing the *energies* of the activities `Stationary`, `MoveFoot`, and `MoveVehicle`, given a feature vector v in the input layer and a set of weights w. The network is applied to each v_t with the same set of weights θ_N. The result is a sequence of vectors $(A(v_t; \theta_N))_{1 \leq t \leq T}$. To obtain the sequence of predictions $a_{1:T}$, the $A(v_t; \theta_N)$'s are used in the definition of the joint probability distribution of a linear-chain conditional random field (CRF) (Koller and Friedman 2009b) that models the conditional dependency of the activity sequence on the observations, which are represented by the sequence of feature vectors $v_{1:T}$. Such CRF allows each prediction a_t to take into account its context by globally normalizing the energy sequence $(A(v_t; \theta_N))_{1 \leq t \leq T}$ (Andor et al. 2016): the likelihood of the previous and next activities are taken into account in the prediction of the current activity. For each t, the CRF defines two random variables: an *observed variable* X_t, which represents the feature vector v_t at time

τ_t, and a *target variable* Y_t, which represents the activity a_t. The domain of Y_t is $\{\texttt{Stationary}, \texttt{MoveFoot}, \texttt{MoveVehicle}\}$. The conditional probability distribution of the CRF's target variables $Y_{1:T}$ given the observed variables X_t is a log-linear model with weights θ_L over its feature functions. Given the neural network weights θ_N, the conditional probability distribution is parametrized by $\theta = (\theta_N, \theta_L)$ and is given by:

$$P(Y_{1:T}|X_{1:T};\theta) = \frac{1}{Z_\theta(X_{1:T})}\tilde{P}_\theta(Y_{1:T}, X_{1:T})$$

where the joint probability distribution and the normalization function are equal to:

$$\tilde{P}_\theta(Y_{1:T}, X_{1:T}) = \prod_{t=2}^{T} \phi_{\theta_L}(Y_{t-1}, Y_t) \prod_{t=1}^{T} \psi_{\theta_N}(Y_t, X_t)$$
$$Z_\theta(X_{1:T}) = \sum_{Y_{1:T}} \tilde{P}_\theta(Y_{1:T}, X_{1:T}).$$

The $\psi_{\theta_N}(Y_t, X_t)$'s and $\phi_{\theta_L}(Y_{t-1}, Y_t)$'s are respectively called the observation and transition factors. For each t and each activity i, the observation factor is the exponential function of the activity's energy, $A(v_t;\theta_N)(i)$:

$$\psi_{\theta_N}(Y_t = i, X_t = v_t) = e^{-A(v_t;\theta_N)(i)}.$$

θ_L consist of 3^2 real parameters defining a square matrix representing the transition energies $\theta_L(i,j)$ between each pair (i,j) of activities. The transition factor at t between an activity i and an activity j is defined as:

$$\phi_{\theta_L}(Y_{t-1} = i, Y_t = j) = e^{-\theta_L(i,j)}.$$

When T is equal to 1 (i.e., the sequence is reduced to a single point), the model is equivalent to a feedforward neural network with a softmax activation function on its output layer.

The model is trained using a dataset of annotated journeys. For each journey, the annotations segment the observation time interval $[\tau^s, \tau^e]$ into time periods of either `Stationary`, `MoveFoot`, or `MoveVehicle` states. These annotations are the ground truth of the activity recognition model. To train the model, the feature vectors $v_{1:T}$ of each journey are plugged into the input layer of the neural network and the annotated $\{\texttt{Stationary}, \texttt{MoveFoot}, \texttt{MoveVehicle}\}$ labels $y_{1:T}$ into the target variables $Y_{1:T}$ of the linear-chain CRF. Training is performed by maximizing the likelihood

$$P(Y_{1:T} = y_{1:T}|X_{1:T} = v_{1:T};\theta)$$

over the set of parameters θ using gradient descent. Since the CRF is a linear-chain, the likelihood $P(Y_t|X_t;\theta)$ can be computed efficiently using the forward-backward algorithm. When this likelihood is log-linear over parameter-free feature functions, the optimization is convex. In our case, however, observation factors are exponential functions of the outputs of a multi-layer feedforward neural network, so the optimization is not necessarily convex. Once the model has been trained, the most likely activity sequence (the most probable assignment) can be computed using the Viterbi algorithm for CRFs (Sutton, McCallum, et al. 2012).

Candidate generation The input to the *candidate generation* stage is the activity sequence $a_{1:T}$ inferred by the previous stage, the location sequences time series $(\tau_t, L_T)_{1 \leq t \leq T}$, and the description of public transportation routes and their schedules $\mathcal{P}$, This stage produces a trellis that has T slices in which the nodes of slice t are the possible *itinerary states* S_t in which Alice may be at time τ_t (e.g., traveling from stop A to stop B via bus route R), and in which the edges represent possible itinerary state transitions (e.g., getting off the bus, getting into a car, or getting on a bicycle). The set of *candidate itineraries* $\mathcal{I}_C \subset \mathcal{I}$ output by this stage is defined as the set of paths from a node in the first slice to a node in the last slice.

The nodes in each slice and the edges between consecutive slices are generated sequentially by processing the input activity sequence and the location sequence time series in order. Initially, a set of itinerary states S_1 is constructed using a_1 and L_1, then, for each $t < T$, S_{t+1} is recursively defined as $S_{t+1} = F(S_t, \tau_{t+1}, a_{t+1}, L_{t+1}) = \bigcup_{s \in S_t} f(s, \tau_{t+1}, a_{t+1}, L_{t+1})$, where f is a function that defines the set of possible states to which a state s can transition to, depending on the predicted activity and the location of the next slice. The set of edges E_t between slices $t-1$ and t is the set of pairs $\{(s, \tilde{s}) | s \in S_{t-1}, s \in f(s)\}$.

We now define the itinerary state space, the initial set of states S_1, and the state transition function f. The components of an itinerary state are:

- the user's location estimate, defined by a point and a radius;
- the transportation mode among $\mathcal{M}$;
- the boarding public transportation stop and the trip pattern taken by the user, which are defined if and only if the transportation mode is public (i.e., in $\mathcal{M}_P$);
- the state's start time, which is equal to the boarding time when the transportation mode is public;
- the alighting stop, which can only be defined if the transportation mode is `foot` and which is used to indicate that to reach the current state the user previously alighted from a public transportation vehicle at this stop;
- the *erroneous activity counter*, which is a positive integer.

The initial set of states S_1 is defined as follows:

- The most recent location in L_1 is the user's location estimate for any state in S_1.
- A state is created for each non-public transportation mode (i.e., in $\mathcal{M} \setminus \mathcal{M}_P$). The state's start time is set to τ_1.
- Public transportation trip patterns that are close to the user's location estimate are used to generate states whose transportation mode is public. The boarding stop of each such state is any of the stops in the trip pattern that are close. The boarding time (the start time) of each such state is estimated from τ_1 and the distance to the boarding stop.

Let us now specify f by defining a set of sufficient and necessary conditions that relate s to any $\tilde{s}$ in $f(s, \tau_{t+1}, a_{t+1}, L_{t+1})$. Let s be a state at some $t \in \{1, \ldots, T-1\}$ (i.e., $s \in S_t$). An itinerary state $\tilde{s}$ belongs to $f(s, \tau_{t+1}, a_{t+1}, L_{t+1})$ if and only if the following statements hold:

- If a non-$\perp$ location exists in L_{t+1}, $\tilde{s}$'s location estimate is equal to the most recent location in L_{t+1}. Otherwise $\tilde{s}$'s location estimate is equal to s's location estimate if a_{t+1} is `Stationary`, or equal to s's location estimate with the uncertainty raised by the maximum distance attainable over the sampling period τ using s's transportation mode (i.e., using the fastest vehicle for this mode).
- $\tilde{s}$'s alighting stop is defined if and only if s's transportation mode is public.
- If s and $\tilde{s}$ have different transportation modes, then a_{t+1} is not `Stationary` and the following statements hold:
  - $\tilde{s}$'s start time is τ_t;
  - $\tilde{s}$'s erroneous activity counter is equal to 0;
  - if s's transportation mode is not `foot`, then a_{t+1} is `MoveFoot`, $\tilde{s}$'s transportation mode is `foot`
    - if s's transportation mode is public then $\tilde{s}$'s alighting stop is defined, follows s's boarding stop in the stop sequence defined by s's trip pattern, is within range of $\tilde{s}$'s location estimate, and within a confidence estimated trip time difference (i.e., τ_t minus s's boarding time is within a bounded distance to the scheduled trip time between $\tilde{s}$'s alighting stop and s boarding stop for s's trip pattern);
  - if s's transportation mode is `foot`, then a_{t+1} is `MoveVehicle` and $\tilde{s}$'s transportation mode is vehicular (i.e., in $\mathcal{M}_V$);
    - if $\tilde{s}$'s transportation mode is public then its departing public transportation stop is within range of the location estimate of $\tilde{s}$ and its trip pattern departs from this stop.
- If s's and $\tilde{s}$'s have the same transportation mode then they also have the same boarding stop, trip pattern, start time, and alighting stop. Additionally, exactly one of the following statements hold:
  - the erroneous activity counter of $\tilde{s}$ is equal to 0 and either
    - a_{t+1} is `Stationary`,
    - a_{t+1} is `MoveFoot` and $\tilde{s}$'s transportation mode is `foot`,
    - or a_{t+1} is `MoveVehicle` and $\tilde{s}$'s transportation mode is not `foot`;
  - or s's erroneous activity counter is less than a threshold ϵ, $\tilde{s}$'s erroneous activity counter is equal to s's plus one, and either
    - a_{t+1} is `MoveFoot` and $\tilde{s}$'s transportation mode is not equal to `foot`

* or a_{t+1} is `MoveVehicle` and $\tilde{s}$'s transportation mode is equal to `foot`.

The above rules follow multiple intuitions. First, while slices where the `MoveFoot` (resp. `MoveVehicle`) activity was detected by the previous stage should in principle correspond to `foot` (resp. vehicular) itinerary states, the rules allow vehicular (resp. `foot`) itinerary states to be used instead, up to a certain number of consecutive times. This behavior allows the itinerary generation stage to not rely on the activity recognition stage being perfectly accurate. Under some circumstances, some of these inaccuracies may actually be hard to prevent, for instance, the user may walk a considerable distance within a long train that is moving. To prevent possibly erroneous states from extending over a long period of time, the erroneous activity counter is used to keep track of the number of consecutive times there is mismatch between the detected activity and the itinerary state's transportation mode. Another intuition is the idea that the user's transportation mode cannot change without significant movement. As such, transportation mode changes between t and $t+1$ are only allowed when a_{t+1} is either `MoveFoot` or `MoveVehicle`. Finally, the rules take into account the assumption that a transition between two segments of vehicular mobility requires some period of pedestrian activity in-between (Zheng et al. 2010; Hemminki, Nurmi, and Tarkoma 2013). To do so, the only allowed transitions are from a vehicular transportation mode to a `foot` mode and vice versa.

Itinerary recognition Itinerary recognition, the last stage, takes as input the set of candidate itineraries $\mathcal{I}_C$ output by the candidate generation stage, the sequence of feature vectors $v_{1:T}$, and the activity sequence $a_{1:T}$. The set of candidate itineraries $\mathcal{I}_C$ is encoded as the set of paths in a trellis with T slices from a node in the first slice to a node in the last slice. The goal of this stage is to find the path in this trellis representing the *most likely itinerary* with respect to the set of observations O. To do so, this stage uses a linear-chain conditional random field that models the conditional probability of a sequence of T itinerary states given the observations. As in the activity recognition stage, the linear-chain CRF uses T target variables $Y_1, \ldots, Y_T$ and T observed variables $X_1, \ldots, X_T$. Each target variable Y_t represents the user's itinerary state at τ_t, and thus the domain of Y_t is S_t. Each observed variable X_t represents a tuple composed by the time τ_t, the feature vector v_t, the activity a_t, and the location L_t. Again, both observation and transition factors are used to model the joint probability distribution $P(Y_{1:T}, X_{1:T}; \rho)$, where $\rho = (\rho_N, \rho_L)$ are the parameters. This time however, the transition factors are slightly different as they take into account the full sequence of observations at each t: $\phi_{\rho_L}(Y_{t-1}, Y_t, X_{1:T})$. The observation factor at t is written $\psi_{\rho_N}(Y_t, X_t)$. In addition to observation and transition factors, the itinerary recognition CRF uses for each t a bias factor $\chi(Y_{t-1} = s, Y_t = \tilde{s}, E_t)$ that is equal to 1 when $(s, \tilde{s})$ is in E_t and is equal to 0 otherwise:

$$\tilde{P}_\rho(Y_{1:T}, X_{1:T}) = \prod_{t=2}^{T} \chi(Y_{t-1}, Y_t, E_t)\phi_{\rho_L}(Y_{t-1}, Y_t, X_{1:T}) \prod_{t=1}^{T} \psi_{\rho_N}(Y_t, X_t).$$

The bias factors force the joint probability distribution $P(Y_{1:T}, X_{1:T})$ to be zero when the valuation of the sequence of target variables $Y_{1:T}$ is not a path in $\mathcal{I}_C$'s trellis, i.e., if it does not represent a candidate itinerary. For an itinerary state s and an activity a, let $V(s,a) \in \mathcal{M} \times \{0,1\} = \mathcal{V}$ be the pair formed by the state's transportation mode and a bit that is equal to 1 if and only if a is equal to `Stationary`. Similarly to the activity recognition CRF, the observation factor $\psi_{\rho_N}(Y_t, X_t)$ is defined using the output of a feedforward neural network consisting of multiple fully-connected layers. The neural network is applied to each v_t, yielding an output vector $B(v_t; \rho_N)$, where ρ_N are the parameters of the neural network. The output vector $B(v_t; \rho_N)$ is a $|\mathcal{M} \times \{0,1\}|$-dimensional real vector defining the energies of each of the elements in $\mathcal{V}$:

$$\psi_{\rho_N}(Y_t = s, X_t = (\tau_t, v_t, a_t, L_t)) = e^{-B(v_t;\rho_N)(V(s,a_t))}.$$

The log of each transition factor $\log \phi_{\rho_L}(Y_{t-1} = s, Y_t = \tilde{s}, X_{1:T} = (\tau_{t'}, v_{t'}, a_{t'}, L_{t'})_{1 \leq t' \leq T})$ is equal to:

$$-\rho_L^g \left(g(s, a_t, \tilde{s}, a_{t+1})\right) + \rho_L^h \cdot H(s, \tilde{s}, (\tau_{t'}, a_{t'}, L_{t'})_{1 \leq t' \leq T}).$$

The first term in the above sum, $-\rho_L^g(g(s, a_t, \tilde{s}, a_{t+1}))$, considers the transition energies in the projected space $\mathcal{V}$ obtained via the mapping $g : (s, a_t, \tilde{s}, a_{t+1}) \mapsto (V(s, a_t), V(\tilde{s}, a_{t+1}))$. Given that some itinerary state transitions are impossible under the rules implemented by the itinerary generation stage, the image over the set of adjacent itinerary states at each t by this mapping is given by

$$\begin{aligned} G = \quad & (\mathcal{M}_V \times \{0,1\}) \times \{(\texttt{foot}, 1)\} \cup (\{\texttt{foot}\} \times \{0,1\}) \times \\ & (\mathcal{M}_V \times \{1\}) \cup \{((m, \sigma), (\tilde{m}, \tilde{\sigma})) \in \mathcal{V}^2 | m = \tilde{m}\} \subset \mathcal{V}^2, \end{aligned}$$

whose cardinal is $|\{0,1\}^2||\mathcal{M}|+2|\{0,1\}||\mathcal{M}_V|$. The energies associated to each resulting transition are given by the $|G|$-dimensional parameter vector $\rho_L^g : G \to \mathbb{R}^{|G|}$. The second term, $\rho_L^h \cdot H(s, \tilde{s}, (\tau_{t'}, a_{t'}, L_{t'})_{1 \leq t' \leq T})$, considers *segment-wise* features that measure the suitability of a candidate segment of unimodal public transportation. Since such measure can only be performed given information about a complete public transportation segment (i.e., from boarding until alighting), this term only materializes when $(s, \tilde{s})$ corresponds to a transition out of a public transportation mode (i.e., when $\tilde{s}$ has an alighting stop). This term is written as the dot product between the 7-dimensional feature vector function H applied to $(s, \tilde{s}, (\tau_{t'}, a_{t'}, L_{t'})_{1 \leq t' \leq T})$ and the parameter vector ρ_L^h. The seven features include information about the boarding time, boarding stop, and trip pattern defined by s, the alighting time and stop defined by $\tilde{s}$, and the activity and location subsequences $(\tau_{t'}, a_{t'}, L_{t'})_{t_s \leq t' \leq t}$ that coincide in time with the segment's time span (t_s is the boarding time). The following segment-wise features are defined:

- Three features measure how well the shape of the segment's location sequence fits the shape of the trip pattern:
  - the average geographical distance between the segment's location sequence and each of the non-underground passages and stops of the trip pattern,

  - the fraction of underground stops (e.g., underground metro stops) within the route that match missing satellite positioning information,
  - and the fraction of `Stationary` subsegments within the segment's activity sequence that match a stop of the trip pattern.

- Four features measure how well the segment's boarding and alighting times fit the schedule:
  - the time difference between the boarding time and the scheduled departure time of the trip departing immediately before (considering the possibility that the previous trip was late),
  - the time difference between the boarding time and the the trip immediately after (considering the possibility that the next trip departed early),
  - the relative time difference between the segment's duration (i.e., the difference between the alighting time and the boarding time) and the expected duration of the trip departing immediately before,
  - and the relative time difference between the segment's duration and the expected duration of the trip departing immediately after.

The itinerary recognition model is trained in a similar way to the activity recognition model used by the algorithm's second stage. For the itinerary recognition model, however, the sequences used as ground truth are harder to build, since these are defined over the itinerary states that the candidate generation stage generates. As such, these sequences can only be properly defined after a set of candidate itineraries is generated from an activity sequence for the observed journey. To build these sequences, we rely on a set of annotations that consist for each journey of a segmentation of the time interval $[\tau^s, \tau^e]$ where each segment is defined by a transportation mode, and where applicable, a public transportation route, a boarding stop, and an alighting stop. To train the model, the input to the candidate generation stage that is used can be either the ground truth activity sequence or the activity sequence predicted by the activity recognition stage, which may contain errors. For the latter, the activity recognition model needs to be trained beforehand. For each journey, after the set of candidate itineraries $\mathcal{I}_C$ is generated, the ground truth itinerary is obtained by computing the most likely sequence $y_{1:T}$ in an auxiliary linear-chain CRF that uses the same target variables and bias factors as the activity recognition model, no transition factors, and an observation factor that assigns a relatively high observation energy to itinerary states with the correct transportation mode, route, and stops. Finally, the itinerary recognition model is trained similarly to the activity recognition model by maximizing the likelihood:

$$P(Y_{1:T} = y_{1:T} | X_{1:T} = (v_t, a_t)_{1 \leq t \leq T}; \rho)$$

over the parameters ρ using gradient descent. Again, once the model has been trained, the most likely itinerary is computed using the Viterbi algorithm for CRFs.

System implementation

The construction of the transportation networks from OSM data, the integration of GTFS data, the enrichment of public transportation routes using the transportation network models, and the integration of sensor data were implemented using the Scala programming language (EPFL 2017). The system imports sensor data collected from the *Hup-me* mobile application presented in Section 3.5 and stored in CSV format into a relational database (PostgreSQL (PostgreSQL Global Development Group 2017)). The relational database also holds a representation of the transportation networks and a representation of public transportation routes and their schedules.

In order to facilitate annotations, we developed a web interface that displays for a recorded journey the location sequence on a map and sensor data in multiple time series plots. The interface allows the annotator to divide the recorded journey into segments of time with an associated transportation mode. When the annotated transportation mode is public, the annotator also identifies route as well as the boarding and alighting stops. In order to provide annotations for the activity recognition stage, i.e., identify the moments where the vehicle or the user was actually moving, the interface allows the annotator to further divide each segment of unimodal transportation into `Stationary`, `MoveFoot`, and `MoveVehicle` segments. Figure 5.7, Figure 5.8 and Figure 5.9 illustrate different components of the annotation interface.

The candidate generation algorithm was implemented in C#. The neural network and conditional random field models were written in Python using TensorFlow, an open source software library for numerical computation and machine learning using data flow graphs (Abadi et al. 2016).

5.7 Evaluation

In this section, we present an evaluation of our system. For this evaluation, we constructed a set of transportation networks for the Paris region from OSM data. It encompasses all roads and railways in this region. The following gives some intuition on the size of these networks: the spatial network consists of 808,093 nodes, 1,045,052 road edges, and 34,414 rail edges.

Additionally, we gathered GTFS data from local agencies and partitioned the set of trips into trip patterns. 3,412 trip patterns were found. Using the procedure described in Section 5.2, we matched metro, train, and tram trip patterns to rail paths in the corresponding transportation networks. Two examples of matched trip patterns are shown in Figure 5.11. To give an idea of the performance of this matching, we computed for each matched path the ratio of the path's length to the sum of the geographical distances between consecutive stops. Figure 5.12 shows the distribution of these ratios. The average ratio is 1.053, and a good amount of them are roughly normally distributed around 1.04 and 1.05. This is consistent with the fact that trains closely follow the geodesics between consecutive stops. The ratios above 1.10 correspond to trip patterns with long inter-stop distances (e.g., because they skip stops or because they cover long distances). An example

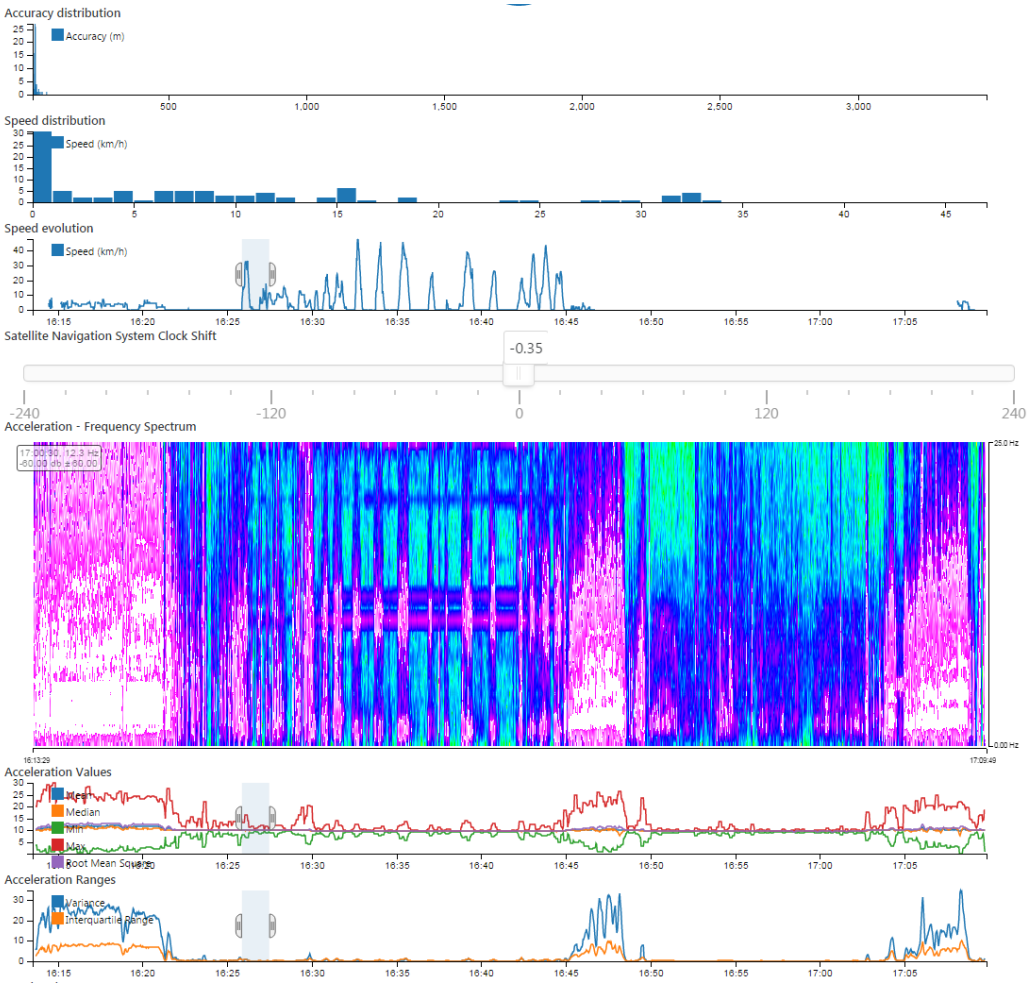


Figure 5.7: A component of Movup's annotation interface that shows the user's speed over time as measured by location sensors and features extracted from accelerometer data. The annotator has selected a time interval roughly between 4:26 and 4:27 p.m. (16:26 and 16:27, as displayed) using the brush selector on one of the time series. This interval is used to filter the points considered by the speed and location accuracy distribution charts, but also the locations that are shown on the map in Figure 5.8.

of a trip pattern that skips stops is given in Figure 5.13. This procedure seems to work remarkably well for rail modes, given that it requires no training. As mentioned before, given a ground truth, more advanced probabilistic models requiring training could be tried as well. The information provided by this alignment is useful for the recognition of itineraries using rail trip patterns, in particular `metro` ones, as these patterns use a lot of underground stops and passages.

To train and evaluate the different features of the itinerary recognition algorithm, we use manual annotations. We collected about 850 journeys (about 750 hours) from users traveling within the Paris region. Among those, we annotated 87 journeys (47 hours) with the help of the web interface we developed for visualizing journeys.

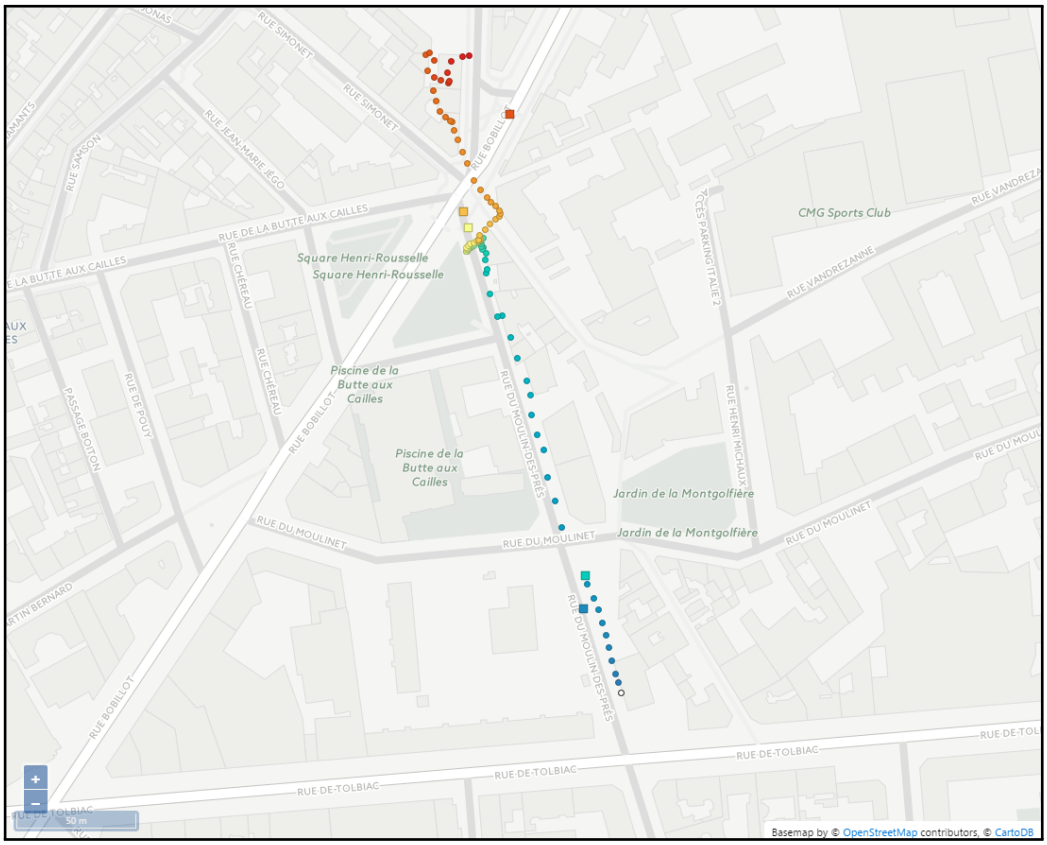


Figure 5.8: A component of Movup's annotation interface that shows the locations on a map corresponding to the time interval selected in the speed over time chart in Figure 5.7. Circle and square points represent locations given by satellite and network positioning sensors, respectively. The color rainbow gradient from blue to red represents the time progression.

At the moment, we are still working on an evaluation of *Movup*. Next, we show the results of our first evaluation that was conducted on the *Hup-me* system. *Hup-me* was evaluated in parallel on a 16-core virtual CPU with 104 GB memory implemented by a 2.3 GHz Intel Xeon E5 v3 CPU. We measured the execution time of our algorithm with respect to the duration of a journey. The curve of time as a function of the journey's duration is roughly linear (Figure 5.14). On average, *Hup-me* takes about 0.1638 seconds for each second in the observed journey, the median being 0.108. In terms of memory consumption, *Hup-me* can take up to 10 GB for some journeys.

Hup-me was evaluated on our set of 87 journeys. We measured the fraction of time where a mode i was predicted as j out of all cases where the actual state of the world is i. The results are presented in a confusion matrix (Table 5.2). To present these results, we purposely merged the `train` and `metro` modes since the dataset was not big enough. Additionally, when the algorithm guessed the correct mode among `bus`, `train`, `tram`, we evaluated whether the correct transportation route was detected (Table 5.3). Overall, the results are comparable to the state of the art (Stenneth et al. 2011; Hemminki, Nurmi, and Tarkoma 2013), considering

Figure 5.9: A component of Movup's annotation interface that shows the output of Google's activity recognition and features extracted from Wi-Fi and Bluetooth sensors.

		Predicted mode accuracy (%)						
		`foot`	`bike`	`car`	`bus`	`train`	`tram`	Time (min)
Actual mode	`foot`	87	8	1	1	2	1	1068
	`bike`	2	98	0	0	0	0	69
	`car`	5	2	82	10	0	0	718
	`bus`	4	5	0	90	1	0	419
	`train`	12	0	2	3	83	0	149
	`tram`	15	3	6	1	0	75	129
Precision		91	36	96	80	81	92	2552

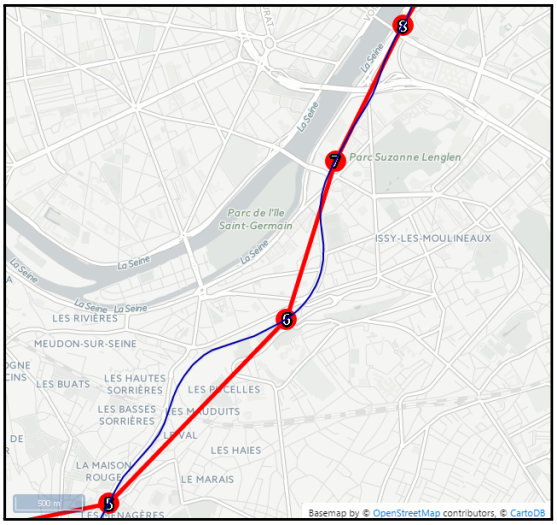
Table 5.2: The confusion matrix of *Hup-me* for the recognition of different transportation modes. The `train` mode includes the `metro` mode.

that we infer a richer itinerary, including the recognition of public transportation routes and stops. However, *Hup-me* has trouble distinguishing `bus` from `car` segments but also between underground train routes, a problem that we hope *Movup* will solve.

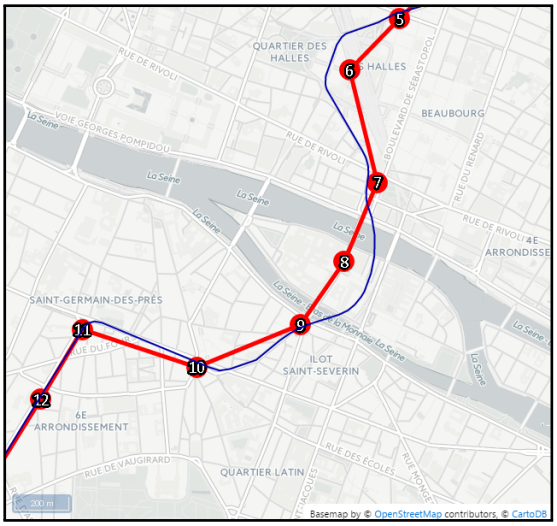
Figure 5.10: A component of Movup's annotation interface that shows the features extracted from GPS/GLONASS sensors, as well as the phone's service status and signal strength, the network connectivity, and screen status over time.

	`bus`	`train`	`tram`
Accuracy (%)	95	78	99
Total Time (min)	381	127	98

Table 5.3: *Hup-me*'s public transportation route recognition performance.



(a) RER C, a rapid transit route, in the suburbs of Paris.



(b) Line 4 of the Paris metropolitan. The section shown is in the center of Paris, and is completely underground.

Figure 5.11: The result of matching a train and a metro trip pattern extracted from Paris public transportation data to paths in the rail transportation networks created from OSM data. The red nodes represent the stops, and the numbers indicate the order of stops in the trip pattern. Consecutive stops are linked by red edges. The blue line represents the path that the trip pattern was mapped to in the corresponding transportation network.

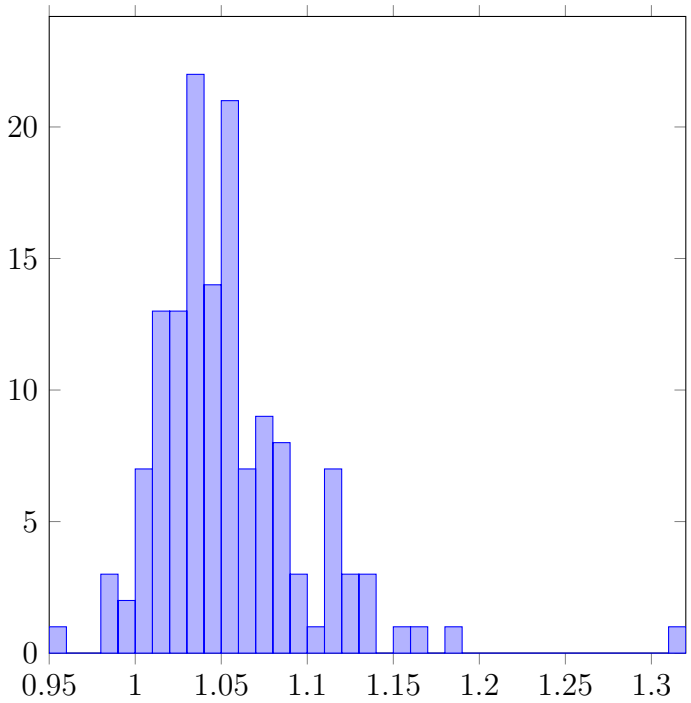


Figure 5.12: The distribution of the ratios of matched paths' lengths to the sum of geographical distances between consecutive stops of metro, train, and tram trip patterns. This average ratio is 1.053.

5.8 Related work

Activity recognition The itinerary recognition task can be seen as an advanced instance of *activity recognition*. Activity recognition has gained increased attention over the years in the fields of artificial intelligence, robotics, and ubiquitous computing. Our work follows pioneering works published over the last ten years on sensor-based activity recognition and most notably accelerometer-based approaches (Bao and Intille 2004; Ravi et al. 2005; Maurer et al. 2006; Pärkkä et al. 2006; Choudhury et al. 2008; Kwapisz, Weiss, and Moore 2011). A survey and a taxonomy of the different activity recognition techniques that applies to mobile phone sensor data can be found in Incel, Kose, and Ersoy (2013) and Shoaib et al. (2015).

Transportation mode recognition More recently, several researchers have focused on transportation mode identification (Wang, C. Chen, and Ma 2010; Reddy et al. 2010; Zheng et al. 2010; Manzoni et al. 2010; Hemminki, Nurmi, and Tarkoma 2013). Hemminki, Nurmi, and Tarkoma (2013) recognize the mode among a rich set of public transportation modes: walk, car, bus, train, metro, and tram. Using existing activity recognition techniques, we can determine at any point in time whether the user is currently *stationary*, *walking*, *running*, *cycling*, or in a *motor vehicle*. This technology is in fact already available to application developers on the two most widespread smartphone platforms, namely Android (Android Developers Guide 2013) and iOS (Apple Inc 2013). One of our

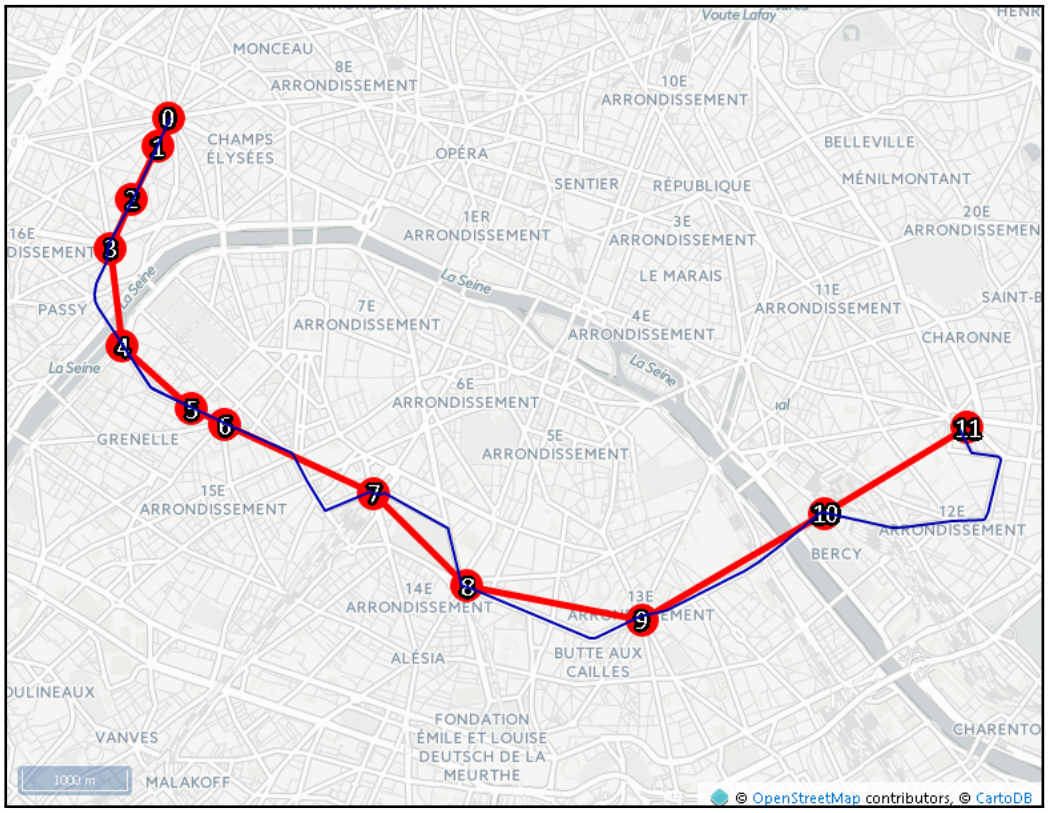


Figure 5.13: The result of matching a trip pattern belonging to Line 6 of the Paris metropolitan to a path in the metro transportation network. The red nodes represent the stops, and the numbers indicate the order of stops in the trip pattern. Consecutive stops are linked by red edges. The blue line represents the path that the trip pattern was mapped to in the corresponding transportation network. The ratio of the blue line's length to the red one is roughly 1.31.

goals is to determine the transportation modes and identify the path taken by the user over the course of her journey. To serve this purpose, and provide context awareness to public transportation navigation technologies, GPS-based activity tracking mobile applications were developed to track the user's most frequent locations, travel routines, transportation modes, and paths taken (Ashbrook and Starner 2003b; Liao, Patterson, et al. 2007; J. Chen and Bierlaire 2015). Location-based vehicle tracking has also been successfully used to automatically generate accurate public transportation maps and schedules and provide real-time information about traffic delays (Thiagarajan, Ravindranath, et al. 2009; Thiagarajan, Biagioni, et al. 2010; Biagioni et al. 2011).

Itinerary recognition Our interest is to provide Alice with an accurate summary of her itinerary. To this respect, our work is closely related to Liao, Patterson, et al. (2007); J. Chen and Bierlaire (2015). They use a combination of online activity recognition techniques and location-based map matching. However, these works do not distinguish between overlapping public transportation routes with different schedules and do not handle long periods of missing satellite-based

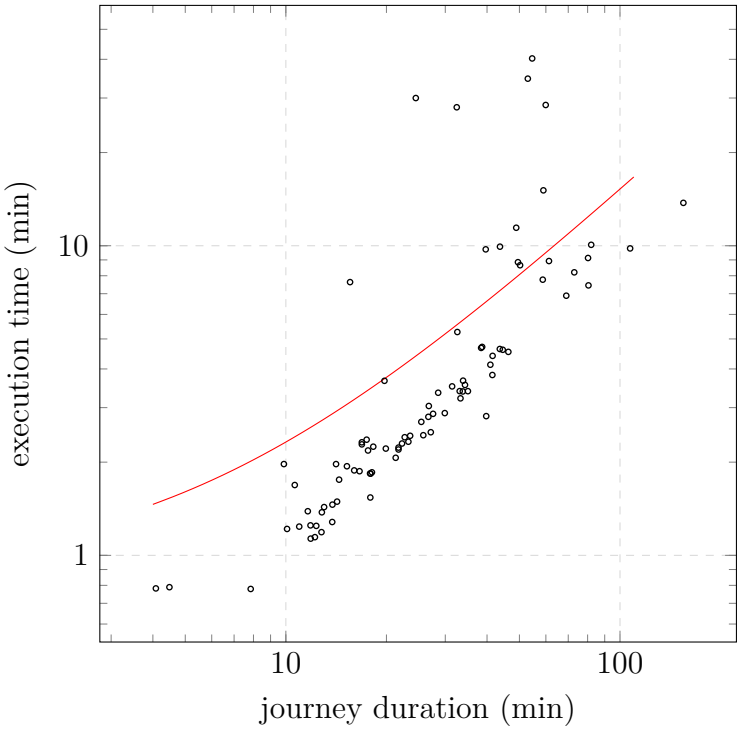


Figure 5.14: The execution time of *Hup-me* with respect to the duration of the journey.

positioning. To this respect, Thiagarajan, Biagioni, et al. (2010) provided a framework for performing tasks related to these problems: recognizing whether the user is on a vehicle, differentiating bus from car travel, identifying the bus route, and tracking the user inside an underground metro system. However, their framework was aimed toward and used for delivering improved transit information to other users rather than providing users with accurate summaries of their journeys. Their work was nonetheless one of the first to use static public transportation schedules to track the user in the transportation network. More recently, Stenneth et al. (2011) started used real-time schedule updates, but these are used for real-time tracking and are not used to infer an itinerary.

Map matching For navigating through road networks, the task of recognizing the path taken by a user in the road network, in particular from locations derived from satellite-based positioning, is known as *map matching*. Map matching requires a logical model of the world such as a spatial network (Barthélemy 2011). *Map matching* can be seen as a particular kind of *activity sequence recognition*, in which the observations are locations and the recognizable activities are positions and paths in the road network. Some indoor localization and tracking problems involve recognizing both the position (e.g., entrance, office, cafeteria) and activity (e.g., chatting, using a computer, having coffee) of multiple people in an indoor environment (Bui, Venkatesh, and West 2002; Oliver and Horvitz 2005). An exhaustive survey of map-matching techniques can be found in (Quddus, Ochieng, and Noland 2007) and more recent results in (Lou et al. 2009; Newson and Krumm

2009; Hunter, Abbeel, and Bayen 2014).

Sequence recognition For recognizing sequences of states from observations, there exist plenty of models and techniques: hidden Markov models, dynamic Bayesian networks, conditional random fields (Koller and Friedman 2009a), and recurrent neural networks (Goodfellow, Bengio, and Courville 2016b). These supervised machine learning techniques fall under the umbrella term of *structured prediction* (Bakir et al. 2007). In particular, they have been applied to great success in different natural language processing tasks (Settles 2004; Huang, Xu, and Yu 2015; Andor et al. 2016), such as speech recognition (Graves, Mohamed, and Hinton 2013) and handwriting recognition (Graves, Liwicki, et al. 2009). In our first system, *Hup-me* (Montoya and Abiteboul 2014; Montoya, Abiteboul, and Senellart 2015), we used a dynamic Bayesian network to infer the path taken by the user over a multimodal transportation network. However, as stated by Hunter, Abbeel, and Bayen (2014), dynamic Bayesian networks encounter the *selection bias problem*, which is especially problematic at low observation frequencies. Instead, they use a conditional random field, which is also the kind of model that Liao, Fox, and Kautz (2007) use the kind we use in *Movup*. Differently however, *Movup* uses a CRF whose feature functions are based on the output of a neural network. This kind of setup has been successfully applied to the part-of-speech tagging task in natural language processing (Andor et al. 2016).

5.9 Conclusion

We have presented a method for inferring the transportation mode and routes of the user from rich mobile device (smartphone) sensor data. Our work is, to our knowledge, the first one considering such a complex transportation network, i.e., a multimodal one distinguishing public transportation routes and schedules, and with such a rich combination of user data, i.e., including data from location sensors, the accelerometer, and radio networks (cellular network, Wi-Fi, and bluetooth). For instance, we take into consideration schedules to counter the possibility of geographic overlap between several transportation modes and routes. Another novelty of our approach is in the processing of long periods of time without location data, which happen frequently in high density urban areas and especially within underground transit systems. At the moment, we are conducting a thorough evaluation of *Movup* and would like to present it in the near future. We would also like to release the part of the dataset that was collected and annotated, from consenting users, so that researchers can use it.

Hup-me, our first version, showed promising results, and gave us a better understanding of the difficulties inherent to this problem. In particular, it showed how difficult it is to use a single probabilistic model to capture the many dimensions of an itinerary (i.e., the transportation mode, the route, the stops, and transition times) and relate them directly to the observations. It may seem that the state space required by such a model would need to be large in order the take into account the important aspects of the problem:

- The need to use segment-wise features for distinguishing public transportation modes and routes, e.g., the vehicle's trip time and possible schedule delay, and the number and frequency of stops over this period. Classic probabilistic models for sequence recognition use fixed-length time frames and inter-frame independence assumptions to model itinerary states and their relationships, which is not suited for calculating segment-wise features.

- The need to consider *stationarity* as an extra dimension in addition to the user's transportation mode when performing transportation mode sequence prediction. For instance, when Alice is on foot or when she travels in a bus, she or the bus may stop at a red light. Assuming that whenever the bus stops its engine also stops (e.g., if the vehicle has an electric engine or a start-stop system), the sensor observations made during a stop period may not be sufficient to distinguish between foot and bus modes. During such a period, the prediction system should indeed recognize Alice as stationary, but the prior probabilities that the prediction transitions during the next time frame into a *foot* or *bus* moving state should take into account the fact that everytime the bus stops, Alice will more often stay on it than she will alight from it. Similarly, modeling the possibility that Alice may walk within a public transportation vehicle such as a bus, a train, a tram, or a metro may also be useful, thus differentiating between the user's *human activity* and her transportation mode (e.g., *pedaling* on a *bicycle*, or *sitting* on a *bus*).

- Recognizing underground public transportation routes within the user's journey requires a specialized kind of processing. On the one hand, the information that the user is underground and moving inside a vehicle serves to limit the possible transportation routes to underground ones, which reduces the state space. On the other hand, continuously missing location information may increase over time the size of the estimated area in which the user may be, increasing possible number of candidate routes. The most useful information in this case is the topology of the underground network and the number of stops and time that are required by each route to move from one place to another.

Finally, one of the main motivations behind the work presented in this chapter is to enrich Alice's personal knowledge base, so that we can use it for more advanced personal analytics or to help an intelligent personal assistant provide more relevant navigation feedback. In the next chapter, we present a system that we designed to realize such a personal knowledge base.

Chapter 6

Personal knowledge integration

This work has been carried out in collaboration with Thomas Pellissier Tanon from ENS Lyon, who spent some months in our team at Inria as part of his master's internship, and researchers from Télécom ParisTech, namely Fabien M. Suchanek and Pierre Senellart. Part of the content of this chapter was presented in (Montoya, Pellissier Tanon, et al. 2016), where our PIM system was demoed and a short overview of it was provided. Dominique Tran and François Camus from ENGIE also contributed in the making of this demo. This chapter includes a more thorough description and evaluation of this system.

In this chapter, we present Thymeflow[1], a system that we designed to investigate the notion of a *personal information management system* (cf. Section 1.4). With this system, the goal is to put Alice back in control of her personal information. We present a description and evaluation of this system.

6.1 Introduction

The system's main goal is *knowledge integration* (introduced in Section 2.3) from multiple heterogeneous sources of personal information. Knowledge is integrated into a *personal knowledge base* (KB) (see Section 2.2 for a definition), providing Alice with a high-level global view of her personal information, which she can use for querying and analysis. This integration, together with powerful query capabilities, are needed for performing more relevant *personal analytics* (understood as in Section 2.3), and possibly prediction and recommendation.

The system is open-source[2], meant with extensibility in mind, and designed to be installed on a machine that Alice controls, i.e., on her personal computer or on a private server. In such a setting, her privacy is preserved by the fact that the KB remains on this machine, and that all integration and analysis happens locally.

The system's secondary goal is to facilitate the automatic enrichment of personal information. It provides a framework for running sophisticated algorithms

[1] `http://thymeflow.com/`

[2] `https://github.com/thymeflow`

using the knowledge present in the KB to automatically derive new facts. This framework is itself used to implement some of the tasks involved in knowledge integration, such as entity resolution and spatiotemporal alignments. To test the framework's ability to generalize to other algorithms, we also used it to implement the stay extraction algorithm, *Thyme*, that we presented in Chapter 4

Finally, we do not want to create a new information silo via this system. First, we do not want this system to re-implement functionalities existing in more specialized tools, such as reading and writing email messages, calendar management, or document filing. Second, the system allows knowledge to be pushed from the KB back to the sources, effectively implementing two-way synchronization (cf. Section 2.3). Via this functionality, we want Alice to keep using her applications as usual while enjoying augmented knowledge derived by Thymeflow. Additionally, this functionality can provide the basis for automatic multi-device and multi-system synchronization.

Challenges Designing such a personal KB is challenging. Data of completely different nature has to be modeled in a uniform manner, pulled into the knowledge base, integrated with other information, and kept synchronized with the sources when changes occur. 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, our integration task spans information of different modalities, for instance, if we want to create a coherent user experience, we have to match events in the user's calendar with locations in her location history and place names.

Results Our system is fully functional and implemented. It can, e.g., provide answers to questions such as:

- Who has been contacted the most by Alice in the past month? (Requiring alignments of the different email addresses.)
- Where did Alice have lunch with Bob last week? (Requiring an alignment between the calendar and the location history.)
- How many times did Alice go to Bob's place last year? (Requires an alignment between the contact list and the location history.)
- What are the telephone numbers of the Alice's birthday party guests? (Requiring an alignment between the calendar and the contact list.)

This chapter is structured as follows: Section 6.2 describes the system, its architecture and implementation. Section 6.3 details our knowledge enrichment processes. Section 6.4 discusses experimental results. Section 6.6 the related work, and Section 6.7 concludes

6.2 The system

Model Thymeflow's KB implements the representation model of a *personal knowledge base* described in Section 2.2, which is based on an RDFS ontology. In summary, this models captures the concepts of *events*, *agents*, *messages*, *locations*, *stays* and *places*. Additionally, it captures the provenance of knowledge (using RDF named graphs), which it uses to distinguish between *observed* and *inferred* statements. Finally, it allows representing that two instances of a concept (identified by two distinct IRIs) are *facets* of the same real-world entity.

Mediation vs. Warehouse One can first see the KB as a virtual view over the personal information sources. The sources are queried in a mediation style (Garcia-Molina et al. 1997), loading the data only on demand. However, accessing, analyzing and integrating these sources on the fly are expensive tasks. In some cases, the integration may require iterating through the entire source, which can be prohibitively costly in the case of emails. Also, we want to avoid having to redo the same inferences. For instance, suppose that we had automatically inferred that an event in the calendar took place at a certain location. We do not want to infer this piece of information again. Our approach therefore loads the sources into a persistent store, in a style that enriches the data warehouse.

System overview Our system is a web application that Alice installs on her machine. Using its web interface, she provides the system with a list of information sources (such as email accounts, calendars, or address books), together with authorizations to access them (such as tokens or passwords) (Figure 6.2). The system accesses these information sources (as she would), and pulls in the data. It uses adapters to access the sources, and to transform the data into RDF. The supported adapters are described in Chapter 3. They may access Alice's email messages, address books, calendars, social network services (Facebook), and location history. While new information is being loaded, the system automatically performs inferences and the whole is consolidated in the KB (a triple store). Since the KB is persistent, the system can be stopped and restarted without losing information. At any time, she may use the interface to query the KB using the SPARQL 1.1 query language (Harris and Seaborne 2013) (extended with full-text search of RDF literals), and visualize the results on a table, on a map or as a graph. She may also write SPARQL update queries (Polleres, Passant, and Gearon 2013) to change the knowledge contained in the KB. The system runs locally on Alice's machine, and she may also define the kind of inferences that may be automatically performed. Thus, she remains in complete control of her data.

Architecture One of the main challenges in the creation of the KB is the temporal factor: information in the sources may change, and these updates have to be reflected in the KB. These 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, our system uses software modules called *synchronizers*

and *enrichers*. Figure 6.1 shows the synchronizers $S_1, \ldots, S_n$ on the left, and the enrichers $E_1, \ldots, E_p$ in the center. Synchronizers are responsible for accessing the information sources. Enrichers are responsible for inferring new statements, such as alignments between entities obtained by entity resolution.

These modules are scheduled dynamically. For example, some modules may be triggered by updates in the information sources (e.g., an update in a calendar event) 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 (e.g., daily) for particularly costly alignment processes. Generally, when a synchronizer detects a change in a source, it triggers the execution of a pipeline of enricher modules, as shown in Figure 6.1. Enrichers can also use knowledge from external data sources, such as Wikidata (Vrandečić and Krötzsch 2014), Yago (Suchanek, Kasneci, and Weikum 2007), or OpenStreetMap, as shown in the figure.

Assuming that the output of enrichers is deterministic and does not depend on the order in which updates are processed, the content of the KB is entirely determined by the information contained in the sources. Inferred knowledge is not automatically reflected in the sources. However, Alice's update queries may result in knowledge being pushed back to the sources. The Updater module is responsible for this, as we describe in Section 6.2.

Loading

Synchronizer modules are responsible for detecting changes in an information resources, retrieving new information, and pushing the updated knowledge to the Loader. They implement the adapters described in Section 2.2 to extract personal information from various sources and transform them into RDF statements. The adapters use synchronization mechanics to efficiently detect and retrieve new information in a source, which is packaged into a *document* (see Section 2.2 for a definition) containing the new knowledge and meta-information about the source. This is of course relatively simple for information sources that provide an API supporting changes, e.g., CalDAV. For others, this requires more processing. The Loader persists the knowledge contained within each new *document* into the KB, and transforms it into a delta Δ_0. Δ_0 consists of a set of RDF statement insertions and a set of deletions. The Loader then triggers the pipeline of enrichers with this delta.

Enrichment

After loading, enricher modules perform inference tasks such as entity resolution, event geolocation, and other knowledge enrichment tasks. We distinguish between two kinds of enrichers. The first kind takes as input the entire current state of the KB and applies to it a set Δ of enrichments (i.e., new statements). For instance, this is the case for the module that performs entity resolution for agents. The second type of enricher works in a differential manner: it takes as input the current state of the KB, and a collection of changes Δ_i that has been pushed by

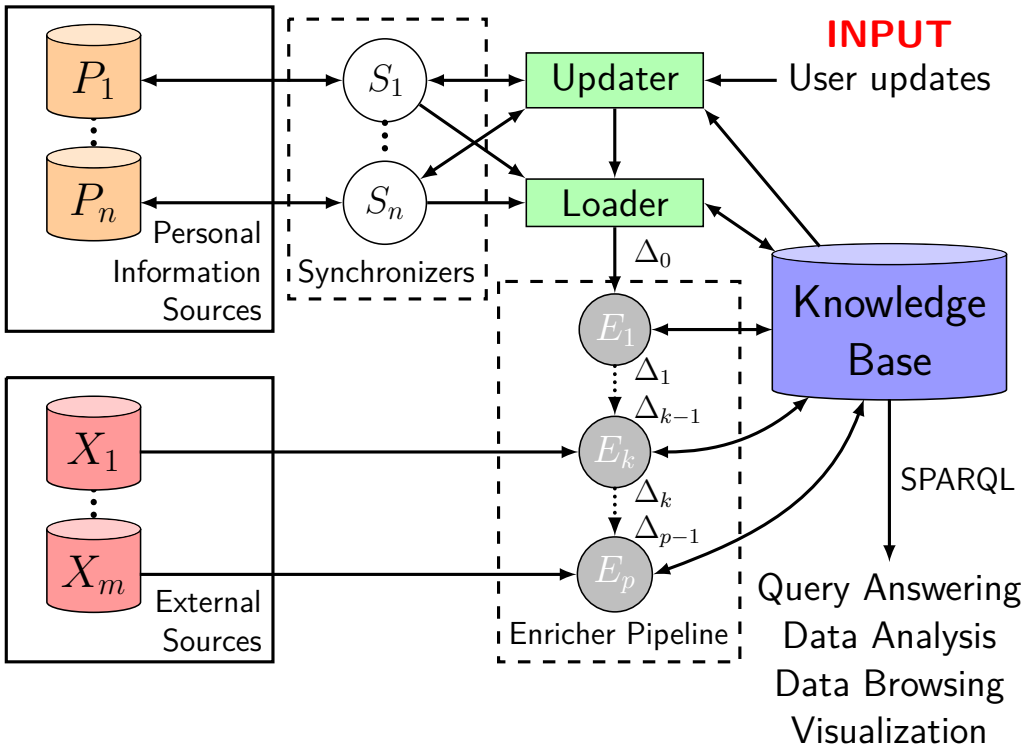


Figure 6.1: The system architecture of Thymeflow.

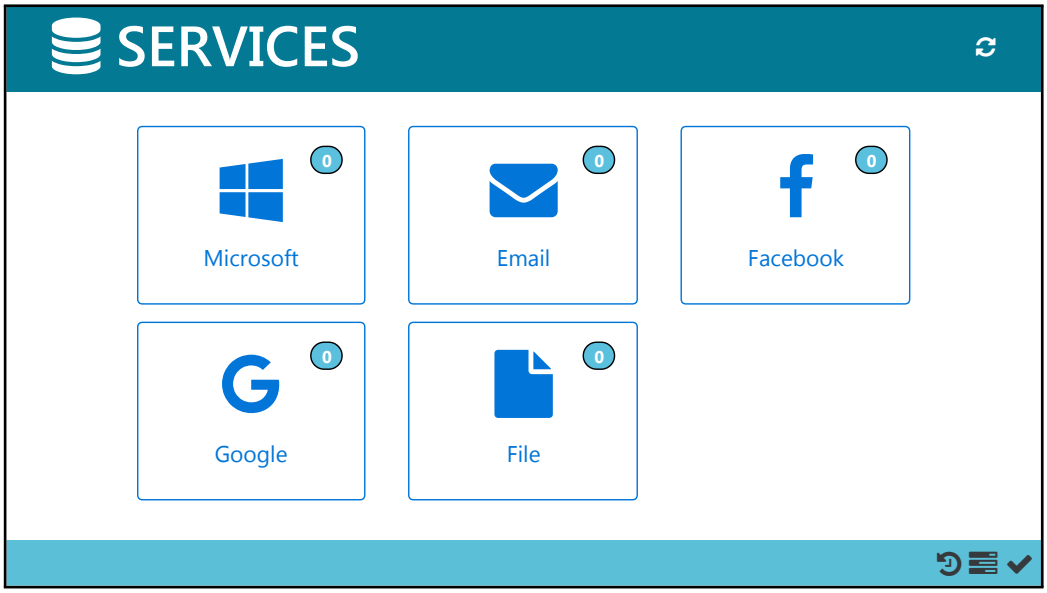


Figure 6.2: The web user interface of Thymeflow for configuring new sources. Alice can authenticate to any of the above service providers to connect them to the system. She can also directly upload files (e.g., vCards) to the system.

the Loader or an enricher that comes before it in the pipeline. It computes a new collection Δ_{i+1} of enrichments that it pushes down the pipeline. Intuitively, this allows reacting to changes of an information source. For instance, when a new event entry is entered in the calendar with an address, a geocoding enricher is called to attempt to locate it. Another enricher will later try to match it with a position in the location history.

Provenance

The system records the provenance of each newly obtained piece of information. For synchronizers, this is the information source, and for enrichers it is the enricher itself. As presented in Section 2.2, knowledge is organized in *documents*, which are collections of statements with the same provenance..

For instance, the statements extracted from an email message by an email synchronizer connected to Alice's email server via IMAP will be packaged into a document that is identified by an IRI that is the concatenation of the server's URL, the message's folder path, and the message's id. The source is the email server's URL, which is an instance of `personal:ObservationSource`. In order to gather different kinds of data sources (e.g., CardDAV, CalDAV, and IMAP servers) that belong to a single provider (e.g., Google or corporate IT services) and to which Alice accesses using the same account, we use an instance of `personal:Account`. Sources that belong to the same account are related to this account via the `personal:sourceOf` property.

On the other hand, the statements inferred by an enricher are packaged into documents whose associated source is the enricher. Each enricher has its own IRI and is an instance of `personal:InferenceSource`.

Provenance in our system can be used for answering queries such as "What are the meetings scheduled for next Monday that are recorded in Alice's work calendar?" For this we need to use the provenance of the meetings, i.e., the work calendar.

Pushing to the source

Finally, the system allows the propagation of information from the KB to the sources. These can be either insertions/deletions derived by the enrichers or insertions/deletions explicitly specified by Alice. For instance, consider the information that different email addresses correspond to the same person. This information can be pushed to the sources, which may for example result in performing the merge of two contacts in Alice's list of contacts. To propagate the information to the source, we have to translate from the structure and terminology of the KB to that of the source and use the API of that source.

Some sources hold information that cannot be updated from the KB, for example email messages. By contrast, CardDAV contacts can be updated. Alice can update the KB by inserting or deleting knowledge statements. Such updates to the KB are specified in the SPARQL Update language (Polleres, Passant, and Gearon 2013). The Updater is responsible for intercepting these updates, and

propagating them to the sources or sending them to the Loader, which makes them persistent and then runs the enricher pipeline. The semantics of propagation is as follows.

If a target is specified (by containing the statements in the named graph of a source), the system tries to propagate the new information to that specific source. If the operation fails, the error is reported to Alice.

Now consider the case when she does not explicitly specify a source where to propagate. For an insertion, the system tries to find all applicable data sources for the operation. All sources where the subject of the statement is already described (i.e., all sources that contain a statement with the same subject) are selected. If no source is able to register an insertion, the system performs the insertion in a special graph in the KB, the *overwrite graph*.

For a deletion, all sources that contain the deleted triple are selected. Then the system attempts to perform the deletion on the selected sources. If one source fails to perform a deletion (e.g., a source containing the statement is read-only), the system removes the statement from the KB anyway (even if the data is still in some upstream source) and adds a negated statement to the *overwrite graph* to remember that this statement should not be added again to the KB. The negated statement has the same subject and object as the original statement but uses a negated version of the predicate. It should be seen as overwriting the source statement. Alice can also specify the deletion of a statement inferred by an enricher (either a specific one with a target, or any). Besides the deletion of the statement from the KB, a negative statement is also added to the *overwrite graph*. This prevents enrichers from rederiving these statements.

Implementation

The system was implemented as two separate components: the *back end* and the *front end*.

Back end The back end's essential components are the supervisor, the synchronizers, the Loader, the Updater, the enricher pipeline, the KB (a triple store), and a web API. It is written in Scala (EPFL 2017) and uses the Akka Actor Model (Lightbend Inc. 2017) to orchestrate the different interactions between its components. This model is based on actors, which are lightweight processes that asynchronously communicate with each other through message passing.

The supervisor is responsible for launching the different actors and ensures the recovery of each actor in the case it crashes. In our implementation, each synchronizer runs on a separate actor and sends messages containing *documents* to the Loader when a source has new information. The Loader is also an actor. It maintains a connection to the KB that it uses to persist new information and calculate the first delta (Δ_0). To trigger the enricher pipeline, the Loader sends a message to it containing this delta.

The enricher pipeline runs on a single actor that is responsible for providing the first enricher with each received Δ_0 and then transmitting the delta output by each enricher to the next enricher in the pipeline. Since information in sources

may significantly change or sources may be plugged at any time, the system needs to be able to extract and process a large volume of information at any time. To efficiently manage this volume of information, we use Akka's implementation of Reactive Streams (Reactive Streams Special Interest Group 2017), which allows the system to put back-pressure on the synchronizers so to slow them down in the case the loader or the enricher pipeline still have a lot of deltas to process. This prevents synchronizers from building up buffers of unbounded amounts of data. Synchronizers are responsible for keeping documents small so as to ensure that the pipeline's buffers take a bounded amount of memory.

The back end's KB is a Sesame based triple store (Broekstra, Kampman, and Harmelen 2002), which is known as RDF4J (Eclipse Foundation 2017). It is equipped with a SPARQL 1.1 compliant engine (Harris and Seaborne 2013) and a full-text search extension based on Apache Lucene (The Apache Software Foundation 2017b).

Finally the back end's web API consists of a SPARQL endpoint and a RESTful JSON endpoint. SPARQL read queries are executed directly by the triple store and allows arbitrary querying of the KB. SPARQL update queries are sent to the Updater, which asynchronously asks the synchronizers if they can handle the updated statements and stores them in the KB if not. The RESTful endpoint is used for configuring the synchronizers, and querying the current status of the system (i.e., what synchronizers are running and their status). The web API is implemented using Akka HTTP.

Front end The front end implements the web user interface as a single-page web application. It is written using Ember.js (Tilde Inc. 2017), which is a framework for implementing single-page web applications following the model-view-view-model pattern. It compiles to a set of HTML, CSS, JavaScript files that are served to Alice's browser by a web server that is separate from the back end. The front end asynchronously communicates with the back end using the SPARQL and RESTful endpoints.

The front end is composed of a component for configuring new sources, another for querying and visualizing results, and two specialized components: *contacts* and *timeline*. The *contacts* component shows a list of all the `personal:Agent` entities in the KB. For each `personal:Agent` entity, the component also lists the attributes of each its facets and their respective sources (Figure 6.3). The *Timeline* component is used to visualize `personal:Stay` instances, and the links they have to events (Figure 4.5b). The visualization component uses the OpenLayers (Team 2017) library for map visualizations and the D3.js (Bostock 2017) library for graphs visualizations. Finally, the front end has a status bar that displays the current status of the system and keeps a history of the SPARQL queries that have been run.

Extensibility It is straightforward in Thymeflow to plug new information sources, new enrichment algorithms (i.e., inference, analytics), or new ontologies. In particular, the architecture clearly separates knowledge extraction from enrichment facilities. To facilitate the development of synchronizer or enricher modules,

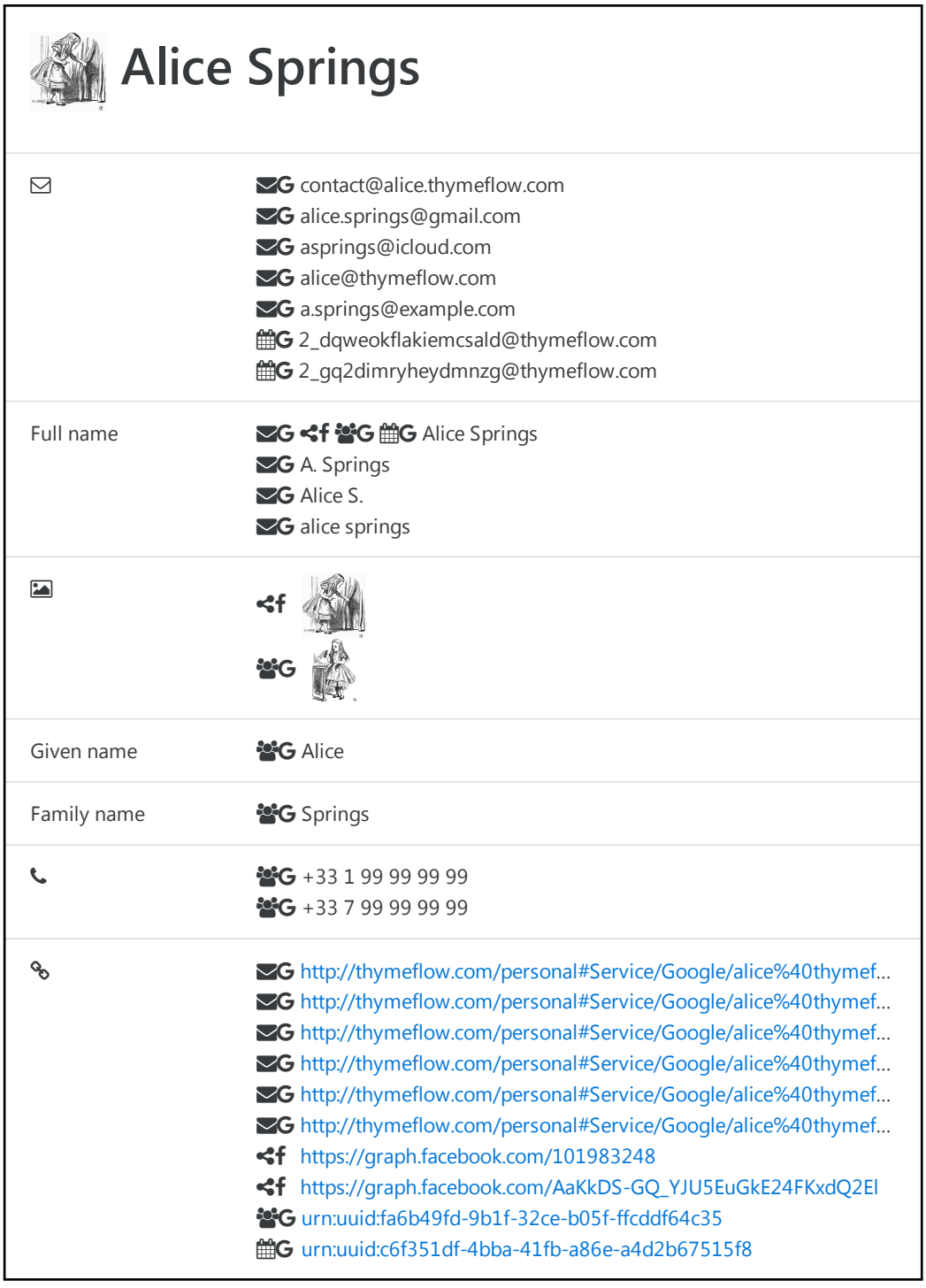


Figure 6.3: A view of Alice's own *agent* entity in Thymeflow's *contact* component. Her different email addresses, names, pictures, and telephone numbers are shown. Each attribute has one or many provenance annotations (e.g., Facebook, her Google calendar, her email, or her address book). At the bottom, the different facets of this entity are shown, identified by their IRIs.

an API is provided for modeling knowledge and handling differential updates. The systems encourages the reusability of converters of small units of information, such as the normalization of a telephone number and its representation in RDF. However, the code is not yet modular and requires that synchronizers and enricher modules be compiled within the same project.

6.3 Enrichers

In this section, we describe three specific enrichers included in our system: stay extraction, agent matching, and event geolocation.

Stay extraction

The system implements the stay extraction algorithm described in Chapter 4 as an enricher. This enricher processes `personal:Location` instances and outputs a set of `personal:Stay`s that it persists in the KB.

Agent matching

The KB keeps knowledge as close to the original sources as possible. Thus, the knowledge base will typically contain several *facets* of the same person, if that person appears with different names and/or different email addresses. We call *agent matching* the task of identifying equivalent *agent facets*. The task of identifying facets of the same real-world entity is known as *entity resolution* in the context of information integration (cf. Section 2.3). In our case, we use techniques that are tailored to our personal KB: 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) that determines the entity in some sense. This approach is commonly used in entity resolution Section 2.2 to achieve a high precision result, but the result may suffer from low recall. In our case, it gives fairly good results for linking agent facets extracted from email messages with the ones extracted from address books. We are aware that 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.

Attribute-based matching Identifier-based matching cannot detect that jdoe@gmail.com and john.doe@hotmail.com are the email addresses of the same person. In some cases, attributes other than email addresses may help. For instance, two agents instances with the same given and family names have a higher probability to represent the same entity than two instances with different names, all other attributes being equal. In our schema, the following attributes can help

the matching process: `schema:name`, `schema:givenName`, `schema:familyName`, `schema:birthDate`, `schema:gender`, and `schema:email`.

We tried the holistic instance matching algorithm suited for RDF graphs described in Suchanek, Abiteboul, and Senellart (2011), which we adapted for our setting. The results turned out to be disappointing (cf. experiments in Section 6.4). We believe this is due to the following:

- Missing attributes: for instance, almost all agent instances have a `schema:email` and possibly also have a `schema:name`, but most of them lack `schema:givenName`, `schema:familyName`, `schema:gender`, and `schema:birthDate` attributes.

- Imprecision: names extracted from email messages may contain pseudonyms, abbreviations, or may lack family names, and this reduces matching precision. Some attributes may appear multiple times with different values.

- Lack of good priors: we cannot reliably compute name frequency metrics from Alice's knowledge base or an external one, since duplicates may exist even within a single information source or it may be the case that a rare name appear frequently for different agents instances if a person with that name happens to be a friend of Alice.

Therefore, we developed our own algorithm, *AgentMatch*, that works as follows:

1. For each agent instance a, let $\rho(a)$ be the number of events or messages it is directly related to and let $\omega(w, a)$ be the occurrence frequency of a word w in the set of `schema:name` attributes of a (e.g., the occurrence frequency of "John" in a class whose names are "John Doe" and "Johnny Doe" is 0.5)

2. The set of `personal:Agent` instances is partitioned using the equivalence relation computed by the *Identifier-based Matching* technique previously described. The set of equivalence classes is denoted $\mathcal{C}$.

3. For each equivalence class $C \in \mathcal{C}$ and each word w, let $\psi(w, C)$ be the arithmetic mean of the occurrence frequencies of all the instances a in C (i.e., the $\omega(w, a)$'s for $a \in C$) weighted by their respective $\rho(a)$. $\psi(w, C)$ is called the occurrence frequency of w in C.

4. For each word w found in the set of `schema:name` attributes, the *inverse document frequency* $\mathrm{idf}(w)$ of this word is computed, where the set of documents is the set of equivalence classes $\mathcal{C}$ and the number of documents where w appears is replaced by the sum of w's occurrence frequencies in the different equivalence classes:

$$\mathrm{idf}(w) = \log \frac{|\mathcal{C}|}{\sum_{C \in \mathcal{C}} \psi(w, C)}.$$

5. Let f be a string similarity metric between words. Let a best f-matching of words between two strings A and B be a maximum weight matching in a weighted complete bipartite graph whose parts U and V are respectively the words in A and B and in which the weight of an edge is the f-similarity between its nodes. Let d_f be a string distance function that first finds a best f-matching of words between two strings, and then returns the arithmetic mean of the f-similarities between the pairs of matched words weighted by the sum of their respective inverse document frequencies.

6. The similarity $\mathcal{F}_f$ (between 0 and 1) between two equivalence classes is the arithmetic mean of the d_f-similarities between the `schema:name` attributes in one class and the ones in the other class weighted by the product of their respective occurrence frequencies.

7. The pairs for which the similarity $\mathcal{F}_f$ is above a certain threshold are considered to be facets of the same real-world entity.

The string similarity metric f that 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, the algorithm uses a 2/3-gram-based index of the words in `schema:name` attributes, and only consider in Step 6. of the above process the pairs of equivalence classes with a ratio of q-grams in common for at least one word greater than or equal to some value S. For instance, two classes whose respective `schema:name` attributes are "Susan Doe" and "Susane Smith" would be candidates. We evaluate the performance of these agent matching techniques in our experiments.

Geolocating events

In this section, we discuss the geolocation of events, i.e., how an enricher providing this functionality can detect for example that Monday's lunch was at "Shana Thai Restaurant, 311 Moffett Boulevard, Mountain View, CA 94043". For this, the input is the set of stays extracted by the *stay extraction* enricher from Alice's location history. The enricher performs spatiotemporal alignment between these stays and the events in her calendars. Finally, geocoding is used to provide semantics to the event's location, such as the name of the place and the street address.

Matching stays with events Matching calendar events with stays turns out to be difficult because:

1. The location of an event (an address and/or geo-coordinates) is often missing.

2. When present, an address often does not identify a geographical entity, as in "John's home" or "room C110".

3. In our experience, start times are generally reasonable (although a person may be late or early for a meeting). However, the duration is often not meaningful. For instance, around 70% of the events in our test datasets were scheduled to last one hour. However, among the 1-hour events that were matched, only 9% lasted between 45 and 75 minutes.

4. Some stays are incorrect.

Because of (1) and (2), the enricher does not rely much on the event locations that appear in Alice's calendars. A stay is matched to an event primarily based on time: the time overlap (or time proximity) and the duration. In particular, a stay is matched to an event if the ratio of the duration of the overlap period to the duration of the entire stay is greater than a threshold θ.

As we have seen, event durations are often unreliable because of (3). Our method still yields reasonable results because it tolerates errors on the start time of the stay. For long stays, this is because of their long duration, while for short ones, this is because calendar events are usually scheduled for at least one hour. If the event has geographic coordinates, we filter out stays that are too far away from the location defined by these coordinates (i.e., when the distance is greater than δ). We discuss the choice of θ and δ for this process in our experimental evaluation in Section 6.4.

Geocoding event addresses Once stays have been matched to events, the idea is to attach richer place semantics (e.g., the country, street name, postal code, or place name) to events. If an event has an explicit address, a *geocoder* is used. A geocoder takes as input a raw street address or the name of a place (such as "Stanford") and returns the geographic coordinates of matching places, as well as structured street address and place information. The enricher only keeps the geocoder's most relevant result and adds its attributes (geographic coordinates, identifier, street address, etc.) to the location in the knowledge base. The system allows the use of different geocoders, including a geocoder that combines the results from both Google Places and Google Maps Geocoding APIs (Google 2017b), and Photon (komoot 2017), which is a geocoder that uses OpenStreetMap data.

Geocoding events using matched stays For events that do not have an explicit address but that have been matched to a stay, we use a geocoder to transform the geographic coordinates of the stay into a list of nearby places. The most precise result is added to the location of the event. If an event has an explicit address and has been matched to a stay, we call the geocoder on this address, while restricting the search to a small area around the location of the stay.

6.4 Experiments

In this section, we present the results of our experiments. We used datasets from two users, Angela and Barack (actual names changed for privacy). Angela's

	load from				restart from	
	remote services		local files		local KB backup	
w/ email bodies & full-text search	yes	no	yes	no	yes	no
MacBook Air 2013 (Intel i5-4250U 2-core 1.3GHz 4GB RAM, SSD)	28	14	13	7.0	0.70	0.67
Desktop PC (Intel i7-2600k 4-core 3.4GHz 20GB RAM, SSD)	19	10	4.0	2.6	0.22	0.20

Table 6.1: The loading time performance of Angela's dataset into Thymeflow's KB (in minutes). Angela's dataset leads to the creation of 1.6M triples.

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 when loaded into the KB. Barack's dataset consists of 136,301 emails, 3,080 calendar events, 1,229,245 location points, and 582 contacts extracted from the same kinds of sources. Within the KB, this corresponds to 10.3M triples.

KB construction

We measured the loading times (Table 6.1) of Angela's dataset in three different scenarios: (1) the dataset is loaded from remote services (using Google's APIs, except for the location history which is not provided by the API and was loaded from a file), (2) the dataset is stored in local files, and (3) restarting the system from a local backup file of the knowledge base. Numbers are given with and without the loading of email message bodies and full-text search support. In general, loading takes in the order of minutes. Restarting from a backup of the knowledge base takes only seconds.

Stay extraction

The results of the performance evaluation of stay extraction were presented in Chapter 4. We use the output of this enricher, i.e., a set of *stays*, for evaluating the enricher that matches stays with events and possibly geocodes them using the information from the stay.

Agent matching

We evaluated the precision and recall of AgentMatch, the algorithm for agent matching described in Section 6.3, 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 `Agent`s, and asked Barack to indicate which of the sampled pairs were equivalent agent facets. Since pairs of non-equivalent instances considerably outnumber pairs of equivalent instances, we performed stratified sampling by partitioning the dataset based on the matches returned by AgentMatch for threshold values between 0 and 1, in steps of 0.05. Figure 6.4 shows the distribution of `Agent` classes by number of distinct emails within the class for classes output by AgentMatch. The identifier-based matching baseline (further simply called IdMatch) is able to reduce the number of distinct agents from 40,483 to 24,677 (61%), while AgentMatch, for a threshold of 0.825, reduces it to 21,603 (53%).

We evaluated four different versions of AgentMatch. We tested both Levenshtein and Jaro–Winkler for the f string similarity metric, as well as with and without inverse document frequency weights. The word q-gram match ratio threshold S was set to 0.6. For each version of AgentMatch, Table 6.2 presents the value of the threshold λ for which the F1-measure is maximal. Precision decreases while recall increases for decreasing threshold values (Figure 6.5).

For comparison, we also considered the Google Contacts's "Find duplicates" feature, and PARIS (Suchanek, Abiteboul, and Senellart 2011), an ontology alignment algorithm that is parametrized by a single threshold. We also tested Mac OS X's contact deduplication feature, but while it was able to identify 29,408 distinct agents, the resulting merged contacts did not include all the metadata from the original ones, which were needed to evaluate the matching.

Google was not able to handle more than 27,000 contacts at the same time, and could not detect all duplicates at once, so we had to run it multiple times in batches until convergence. However, we noticed that the final output was dependent on the order in which contacts were input, and present two results: one in which the contacts were supplied sorted by email address (Google1) and another in which the contacts were supplied in a random order (Google2). Since we noticed that Google's algorithm failed to merge contacts that IdMatch had merged, we also tested running IdMatch on Google's output (GoogleId) for both runs. For PARIS, we used the f string similarity metric for email addresses, and used AgentMatch's F_f similarity metric for names, except that it was applied to single `Agent` instances instead of equivalent classes. 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 on the output of AgentMatch. Finally, we also show the performance of the parameter-free IdMatch baseline.

Our experiments (Table 6.2) show that the highest F1-measure is reached for AgentMatch with Jaro–Winkler distance for a threshold of 0.825, for which AgentMatch has a precision of 0.954, a recall of 0.945, and an F1-measure of 0.949. It also out-performs PARIS by a large margin – for reasons that we discussed in Section 6.3. GoogleId and IdMatch favor precision, reaching 0.997 and 1.000 respectively, with a recall of 0.625 and an F1-measure of 0.768 for GoogleId, while IdMatch only has a recall of 0.430 and an F1-measure of 0.601. However, by changing AgentMatch's threshold, one can reach similar precision levels for higher recalls. For instance, AgentMatch with the Jaro–Winkler distance and

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

Table 6.2: Precision and recall of AgentMatch, IdMatch, Google's algorithms, and PARIS on Barack's dataset for varying parameters.

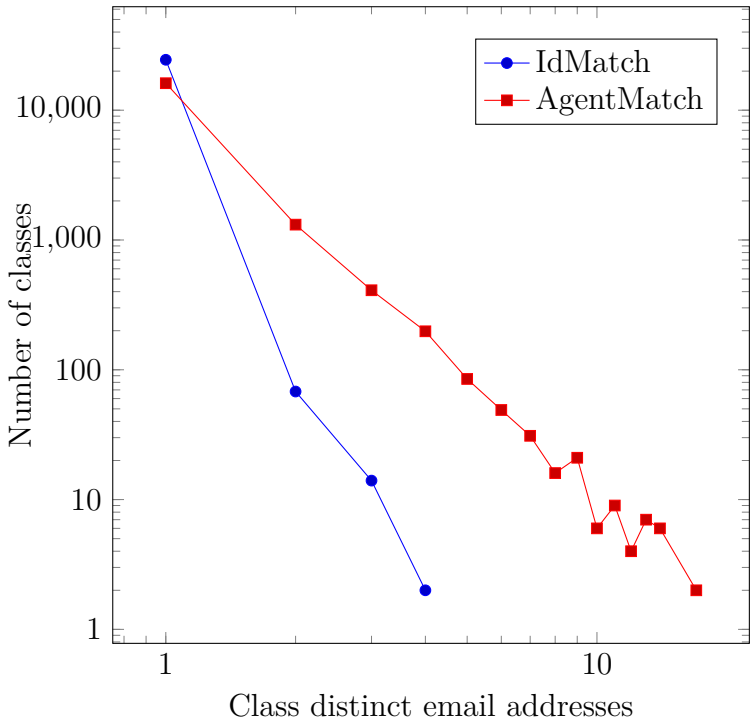


Figure 6.4: Distribution of `Agent` equivalence classes by number of distinct email addresses for matchings generated on Barack's dataset by IdMatch and the best run of AgentMatch.

IDF weights has a precision of 0.992, a recall of 0.810 and a F1-measure of 0.892 for a threshold of 0.975.

Geolocating events

Matching stays with events We evaluated the matching of stays with calendar events on Angela and Barack's datasets. On Angela's dataset, we considered

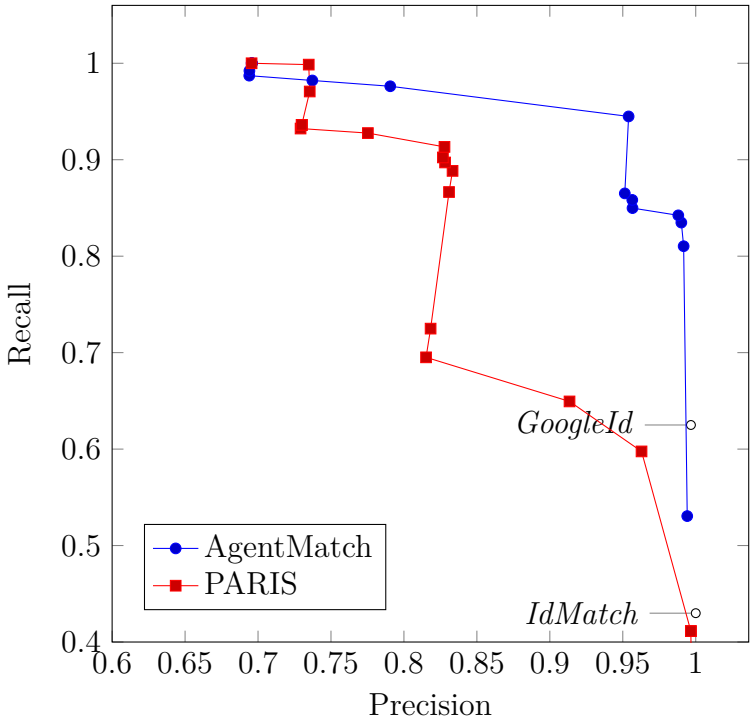


Figure 6.5: Precision-recall curves of AgentMatch and PARIS on Barack's dataset for different thresholds. IdMatch and GoogleId are given as a reference. Precision decreases while recall increases for decreasing thresholds.

a period of one year. This year consists of 129 events and 11,287 stays (as detected by our system) with an average duration of 44 minutes. Candidate matches, that is, the set of all stay-event matches outputted by the algorithm whatever the chosen thresholds, were manually labeled by Angela. For Barack, we ran the matching algorithm on the full dataset (4.5 years), and then we asked Barack to label the candidate matches for 216 events picked uniformly at random. Barack's dataset consists of 3,667 events and 49,301 stays totaling 3,105 candidate matchings. The process of matching stays with calendar events relies on two parameters, namely the overlap duration ratio threshold θ and the filtering distance δ (cf. Section 6.3). Figure 6.6 shows how precision and recall vary depending on θ for a filtering distance set to infinity. It shows some sensibility to θ; the best results are obtained for a value of around 0.15. With θ set to 0.2, Figure 6.7 shows the impact of the filtering distance δ. The performance slowly increases with increasing values of δ. This indicates that filtering out stays that are too far from the location defined by the event's coordinates (where available) should is not necessary and we have disabled this filter in the final version of this matching implementation (or equivalently, set δ to infinity). In general, the matching performs quite well: We achieve a precision and recall of around 70%.

Geocoding We experimented with the different geocoding enricher techniques described in Section 6.3 on Barack's dataset. On this dataset, we evaluated the performance of geocoding event–stay matches on Barack's dataset from three

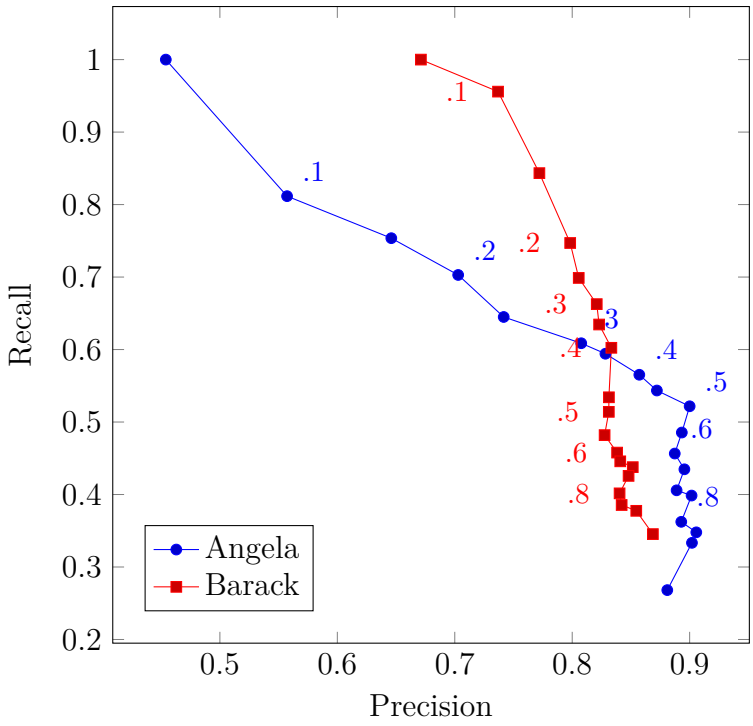


Figure 6.6: Precision-recall curves of matching stays with events for different values of the overlap duration ratio threshold θ and for a filtering distance δ set to infinity.

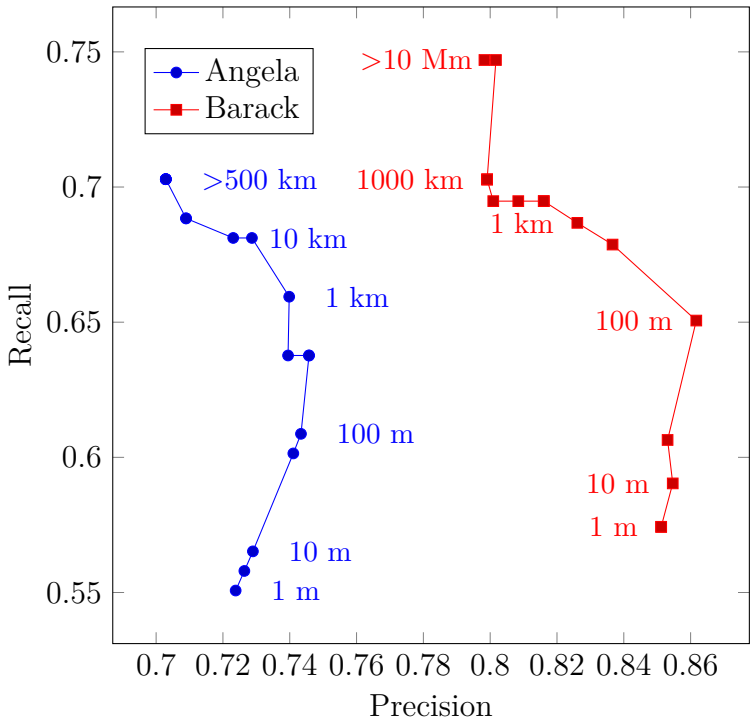


Figure 6.7: Precision-recall curves of matching stays with events for different values of the filtering distance δ and for an overlap duration ratio threshold θ set to 0.2.

Method	M	F	T	P_T	T|A	$P_{T|A}$	F1
Event	26	63.6	4.0	5.4	10.0	13.6	22.9
EventSingle	50	40.8	4.0	7.9	9.6	19.0	27.7
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
GoogleTimeline	0	82.8	14.8	14.8	17.2	17.2	29.4

Table 6.3: Evaluation results for different geocoding techniques on 250 randomly picked event–stay matches in Barack's dataset (in %).

different inputs: (i). from the event's textual location attribute (Event), (ii). from the stay's coordinates (Stay), (iii). from the event's textual location attribute using the geographic proximity to the stay as a bias (StayEvent). For this task, we used Google Places and Maps Geocoding APIs. 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 such as "15 Place Vendôme, Paris" would be qualified as a true address. For comparison, we also evaluated the place given by Barack's own Google Timeline. Due to limitations of the APIs, geocoding from stay coordinates mostly yielded address results (99.2% of the time). To evaluate these methods better, 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 precision $P_{T|A}$, and a recall assimilated to the number of times the geocoder returned a result $(1 - M)$. The evaluation was performed on 250 randomly picked event–stay matches, and the results are presented in Table 6.3. We notice that geocoding using the StayEvent method yields the best precision ($P_{T|A}$ is equal to 67.2%), but only returns a result 50.0% of the time. To cope with that, we consider a fourth method, termed StayEvent|Stay, which returns the result given by StayEvent if it exists and the result given by Stay otherwise. This fourth method always returned a result, and gave the right place 28.4% of the time, and the right place or address 50.0% of the time, which is our best result. This is an OK result considering that around 45% of events' textual locations were room numbers without a mention of either a building or place name (i.e., C101). For comparison, Google Timeline gave the right place or address 17.2% of the time. To showcase the ambiguity present in an events' textual locations, we also evaluated the output of fifth method, EventSingle, that is equivalent to Event, except that it keeps the most relevant result only if the geocoder returns a single result.

```
PREFIX schema: <http://schema.org/>
PREFIX personal: <http://thymeflow.com/personal#>

SELECT ?attendeeTelephone
WHERE {
   ?event a schema:Event ;
      schema:name "Alice's 25th Birthday Party" ;
      schema:attendee/personal:sameAs*
         /schema:telephone
         /schema:name ?attendeeTelephone .
} GROUP BY ?attendeeTelephone
```

Figure 6.8: A query to retrieve the telephone numbers of the attendees of "Alice's 25th Birthday Party" Facebook event.

6.5 Use cases

Finally, we illustrate the uses of a personal KB with some queries. Alice can ask for instance the KB:

- What are the telephone numbers of her birthday party guests? (So she can send them a last-minute message.)
- What are the places that she visited during her previous trip to London? (So she does not go there a second time.)
- What are the latest emails sent by participants of the "Financial Restructuring" working group? (So she quickly reviews them to be ready for this afternoon's meeting.)

Since the KB unites different information sources, queries seamlessly span multiple sources and data types. For instance, the first question in the previous list could ask for the telephone numbers of participants of a Facebook event. However, for multiple different reasons, not all of Alice's friends may have associated their telephone numbers with their Facebook accounts, or Alice may not have access to these numbers. Since she has also connected her personal address book to the KB, she can exploit matches found by the system between her Facebook friends and contacts in her personal address book whose telephone numbers are known. For retrieving such matches, we use the `personal:sameAs` relation. The exact query is shown in Figure 6.8.

The second question can be answered by retrieving all places ever visited by Alice, and filter on an area around London. Alternatively, she can first retrieve the event titled "London Summer 2015" from her calendar, and select the stays whose time span intersects the event's (see Figure 6.9 for the exact query). This way, assuming instead that Alice had been to London several times in the past and that she was trying to recall the location of this great pub she had been to

```
PREFIX schema: <http://schema.org/>
PREFIX personal: <http://thymeflow.com/personal#>

SELECT ?longitude ?latitude WHERE {
   ?event a schema:Event ;
      schema:name "London Summer 2015" ;
      schema:startDate ?eventStartDate ;
      schema:endDate ?eventEndDate .

   ?stay a personal:Stay ;
      schema:geo [
         schema:longitude ?longitude ;
         schema:latitude ?latitude
      ] ;
      schema:startDate ?stayStartDate ;
      schema:endDate ?stayEndDate .

   FILTER( ?stayStartDate <= ?eventEndDate &&
      ?stayEndDate >= ?eventStartDate)
}
```

Figure 6.9: A query to display on a map the places visited during London's 2015 trip.

in the summer 2015, this time filter based on an event's time span would instead allow her to more easily find it.

For the third question, Alice uses the KB's full-text search capabilities to find the events whose name partially matches "Financial Restructuring". Then, she computes the set of the attendees of these meetings and finally outputs the most recent 100 messages sent by any one of them, regardless of the address they used (Figure 6.10).

Analytics Finally, Alice can also perform analytics such as:

- Who does she most frequently communicate with? (Taking into account that some people have several email addresses.)
- What are the places where she usually meets one particular person (based on her calendar)?
- What are the most used email providers used by my contacts?
- Who has sent her the most emails in this particular email folder?

To get an answer to the first question, we start by retrieving the set of all the agents to which Alice sends email messages. Then, it suffices to group by each

```
PREFIX schema: <http://schema.org/>
PREFIX personal: <http://thymeflow.com/personal#>
PREFIX search: <http://www.openrdf.org/contrib/
   lucenesail#>

SELECT ?messageHeadline WHERE {
   ?event a schema:Event ;
      schema:attendee ?attendee ;
      search:matches [
         search:query "Financial Restructuring" ;
         search:property schema:name
      ] .

   ?message a schema:Message ;
      schema:sender/personal:sameAs* ?attendee ;
      schema:headline ?messageHeadline ;
      schema:dateSent ?dateSent .
} ORDER BY DESC(?dateSent) LIMIT 100
```

Figure 6.10: A query to list the most recent 100 messages sent by a participant of the "Financial Restructuring" meetings.

agent equivalence class, and count the number of email messages for each class. To select equivalence classes, we use the special `personal:PrimaryFacet` RDFS class, to which the representative for each equivalence class belongs. The members of this class are computed by a special enricher that chooses a representative for each equivalence class defined by the `personal:sameAs` relation. The query is illustrated in Figure 6.11.

Analytics also provide better visualizations of Alice's personal information. For instance, she could use analytics to extend the contact visualization show in Figure 6.3, by including for each contact a detailed summary of her interactions with this person (e.g., events and places in common, communications). Another way is to show these interactions on a graph visualization. For instance, Figure 6.12 shows a graph of the events and their attendees found in Alice's KB.

User updates The user can use the KB to source synchronization facilities to enrich her personal address book with knowledge inferred by the system. For instance, she can add to each contact in her vCard address book the email addresses found in other facets matched to this contact (see Figure 6.13 for an example query).

```
PREFIX schema: <http://schema.org/>
PREFIX personal: <http://thymeflow.com/personal#>

SELECT (SAMPLE(?emailAddress) AS ?emailAddress)
   (SAMPLE(?name) AS ?name)
   ?primaryAgent
   (SUM(?count) as ?c)
{
   ?primaryAgent a personal:PrimaryFacet .
   ?agent personal:sameAs* ?primaryAgent .

   {
      SELECT ?agent (COUNT(?email) as ?count) WHERE
         {
         ?email schema:sender ?userFacet ;
            schema:recipient ?agent .
         {
            SELECT DISTINCT ?userFacet {
               ?userFacet personal:sameAs*/schema:
                  email <mailto:alice@thymeflow.com>
                  .
            }
         }
      } GROUP BY ?agent
   }

   OPTIONAL {
      ?primaryAgent schema:email ?emailAddress ;
         schema:name ?name .
   }
} GROUP BY ?primaryAgent
ORDER BY DESC(?count)
```

Figure 6.11: A query to list the contacts to which Alice sends the most email messages. For each contact, one email address and/or name is retrieved. The query has been optimized. It takes into account in particular that the `schema:email` relation may appear in multiple named graphs.

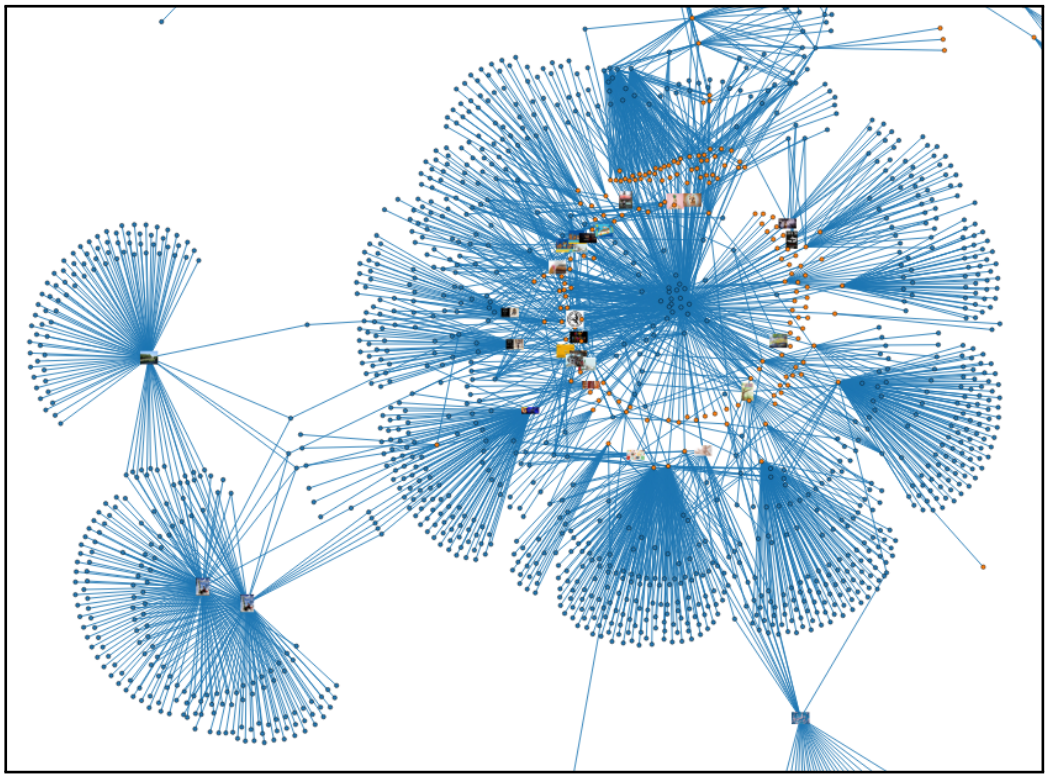


Figure 6.12: A graph visualization of the events and their attendees in Alice's knowledge base. A blue node represents an agent, while an image or an orange node represents an event. Agent nodes are connected to events via `schema:attendee` edges. Alice can interact with a node to highlight its neighbors or show its attributes (e.g., event's name, agent's telephone number).

6.6 Related work

The work presented in this chapter was motivated by the general concept of personal information management, taking the viewpoint of Abiteboul, André, and Kaplan (2015) as to what a PIMS should be. In Section 1.4, we gave some notable examples of the current state of the art in PIMS. In this section, we highlight the works to which ours is the most related.

Personal knowledge bases The problem of building a knowledge base for querying and managing personal information is not new. Among the first projects in this direction were IRIS (Cheyer, Park, and Giuli 2005), SEMEX (Dong and A. Y. Halevy 2005), and NEPOMUK (Groza et al. 2007). These used Semantic Web technologies to exchange data between different applications within a single desktop computer. Some of them also provided semantic search facilities for desktop information. However, these projects date from 2007 and before, and much has changed since then. Today, most of our personal information is not stored on an individual computer, but spread across several devices (Abiteboul, André, and Kaplan 2015).

Our work is different from these projects in three aspects. (i) We do not

```
PREFIX schema: <http://schema.org/>
PREFIX personal: <http://thymeflow.com/personal#>
INSERT {
   GRAPH ?provenance {
      ?agent schema:email ?email
   }
} WHERE {
   GRAPH ?provenance {
      ?agent a personal:Agent .
   }

   ?provenance a personal:ContactsSource ;
      personal:account/schema:name
         "alice.springs@gmail.com" .

   ?agent personal:sameAs+/schema:email ?email .
   FILTER NOT EXISTS {
      ?agent schema:email ?email .
   }
}
```

Figure 6.13: A query that adds to each contact in Alice's Google account the email addresses found on matched agents.

tackle personal information management by reinventing the user experience for reading/writing emails, managing a calendar, organizing files, etc. This is the case for IRIS and NEPOMUK. (ii) We embrace personal information as being fundamentally distributed and heterogeneous and we focus on the need of providing knowledge integration on top for creating completely new services (complex query answering, analytics). SEMEX is the only one to provide tight integration, and we will discuss how our system differs from SEMEX in the next paragraph. (iii) While NEPOMUK provides text analysis tools for extracting entities from rich text and linking them with elements of the KB, our first focus is on enriching existing semi-structured data, which improves the quality of available data for use by other services.

Information Integration Entity resolution and information integration in general is not a new problem (cf. Section 2.3). In the context of personal information, SEMEX (Dong and A. Y. Halevy 2005) also integrates entity resolution facilities. It 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, they do this integration at import time so Alice cannot later manually revoke it through an update. Also, they do not handle incremental synchronization. Recently,

contact managers from commercial vendors have started providing de-duplication tools for finding duplicate contacts and merging them in bulk. However, these tools restrict themselves to contacts present in Alice's address book and do not necessarily merge contacts from social networks or emails.

Combining the location history with the calendar Improvements in accuracy and battery efficiency of mobile location technologies have made possible the estimation of Alice's activities and visited places on a daily basis (cf. Section 2.3). Most of these works have mainly exploited sensor data (acceleration, radio communications, and location) and readily available geographic data. Few of them, however, have exploited Alice'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 Alice's calendar: e.g., for distinguishing genuine real-world events from reminders (Lovett et al. 2010).

Ontology 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 and iCalendar content into RDF (Iannella and McKinney 2014; Miller and Connolly 2005). While such standards are useful in our context, they do not provide the means to match calendars, email messages, and events, as we do. The only set of vocabularies besides `schema.org` that provides a broad coverage of all entities we are dealing with is the OSCAF ontologies (Nepomuk Consortium and OSCAF 2007). 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 (R. V. Guha, Brickley, and Macbeth 2016). Recently, a personal data service has been proposed that reuses the OSCAF ontologies, but they use a relational database instead of a knowledge base (Sjöberg et al. 2016).

Email Analysis Several commercial applications and academics have taken to analyzing the content of email. For example, they can recognize simple patterns such as salutations, signatures, and dates (Carvalho and Cohen 2004). Apple Mail and Gmail can find dates and suggest updates to the user's calendar. Apple Mail can also identify contact information and suggest an update to the user's address book (Apple 2016). Semantic data attached to email content in the form of JSON-LD/Microdata annotations can provide information about flight, hotel, train, bus, and restaurant reservations (Google 2016). Other services, such as TripIt or Wipolo, can parse the content of emails to extract travel data and construct a trip schedule. All of these approaches are orthogonal to our work: they can provide enrichment modules in our system. Our work as a whole aims to construct a coherent knowledge base on top of different sources of personal information.

Commercial Solutions Some commercial providers, such as Google and Apple ecosystems, 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 pro-actively interact with Alice. However, these are closed-source and promote vendor lock-in.

6.7 Conclusion

In this chapter, we presented and evaluated Thymeflow, a system that allows the user to build her own personal knowledge base. Our system integrates information from her email messages, calendars, address books, social network services, and location history. It can identify agent entities and merge its different facets across the different information sources, determine long stays in the location history, and align them with calendars events. The user can use the system to visualize the aggregated information that is associated with an agent, her timeline, or perform more complex or analytical queries over her personal information. The system is meant to remain under the direct control of the user and fully respect the privacy of the user's data. It is available under an open-source software license[3], so that people can freely use it, and researchers and developers can build on it.

Contributions We provide a fully functional and open-source personal knowledge management system. In particular, we provide the following contributions:

1. The management of location histories in a personal KB and their integration with other types of personal information. As discussed previously, location histories are increasingly being collected by various kinds of mobile applications (cf. Section 3.5). While a location history is a great source of knowledge by itself (Chapter 4, Chapter 5), we believe it only becomes truly useful if it is semantically enriched with events and persons in the user's personal space – which is what we do.

2. The adaptation of ontology alignment techniques to the context of personal KBs. The alignment of persons and organizations is rather standard. More novel are the alignments based on text (a contact and a person mentioned in a calendar entry), on time (a calendar event and a location), or on space (the address of a contact and a place).

3. An architecture that allows the integration of heterogeneous personal information sources into a coherent whole. This includes a framework for implementing knowledge enrichment algorithms and the design of incremental synchronization, where a change in a source triggers the loading and treatment of just these changes in the central KB. Inversely, the user is able to perform updates on the KB, which are persisted wherever possible in the sources.

[3] `https://github.com/thymeflow`

Future directions Our system can be extended in a number of directions, including

- Incorporating more information: In particular, we want to include the user's web search history and her bank statements, as well as other services, e.g., TripAdvisor and Spotify.
- Extracting semantics from text: So far, we simply index text. We could, for instance, find in the body of an email message the date and location of a meeting that has been planned and perhaps the name of its participants (listed or not as recipients of the message) (Carvalho and Cohen 2004; Google 2016).
- Inferring more advanced knowledge: For instance, we could try to infer the kind of affiliation or relationship (friend, colleague, or family) that the user has with each of her acquaintances from the information that the system has about them (communications, events, contact information, etc.).
- Exploiting this knowledge: As illustrated in Section 6.5, our system can be used for querying. One could consider using it for more advanced personal analytics and prediction.
- Improving the user interface: We would like to allow for a simpler query language, maybe even natural language, to open our tool to non-technical users. At the moment, our most user-friendly visualizations are the timeline and contact components.

Conclusion

We have presented a novel system for personal information management. The system builds a personal knowledge base from multiple heterogeneous sources of personal information: email, calendars, address books, social network services, and location history. Throughout our doctoral research, we designed, implemented, and evaluated different algorithms for inferring knowledge from this data: stay point extraction, transportation mode and route recognition, entity resolution, spatiotemporal event-location alignment. With these algorithms, and with an architecture that keeps the knowledge base continuously synchronized with the sources, we showed how the user can use the knowledge base to answer new questions.

Although the different enrichment algorithms could be further improved, they provide a flavor of some of the problems that need to be solved in order to build a coherent experience for the user, using readily available information.

Our work could be extended in different ways:

- Stay point extraction and transportation mode and route recognition should perhaps work as a single coherent task. Rather than first identifying the places that the user visited then deriving the transportation modes that she used to go from one place to another, it may be better to perform a joint derivation of a sequence of heterogeneous user "activities", e.g., "having dinner at a restaurant", "taking the bus home", and "walking to work".

- The alignment procedure between events and stay points (or ideally "activities") could be expanded to take event attributes other than the event's duration and location. In addition to the event's description, its list of attendees, its context (i.e., the previous and next events), related email messages, as well as the history of previously confirmed *event to stay point* alignments could be used to improve the alignment.

- Thymeflow could be expanded to incorporate more information, to infer more knowledge. We think however that the focus should also be put in stabilizing and improving its core features. For instance, to handle large amounts of data, strategies should be designed to infer knowledge without necessarily keeping a copy of the base data (e.g., pictures). Also, new strategies should be designed for pushing knowledge to the sources. For instance, we need to improve our understanding on how the pushing of reconciled knowledge back to a source may cause a loss of information in the source that could be useful in the future.

We would also like to raise the following issues which could drive future research and development:

Gathering data for experiments Throughout our research, a considerable amount of effort was spent collecting, organizing, and annotating personal information from willing participants, but also geographic and public transportation information from online sources. In practice, it is easy to collect much more than what may be needed, more than we can use to evaluate our algorithms. There is a balance to be found between putting effort into annotating and organizing already available data and putting effort into collecting more complex or cleaner data. The research community might benefit from building and maintaining sets of annotated and multi-dimensional personal information for use in solving different kinds of tasks and evaluating the solutions.

Opportunities Many Internet companies that already hold a lot of user data are not yet integrating everything they have about each user into 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 personal information alone.

Inaccessible information The hard truth is that many popular Internet-based services still do not provide APIs for conveniently retrieving user data out of them (e.g., WhatsApp), or provide APIs that do not allow the retrieval of all user data (e.g., Facebook). For this reason, solutions for automatically extracting user data and other relevant personal information no matter the available interface could definitely be useful. While Web crawlers are useful for obtaining information from online Web services, a more suitable alternative for obtaining relevant personal information may be to systematically intercept the Web content that the user sees when using these services, in the style of *lifelogging* but where the focus is on extracting relevant structured data and where the relevance of an item could be inferred from the user interactions (e.g., mouse clicks, text input) and preexisting knowledge. Admittedly, such an alternative generalizes to services without a Web interface assuming that a content interceptor can exist between the service's interface and the user (e.g., a screenshot of a mobile application).

In 2016, Gartner introduced *personal analytics* into its "hype cycle" for emerging technologies (Gartner 2016). This represents a strong belief by the firm that this fairly young technology shows promise to deliver a high degree of competitive advantage over the next years. We believe so too, and think that with the increasing diversification and popularity of quantified-self wearables and the development of context-aware applications such as intelligent personal assistants, the sources needed and the need to derive the emotions, activities, schedules, well-being, and mobility patterns of an individual will keep augmenting. Recognizing habits, tastes, patterns, and motives in different aspects of a individual's life such

as work time, leisure, health, personal finance, and consumption has long been in the agenda of marketers, advertisers, credit bureaus, and health corporations.

However, this need is still greatly driven by corporations that are looking to create value from this data, especially when the understanding comes from aggregating data from a large group of individuals. Individuals themselves have not yet embraced this power, as they are still struggling to take control over their personal information, concerned by their own privacy yet still making use of many services to which they give a lot of information away, for the convenience they provide.

While our vision of a PIMS does seem to be a solid foundation for personal analytics, the difficulty is finding the precise leverage to attract and deploy it to consumers. New functionalities are indeed attractive, but are only a necessary condition. An important issue is the business model that, in our opinion, should stay away from standard models for Web services based on monetizing the user's attention or data. Another issue is which companies would be involved in the ecosystem of such a product. Such a PIMS should interest traditional companies that feel increasingly disintermediated by pure Internet players and startups that want to quickly develop novel products that could leverage cleaner, richer, and/or synthesized information from the user. These issues may suggest expansions of the Thymeflow model that we intend to investigate in the future.

Self-references

Montoya, D. and S. Abiteboul (2014). "Inférence d'itinéraires multimodaux à partir de données smartphone". In: *Gestion de Données — Principes, Technologies et Applications*. Actes de la 30ème Conférence. Ed. by D. Gross-Amblard, C. Collet, C. Bobineau, and F. Jouanot. BDA '14. Autrans, France, pp. 38–42. URL: https://montoya.one/fr/publication/hupme-bda-2014/.

Montoya, D., S. Abiteboul, and P. Senellart (2015). "Hup-Me: Inferring and Reconciling a Timeline of User Activity with Smartphone and Personal Data". In: *Advances in Geographic Information Systems*. Proceedings of the 23rd SIGSPATIAL International Conference. Ed. by M. Ali, H. Yan, M. Gertz, M. Renz, and J. Sankaranarayanan. SIGSPATIAL '15. Seattle, Washington, USA, 62:1–4. DOI: 10.1145/2820783.2820852.

Montoya, D., T. Pellissier Tanon, S. Abiteboul, and F. Suchanek (2016). "Thymeflow, A Personal Knowledge Base with Spatio-temporal Data". In: *Information and Knowledge Management*. Proceedings of the 25th ACM International Conference. CIKM '16. Indianapolis, Indiana, USA: ACM, pp. 2477–2480. DOI: 10.1145/2983323.2983337.

Other references

93rd United States Congress (1974). *Privacy Act of 1974*. Public Law 93–579. URL: http://legislink.org/us/pl-93-579.

Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng (2016). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. arXiv: 1603.04467 [abs].

Abiteboul, S., B. André, and D. Kaplan (2015). "Managing Your Digital Life". In: *Communications of the ACM* 58.5, pp. 32–35. DOI: 10.1145/2670528.

Abiteboul, S., I. Manolescu, P. Rigaux, M.-C. Rousset, and P. Senellart (2011). *Web Data Management*. Cambridge University Press.

Adams, B., D. Phung, and S. Venkatesh (2006). "Extraction of Social Context and Application to Personal Multimedia Exploration". In: *Multimedia*. Proceedings of the 14th ACM International Conference. MM '06. Santa Barbara, CA, USA: ACM, pp. 987–996. DOI: 10.1145/1180639.1180857.

Andor, D., C. Alberti, D. Weiss, A. Severyn, A. Presta, K. Ganchev, S. Petrov, and M. Collins (2016). "Globally Normalized Transition-Based Neural Networks". In: *Association for Computational Linguistics*. Proceedings of the 54th Annual Meeting. ACL '16. Berlin, Germany. URL: http://aclweb.org/anthology/P/P16/P16-1231.pdf.

Android Developers Guide (2013). *Recognizing the User's Current Activity*. URL: http://developer.android.com/training/location/activity-recognition.html (visited on 05/16/2014).

Apple (2016). *Mail: Add events, contacts, and more from messages*. URL: https://support.apple.com/kb/PH22304 (visited on 04/18/2016).

Apple Inc. (2017a). *CLLocationManager – Core Location*. URL: https://developer.apple.com/reference/corelocation/cllocationmanager (visited on 01/30/2017).

Apple Inc (2013). *CMMotionActivity*. URL: https://developer.apple.com/library/ios/documentation/CoreMotion/Reference/CMMotionActivity_class/ (visited on 05/16/2014).

Apple Inc. (2017b). *iOS 10 for Developers – Apple Developer*. URL: https://developer.apple.com/ios/ (visited on 01/30/2017).

Argote, L. and P. Ingram (2000). "Knowledge transfer: A basis for competitive advantage in firms". In: *Organizational behavior and human decision processes* 82.1, pp. 150–169. DOI: 10.1006/obhd.2000.2893.

A.R.O., Inc. (2011). *SAGA: Choose your own adventure*. URL: http://www.getsaga.com/ (visited on 01/05/2017).

Ashbrook, D. and T. Starner (2003a). "Using GPS to learn significant locations and predict movement across multiple users". In: *Personal and Ubiquitous Computing* 7.5, pp. 275–286. DOI: 10.1007/s00779-003-0240-0.

— (2003b). "Using GPS to learn significant locations and predict movement across multiple users". English. In: *Personal and Ubiquitous Computing* 7.5, pp. 275–286. DOI: 10.1007/s00779-003-0240-0.

Atkinson, R. C. and R. M. Shiffrin (1968). "Human Memory: A Proposed System and its Control Processes". In: *Psychology of Learning and Motivation* 2. Ed. by K. W. Spence and J. T. Spence, pp. 89–195. DOI: 10.1016/S0079-7421(08)60422-3.

Bakir, G. H., T. Hofmann, B. Schölkopf, A. J. Smola, B. Taskar, and S. V. N. Vishwanathan (2007). *Predicting Structured Dataa*. MIT press.

Bao, L. and S. S. Intille (2004). "Activity Recognition from User-Annotated Acceleration Data". In: *Pervasive Computing*. Proceedings of the Second International Conference, PERVASIVE 2004. Ed. by A. Ferscha and F. Mattern. Vol. 3001. Lecture Notes In Computer Science. Berlin, Heidelberg: Springer, pp. 1–17. DOI: 10.1007/978-3-540-24646-6_1.

Barnes, S. B. (2006). "A privacy paradox: Social networking in the United States". In: *First Monday*. DOI: 10.5210/fm.v11i9.1394.

Barthélemy, M. (2011). "Spatial networks". In: *Physics Reports* 499.1–3, pp. 1–101. DOI: 10.1016/j.physrep.2010.11.002.

Bell, G. and J. Gemmell (2009). *Total Recall. How the E-memory Revolution Will Change Everything*. Dutton.

Berners-Lee, T. (2006). *Linked Data – Design issues*. URL: http://www.w3.org/DesignIssues/LinkedData.html (visited on 01/14/2017).

Berners-Lee, T., J. Hendler, and O. Lassila (2001). "The Semantic Web". In: *Scientific American*. URL: https://www.scientificamerican.com/article/the-semantic-web/.

Bernstein, M., M. Van Kleek, D. Karger, and M. C. Schraefel (2008). "Information Scraps: How and Why Information Eludes Our Personal Information Management Tools". In: *ACM Transactions on Information Systems* 26.4, 24:1–24:46. DOI: 10.1145/1402256.1402263.

Biagioni, J., T. Gerlich, T. Merrifield, and J. Eriksson (2011). "EasyTracker: Automatic Transit Tracking, Mapping, and Arrival Time Prediction Using Smartphones". In: *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*. SenSys '11. (Seattle, Washington). New York, NY, USA: ACM, pp. 68–81. DOI: 10.1145/2070942.2070950.

Biron, P. V. and A. Malhotra (2004). *XML Schema Part 2: Datatypes Second Edition*. W3C Recommendation. W3C. URL: https://www.w3.org/TR/xmlschema-2/.

Bizer, C., T. Heath, and T. Berners-Lee (2009). "Linked data — the story so far". In: *International Journal on Semantic Web and Information Systems* 5.3, pp. 1–22. DOI: 10.4018/jswis.2009081901.

Bizer, C., R. Meusel, and A. Primpeli (2016). *Web Data Commons —- RDFa, Microdata, and Microformat Data Set*. URL: http://webdatacommons.org/structureddata/ (visited on 12/28/2016).

Blunschi, L., J. Dittrich, O. R. Girard, S. K. Karakashian, and M. A. V. Salles (2007). "A Dataspace Odyssey: The iMeMex Personal Dataspace Management System". In: *Innovative Data Systems Research*. Third Biennial Conference. CIDR 2007. Asilomar, California, USA, pp. 114–119. URL: http://cidrdb.org/cidr2007/papers/cidr07p13.pdf.

Bohn, R. E. and J. E. Short (2009). *How Much Information? 2009 Report on American Consumers*. Tech. rep. Global Information Industry Center, University of California, San Diego.

Bolger, N., A. Davis, and E. Rafaeli (2003). "Diary methods: Capturing life as it is lived". In: *Annual Review of Psychology* 54, pp. 579–616. DOI: 10.1146/annurev.psych.54.101601.145030.

Bostock, M. (2017). *D3.js – Data-Driven Documents*. URL: https://d3js.org/ (visited on 01/30/2017).

Brickley, D. and R. Guha (2014). *RDF Schema 1.1*. W3C Recommendation. W3C. URL: http://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

Brickley, D. and L. Miller (2014). *FOAF Vocabulary Specification 0.99*. URL: http://xmlns.com/foaf/spec/20140114.html (visited on 12/28/2016).

Broekstra, J., A. Kampman, and F. van Harmelen (2002). "Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema". In: *The Semantic Web — ISWC 2002*. Proceedings of the First International Semantic Web Conference. Ed. by I. Horrocks and J. Hendler. Vol. 2342. Lecture

Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 54–68. DOI: 10.1007/3-540-48005-6_7.

Buckle, C. (2016). *Digital consumers own 3.64 connected devices*. Global Web Index. URL: https://www.globalwebindex.net/blog/digital-consumers-own-3.64-connected-devices (visited on 01/12/2017).

Bui, H. H., S. Venkatesh, and G. West (2002). "Policy Recognition in the Abstract Hidden Markov Model". In: *Journal of Artificial Intelligence Research* 17, pp. 451–499. DOI: 10.1613/jair.839.

Bush, V. (1945). "As We May Think". In: *The Atlantic Monthly* 176.1, pp. 101–108. URL: https://www.theatlantic.com/magazine/archive/1945/07/as-we-may-think/303881/.

Byrd, A. and L. Grégoire (2016). *OpenTripPlanner v1.0.0*. URL: http://opentripplanner.org (visited on 01/10/2017).

Carothers, G. and A. Seaborne (2014). *RDF 1.1 TriG*. W3C Recommendation. W3C. URL: http://www.w3.org/TR/2014/REC-trig-20140225/.

Carroll, J. and G. Klyne (2004). *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation. W3C. URL: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

Carvalho, V. R. de and W. W. Cohen (2004). "Learning to Extract Signature and Reply Lines from Email". In: *Email and Anti-Spam*. Proceedings of the First Conference. CEAS 2004. Mountain View, California, USA. URL: http://www.ceas.cc/papers-2004/135.pdf.

Chen, J. and M. Bierlaire (2015). "Probabilistic Multimodal Map Matching With Rich Smartphone Data". In: *Journal of Intelligent Transportation Systems: Technology, Planning and Operations* 19.2, pp. 134–148. DOI: 10.1080/15472450.2013.764796.

Chen, L., C. D. Nugent, J. Biswas, and J. Hoey (2011). *Activity Recognition in Pervasive Intelligent Environments*. Vol. 4. Atlantis Ambient and Pervasive Intelligence. Atlantis Press. DOI: 10.2991/978-94-91216-05-3.

Cheyer, A., J. Park, and R. Giuli (2005). *IRIS: Integrate. Relate. Infer. Share*. Tech. rep. Aachen, Germany, pp. 64–78. URL: http://ceur-ws.org/Vol-175/17_park_iris_final.pdf.

Choudhury, T., G. Borriello, S. Consolvo, D. Haehnel, B. Harrison, B. Hemingway, J. Hightower, P. " Klasnja, K. Koscher, A. LaMarca, J. A. Landay, L. LeGrand, J. Lester, A. Rahimi, A. Rea, and D. Wyatt (2008). "The Mobile Sensing Platform: An Embedded Activity Recognition System". In: *IEEE Pervasive Computing* 7.2, pp. 32–41. DOI: 10.1109/MPRV.2008.39.

Christen, P. (2012). *Data matching. Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer. DOI: 10.1007/978-3-642-31164-2.

Constine, J. (2015). *Facebook Is Shutting Down Its API For Giving Your Friends' Data To Apps*. URL: https://techcrunch.com/2015/04/28/facebook-api-shut-down/ (visited on 12/28/2016).

Cooper, B. B. and J. Sharp (2017). *Exist – Understand your behaviour.* URL: https://exist.io/ (visited on 01/30/2017).

Cowie, J. and W. Lehnert (1996). "Information Extraction". In: *Communications of the ACM* 39.1, pp. 80–91. DOI: 10.1145/234173.234209.

Cozy Cloud (2016). *Cozy — Simple, versatile, yours.* URL: https://cozy.io/ (visited on 01/15/2017).

Crispin, M. (2003). *Internet Message Access protocol - version 4rev1*. RFC 3501. IETF. URL: https://tools.ietf.org/html/rfc3501.

Daboo, C. (2011). *CardDAV: vCard Extensions to Web Distributed Authoring and Versioning (WebDAV)*. RFC 6352. IETF. URL: https://tools.ietf.org/html/rfc6352.

Daboo, C., B. Desruisseaux, and L. Dusseault (2007). *Calendaring Extensions to WebDAV (CalDAV)*. RFC 4791. IETF. URL: https://tools.ietf.org/html/rfc4791.

Daboo, C. (2009). *iCalendar Transport-Independent Interoperability Protocol (iTIP)*. RFC 5546. IETF. URL: https://tools.ietf.org/html/rfc5546.

Daigle, L. (2004). *WHOIS Protocol Specification*. RFC 3912. IETF. URL: https://tools.ietf.org/html/rfc3912.

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

Dienlin, T. and S. Trepte (2015). "Is the privacy paradox a relic of the past? An in-depth analysis of privacy attitudes and privacy behaviors". In: *European Journal of Social Psychology* 45.3, pp. 285–297. DOI: 10.1002/ejsp.2049.

Dittrich, J. and M. A. V. Salles (2006). "iDM: A Unified and Versatile Data Model for Personal Dataspace Management". In: *Very Large Data Bases*. Proceedings of the 32nd International Conference. VLDB '06. Seoul, Korea: VLDB Endowment, pp. 367–378. URL: http://dl.acm.org/citation.cfm?id=1182635.1164160.

Doherty, A. R., N. Caprani, C. Ó. Conaire, V. Kalnikaite, C. Gurrin, A. F. Smeaton, and N. E. O'Connor (2011). "Passively recognising human activities through lifelogging". In: *Computers in Human Behavior* 27.5, pp. 1948–1958. DOI: 10.1016/j.chb.2011.05.002.

Doherty, A. R., C. J. A. Moulin, and A. F. Smeaton (2011). "Automatically assisting human memory: A SenseCam browser". In: *Memory* 19.7, pp. 785–795. DOI: 10.1080/09658211.2010.509732.

Dong, X. and A. Y. Halevy (2005). "A Platform for Personal Information Management and Integration". In: *Innovative Data Systems Research*. Second Biennial Conference. CIDR 2005. Asilomar, CA, USA, pp. 119–130. URL: http://cidrdb.org/cidr2005/papers/P10.pdf.

Drago, I., M. Mellia, M. M. Munafo, A. Sperotto, R. Sadre, and A. Pras (2012). "Inside Dropbox: Understanding Personal Cloud Storage Services". In: *Internet Measurement Conference*. Proceedings of the 2012. IMC '12. Boston, Massachusetts, USA: ACM, pp. 481–494. DOI: 10.1145/2398776.2398827.

Dumais, S. T., E. Cutrell, J. J. Cadiz, G. Jancke, R. Sarin, and D. C. Robbins (2003). "Stuff I've Seen: A System for Personal Information Retrieval and Re-use". In: *Research and Development in Information Retrieval*. Proceedings of the 26th Annual International ACM SIGIR Conference. SIGIR '03. Toronto, Canada: ACM, pp. 72–79. DOI: 10.1145/860435.860451.

Dusseault, L. (2007). *HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)*. RFC 4918. IETF. URL: https://tools.ietf.org/html/rfc4918.

Eclipse Foundation (2017). *Eclipse RDF4J – a Java framework for RDF*. URL: http://rdf4j.org/ (visited on 01/30/2017).

École Polytechnique Fédérale de Lausanne (2017). *The Scala Programming Language*. URL: https://www.scala-lang.org/ (visited on 01/30/2017).

European Parliament and Council (2016). "Article 17. Right to erasure ('right to be forgotten')". In: *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance)*, pp. 43–44. URL: http://data.europa.eu/eli/reg/2016/679/oj.

— (1995). "Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data". In: *Official Journal of the European Communities* L 281, pp. 31–50. URL: http://data.europa.eu/eli/dir/1995/46/oj.

Euzenat, J. and P. Shvaiko (2013). *Ontology Matching*. 2nd ed. Berlin, Heidelberg: Springer. DOI: 10.1007/978-3-642-38721-0.

Facebook (2017). *Changelog – Graph API*. URL: https://developers.facebook.com/docs/apps/changelog (visited on 01/30/2017).

— (2016a). *Company Info — Stats*. URL: https://web.archive.org/web/20161228131003/http://newsroom.fb.com/company-info/ (visited on 12/28/2016).

— (2016[b]). *The Graph API*. URL: https://developers.facebook.com/docs/graph-api/ (visited on 09/01/2016).

Fisher, D., A. J. Brush, E. Gleave, and M. A. Smith (2006). "Revisiting Whittaker & Sidner's "Email Overload" Ten Years Later". In: *Computer Supported Cooperative Work*. Proceedings of the 2006 20th Anniversary Conference. CSCW '06. Banff, Alberta, Canada: ACM, pp. 309–312. DOI: 10.1145/1180875.1180922.

Flint, L. N. (1917). *Newspaper writing in high schools. Containing an outline for the use of teachers*. Lloyd Adams Noble. URL: https://archive.org/details/newspaperwriting00flinrich.

Franklin, M., A. Halevy, and D. Maier (2005). "From Databases to Dataspaces: A New Abstraction for Information Management". In: *SIGMOD Record* 34.4, pp. 27–33. DOI: 10.1145/1107499.1107502.

Fraser, N. (2009). "Differential Synchronization". In: *Document Engineering*. Proceedings of the 9th ACM Symposium. DocEng '09. Munich, Germany: ACM, pp. 13–20. DOI: 10.1145/1600193.1600198.

Freeman, E. T. (1997). "The Lifestreams Software Architecture". PhD thesis. Yale University.

Freeman, E. and D. Gelernter (1996). "Lifestreams: A Storage Model for Personal Data". In: *SIGMOD Record* 25.1, pp. 80–86. DOI: 10.1145/381854.381893.

Garcia-Molina, H., Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, and J. Widom (1997). "The TSIMMIS Approach to Mediation: Data Models and Languages". In: *Journal of Intelligent Information Systems* 8.2, pp. 117–132. DOI: 10.1023/A:1008683107812.

Gartner (2015). *Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015*. URL: http://www.gartner.com/newsroom/id/3165317 (visited on 12/28/2016).

— (2016). *Gartner's 2016 Hype Cycle for Emerging Technologies Identifies Three Key Trends That Organizations Must Track to Gain Competitive Advantage*. URL: http://www.gartner.com/newsroom/id/3412017 (visited on 01/18/2017).

Gellman, B. and L. Poitras (2013). "U.S., British intelligence mining data from nine US Internet companies in broad secret program". In: *The Washington Post*.

Gemalto (2016). *It's All About Identity Theft. First half findings from the 2016 Breach Level Index*. URL: http://breachlevelindex.com/assets/Breach-Level-Index-Report-H12016.pdf (visited on 01/13/2017).

Gemmell, J., G. Bell, and R. Lueder (2006). "MyLifeBits: A Personal Database for Everything". In: *Communications of the ACM* 49.1, pp. 88–95. DOI: 10.1145/1107458.1107460.

Gemmell, J., G. Bell, R. Lueder, S. Drucker, and C. Wong (2002). "MyLifeBits: Fulfilling the Memex Vision". In: *Multimedia*. Proceedings of the Tenth ACM International Conference. MULTIMEDIA '02. Juan-les-Pins, France: ACM, pp. 235–238. DOI: 10.1145/641007.641053.

GeoTelematic Solutions, Inc. (2017). *GPS Tracking: Open-Source GPS Tracking System – OpenGTS*. URL: http://www.opengts.org/ (visited on 01/30/2017).

Goodfellow, I., Y. Bengio, and A. Courville (2016a). "Deep Feedforward Networks". In: *Deep Learning*. MIT Press. Chap. 6, pp. 168–227. URL: http://www.deeplearningbook.org/contents/mlp.html.

— (2016b). "Sequence Modeling: Recurrentand Recursive Nets". In: *Deep Learning*. MIT Press. Chap. 10, pp. 373–420. URL: http://www.deeplearningbook.org/contents/rnn.html.

Google (2017a). *Android*. URL: https://www.android.com/ (visited on 01/30/2017).

— (2016). *Email Markup*. URL: https://developers.google.com/gmail/markup/ (visited on 04/18/2016).

— (2015a). *General Transit Feed Specification Reference*. URL: https://developers.google.com/transit/gtfs/reference (visited on 03/23/2015).

— (2017b). *Google Maps APIs*. URL: https://developers.google.com/maps/documentation (visited on 01/30/2017).

— (2015b). *Google Timeline and Location History*. URL: https://www.google.fr/maps/timeline (visited on 09/11/2016).

— (2017c). *LocationRequest – Google APIs for Android*. URL: https://developers.google.com/android/reference/com/google/android/gms/location/LocationRequest (visited on 01/30/2017).

Graves, A., M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber (2009). "A Novel Connectionist System for Unconstrained Handwriting Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31.5, pp. 855–868. DOI: 10.1109/TPAMI.2008.137.

Graves, A., A.-r. Mohamed, and G. Hinton (2013). "Speech recognition with deep recurrent neural networks". In: *Acoustics, Speech and Signal Processing*. Proceedings of the 2013 IEEE International Conference. IEEE, pp. 6645–6649. DOI: 10.1109/ICASSP.2013.6638947.

Greenwald, G. (2013). "NSA collecting phone records of millions of Verizon customers daily". In: *The Guardian*. URL: http://www.guardian.co.uk/world/2013/jun/06/nsa-phone-records-verizon-court-order.

Grevet, C., D. Choi, D. Kumar, and E. Gilbert (2014). "Overload is Overloaded: Email in the Age of Gmail". In: *Human Factors in Computing Systems*. Proceedings of the SIGCHI Conference. CHI '14. Toronto, Ontario, Canada: ACM, pp. 793–802. DOI: 10.1145/2556288.2557013.

Groza, T., S. Handschuh, K. Möller, G. Grimnes, L. Sauermann, E. Minack, C. Mesnage, M. Jazayeri, G. Reif, and R. Gudjónsdóttir (2007). "The NEPOMUK Project — On the way to the Social Semantic Desktop". In: *Proceedings of I-SEMANTICS 2007*. (Graz, Austria). Ed. by T. Pellegrini and S. Schaffert, pp. 201–211. DOI: 10379/437.

Guha, R. V., D. Brickley, and S. Macbeth (2016). "Schema.Org: Evolution of Structured Data on the Web". In: *Commununications of the ACM* 59.2, pp. 44–51. DOI: 10.1145/2844544.

Guha, R. and D. Brickley (2004). *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation. W3C. URL: http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

Gurrin, C., A. F. Smeaton, and A. R. Doherty (2014). "LifeLogging: Personal Big Data". In: *Foundations and Trends in Information Retrieval* 8.1, pp. 1–125. DOI: 10.1561/1500000033.

Haklay, M. and P. Weber (2008). "OpenStreetMap: User-Generated Street Maps". In: *IEEE Pervasive Computing* 7.4, pp. 12–18. DOI: 10.1109/MPRV.2008.80.

Hansmann, U., R. Mettälä, A. Purakayastha, and P. Thompson (2002). *SyncML. Synchronizing and Managing Your Mobile Data*. Prentice Hall.

Hariharan, R. and K. Toyama (2004). "Project Lachesis: Parsing and Modeling Location Histories". In: *Geographic Information Science*. Proceedings of the Third International Conference, GIScience 2004. (Adelphi, MD, USA). Ed. by M. J. Egenhofer, C. Freksa, and H. J. Miller. Vol. 3234. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 106–124. DOI: 10.1007/978-3-540-30231-5_8.

Harris, S. and A. Seaborne (2013). *SPARQL 1.1 Query Language*. W3C Recommendation. W3C. URL: http://www.w3.org/TR/2013/REC-sparql11-query-20130321/.

HAT Data Exchange Ltd. (2017). *Hub of All Things*. URL: https://hubofallthings.com/ (visited on 01/30/2017).

Hellinger, E. (1909). "Neue Begründung der Theorie quadratischer Formen von unendlichvielen Veränderlichen". In: *Journal für die reine und angewandte Mathematik* 136, pp. 210–271.

Hemminki, S., P. Nurmi, and S. Tarkoma (2013). "Accelerometer-based Transportation Mode Detection on Smartphones". In: *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*. SenSys '13. (Roma, Italy). New York, NY, USA: ACM, 13:1–13:14. DOI: 10.1145/2517351.2517367.

Hightower, J., S. Consolvo, A. LaMarca, I. Smith, and J. Hughes (2005). "Learning and Recognizing the Places We Go". In: *UbiComp 2005: Ubiquitous Computing*. Proceedings of the 7th International Conference, UbiComp 2005. (Tokyo, Japan). Vol. 3660. Lecture Notes in Computer Science. Springer, pp. 159–176. DOI: 10.1007/11551201_10.

Huang, Z., W. Xu, and K. Yu (2015). *Bidirectional LSTM-CRF Models for Sequence Tagging*. arXiv: 1508.01991 [abs].

Hunter, T., P. Abbeel, and A. Bayen (2014). "The Path Inference Filter: Model-Based Low-Latency Map Matching of Probe Vehicle Data". In: *IEEE Transactions on Intelligent Transportation Systems* 15.2, pp. 507–529. DOI: 10.1109/TITS.2013.2282352.

Iannella, R. and J. McKinney (2014). *vCard Ontology - for describing People and Organizations*. W3C Note. W3C. URL: http://www.w3.org/TR/2014/NOTE-vcard-rdf-20140522/.

IBM (2016). *What is big data?* URL: https://www-01.ibm.com/software/data/bigdata/what-is-big-data.html (visited on 12/20/2016).

Incel, O. D., M. Kose, and C. Ersoy (2013). "A Review and Taxonomy of Activity Recognition on Mobile Phones". In: *BioNanoScience* 3.2, pp. 145–171. DOI: 10.1007/s12668-013-0088-3.

Intermedia (2014). *Intermedia's Death by 1000 Cloud Apps. The 2014 Intermedia SMB Cloud Landscape Report*. Tech. rep. Intermedia.net. URL: https://www.intermedia.net/assets/pdf/death_by_1000_cloud_apps_ebook.pdf.

Jones, W. (2010). *Keeping Found Things Found: The Study and Practice of Personal Information Management*. Morgan Kaufmann Publishers.

Jones, W. and J. Teevan (2011). *Personal Information Management*. Seattle, Washington, USA: University of Washington Press.

Kang, J. H., W. Welbourne, B. Stewart, and G. Borriello (2004). "Extracting Places from Traces of Locations". In: *Wireless Mobile Applications and Services on WLAN Hotspots*. Proceedings of the 2nd ACM International Workshop. (Philadelphia, PA, USA). WMASH '04. New York, NY, USA: ACM, pp. 110–118. DOI: 10.1145/1024733.1024748.

Karger, D. R., K. Bakshi, D. Huynh, D. Quan, and V. Sinha (2005). "Haystack: A Customizable General-Purpose Information Management Tool for End Users of Semistructured Data". In: *Innovative Data Systems Research*. Second Biennial Conference. CIDR 2005. Asilomar, CA, USA, pp. 13–26. URL: http://cidrdb.org/cidr2005/papers/P02.pdf.

Karich, P. and S. Schröder (2016). *GraphHopper v0.8.2*. URL: http://graphhopper.com/ (visited on 01/10/2017).

Kim, D. H., Y. Kim, D. Estrin, and M. B. Srivastava (2010). "SensLoc: Sensing Everyday Places and Paths Using Less Energy". In: *Proceedings of the 8th*

ACM Conference on Embedded Networked Sensor Systems. SenSys '10. Zürich, Switzerland: ACM, pp. 43–56. DOI: 10.1145/1869983.1869989.

Koller, D. and N. Friedman (2009a). *Probabilistic Graphical Models. Principles and Techniques*. MIT Press. URL: http://mitpress.mit.edu/9780262013192.

— (2009b). "Undirected Graphical Models. Principles and Techniques". In: MIT Press, pp. 103–156. URL: http://mitpress.mit.edu/9780262013192.

komoot (2017). *Photon, search-as-you-type with OpenStreetMap*. URL: https://photon.komoot.de (visited on 01/30/2017).

Koontz, L. D. (2008). *Privacy: Alternatives Exist for Enhancing Protection of Personally Identifiable Information*. Tech. rep. GAO-08-536. United States Government Accountability Office. URL: http://www.gao.gov/products/GAO-08-536.

Kraak, M.-J. (2003). "The space-time cube revisited from a geovisualization perspective". In: *International Cartographic Conference*. Proceedings of the 21st. ICC 2003, pp. 1988–1996.

Kwapisz, J. R., G. M. Weiss, and S. A. Moore (2011). "Activity Recognition Using Cell Phone Accelerometers". In: *ACM SIGKDD Explorations Newsletter* 12.2, pp. 74–82. DOI: 10.1145/1964897.1964918.

Lane, N. D., E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell (2010). "A survey of mobile phone sensing". In: *IEEE Communications Magazine* 48.9, pp. 140–150. DOI: 10.1109/MCOM.2010.5560598.

Lara, O. D. and M. A. Labrador (2013). "A Survey on Human Activity Recognition using Wearable Sensors". In: *IEEE Communications Surveys and Tutorials* 15.3, pp. 1192–1209. DOI: 10.1109/SURV.2012.110112.00192.

Le Bras, T. (2015). *Online Overload — It's Worse Than You Thought*. Dashlane. URL: https://blog.dashlane.com/infographic-online-overload-its-worse-than-you-thought/ (visited on 01/12/2017).

Lefort, L., C. Henson, and K. Taylor (2011). *Semantic Sensor Network XG Final Report*. Tech. rep. World Wide Web Consortium. URL: https://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/.

Levesque, H. J. (1986). "Knowledge Representation and Reasoning". In: *Annual Review of Computer Science* 1, pp. 255–287. DOI: 10.1146/annurev.cs.01.060186.001351.

Li, Q., Y. Zheng, X. Xie, Y. Chen, W. Liu, and W.-Y. Ma (2008). "Mining User Similarity Based on Location History". In: *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM GIS 2008. (Irvine, CA, USA). New York, NY, USA: ACM, 34:1–34:10. DOI: 10.1145/1463434.1463477.

Liao, L. (2006). "Location-Based Activity Recognition". PhD thesis. University of Washington.

Liao, L., D. Fox, and H. Kautz (2007). "Extracting Places and Activities from GPS Traces Using Hierarchical Conditional Random Fields". In: *The International Journal of Robotics Research* 26.1, pp. 119–134. DOI: 10.1177/0278364907073775.

Liao, L., D. J. Patterson, D. Fox, and H. Kautz (2007). "Learning and inferring transportation routines". In: *Artificial Intelligence* 171.5–6, pp. 311–331. DOI: 10.1016/j.artint.2007.01.006.

Lightbend Inc. (2017). *Akka*. URL: https://akka.io/ (visited on 01/30/2017).

Lou, Y., C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang (2009). "Map-matching for Low-sampling-rate GPS Trajectories". In: *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. GIS '09. (Seattle, Washington). New York, NY, USA: ACM, pp. 352–361. DOI: 10.1145/1653771.1653820.

Lovett, T., E. O'Neill, J. Irwin, and D. Pollington (2010). "The Calendar As a Sensor: Analysis and Improvement Using Data Fusion with Social Networks and Location". In: *Proceedings of the 12th ACM International Conference on Ubiquitous Computing*. UbiComp '10. (Copenhagen, Denmark). New York, NY, USA: ACM, pp. 3–12. DOI: 10.1145/1864349.1864352.

Lv, M., L. Chen, and G. Chen (2012). "Discovering Personally Semantic Places from GPS Trajectories". In: *Information and Knowledge Management*. Proceedings of the 21st ACM International Conference. CIKM '12. Maui, Hawaii, USA: ACM, pp. 1552–1556. DOI: 10.1145/2396761.2398471.

Mander, J. (2015). *Internet users have average of 5.54 social media accounts*. Global Web Index. URL: https://www.globalwebindex.net/blog/internet-users-have-average-of-5-social-media-accounts (visited on 01/12/2017).

Manzoni, V., D. Maniloff, K. Kloeckl, and C. Ratti (2010). *Transportation mode identification and real-time CO2 emission estimation using smartphones*. Tech. rep. SENSEable City Lab, MIT, Cambridge.

Maurer, U., A. Smailagic, D. P. Siewiorek, and M. Deisher (2006). "Activity Recognition and Monitoring Using Multiple Sensors on Different Body Positions". In: *Proceedings of the International Workshop on Wearable and Implantable Body Sensor Networks*. BSN'06. (Cambridge, MA, USA), pp. 116–120. DOI: 10.1109/BSN.2006.6.

Mayrhofer, A. and C. Spanring (2010). *A Uniform Resource Identifier for Geographic Locations ('geo' URI)*. RFC 5870. IETF. URL: https://tools.ietf.org/html/rfc5870.

Mendhak (2011). *Lightweight GPS Logging Application For Android*. URL: https://github.com/mendhak/gpslogger/ (visited on 04/18/2016).

Microsoft (2017). *Windows Phone API reference*. URL: https://msdn.microsoft.com/en-us/library/windows/apps/ff626516(v=vs.105).aspx (visited on 01/30/2017).

Mike, B. and A. Robin (2014). *OGC® SensorML: Model and XML Encoding Standard*. Tech. rep. 12-000. Open Geospatial Consortium. URL: http://www.opengis.net/doc/IS/SensorML/2.0.

Miklós, B. (2015). *Computer, respond to this email: Introducing Smart Reply in Inbox by Gmail*. URL: https://blog.google/products/gmail/computer-respond-to-this-email/ (visited on 12/28/2016).

Miller, L. and D. Connolly (2005). *RDF Calendar - an application of the Resource Description Framework to iCalendar Data*. W3C Note. W3C. URL: http://www.w3.org/TR/2005/NOTE-rdfcal-20050929/.

Mockapetris, P. (1987). *Domain names – concepts and facilities*. RFC 1034. IETF. URL: https://tools.ietf.org/html/rfc1034.

Montjoye, Y.-A. de, E. Shmueli, S. S. Wang, and A. S. Pentland (2014). "openPDS: Protecting the Privacy of Metadata through SafeAnswers". In: *PLOS ONE* 9.7, pp. 1–9. DOI: 10.1371/journal.pone.0098790.

Moynihan, T. (2016). *How Google's AI Auto-Magically Answers Your Emails*. URL: https://www.wired.com/2016/03/google-inbox-auto-answers-emails/ (visited on 12/28/2016).

Naragon, K. (2015). *Subject: Email we just can't get enough*. URL: https://blogs.adobe.com/conversations/2015/08/email.html (visited on 12/28/2016).

Nardi, B. A., D. J. Schiano, and M. Gumbrecht (2004). "Blogging As Social Activity, or, Would You Let 900 Million People Read Your Diary?" In: *Computer Supported Cooperative Work*. Proceedings of the 2004 ACM Conference. CSCW '04. Chicago, Illinois, USA: ACM, pp. 222–231. DOI: 10.1145/1031607.1031643.

Nepomuk Consortium and OSCAF (2007). *OSCAF Ontologies*. URL: http://oscaf.sourceforge.net/ (visited on 05/15/2017).

Newson, P. and J. Krumm (2009). "Hidden Markov Map Matching Through Noise and Sparseness". In: *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. GIS '09. (Seattle, Washington). New York, NY, USA: ACM, pp. 336–343. DOI: 10.1145/1653771.1653818.

Nishida, K., H. Toda, and Y. Koike (2015). "Extracting Arbitrary-shaped Stay Regions from Geospatial Trajectories with Outliers and Missing Points". In: *Computational Transportation Science*. Proceedings of the 8th ACM SIGSPA-

TIAL International Workshop. IWCTS 2015. ACM. Seattle, Washington, USA, pp. 1–6.

Office of Oversight and Investigations Majority Staff (2013). *A Review of the Data Broker Industry: Collection, Use, and Sale of Consumer Data for Marketing Purposes*. Tech. rep. United States Senate Committee on Commerce, Science, and Transportation.

O'Hara, K., M. M. Tuffield, and N. Shadbolt (2008). "Lifelogging: Privacy and empowerment with memories for life". In: *Identity in the Information Society* 1.1, pp. 155–172. DOI: 10.1007/s12394-009-0008-4.

Oliver, N. and E. Horvitz (2005). "A Comparison of HMMs and Dynamic Bayesian Networks for Recognizing Office Activities". In: *User Modeling 2005*. Proceedings of the 10th International Conference, UM 2005. Ed. by L. Ardissono, P. Brna, and A. Mitrovic. Vol. 3538. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 199–209. DOI: 10.1007/11527886_26.

Otten, H., L. Hildebrandt, T. Nagel, M. Dörk, and B. Müller (2015). "Are there networks in maps? An experimental visualization of personal movement data". In: *Personal Visualization Workshop*. Proceedings of the IEEE VIS 2015. Chicago, USA. URL: https://uclab.fh-potsdam.de/wp/wp-content/uploads/ShiftedMaps_Paper_V2.pdf.

ownCloud (2016). *ownCloud — A safe home for all your data*. URL: https://owncloud.org/ (visited on 01/15/2017).

Pärkkä, J., M. Ermes, P. Korpipää, J. Mäntyjärvi, J. Peltola, and I. Korhonen (2006). "Activity Classification Using Realistic Data From Wearable Sensors". In: *IEEE Transactions on Information Technology in Biomedicine* 10.1, pp. 119–128. DOI: 10.1109/TITB.2005.856863.

Perreault, S. (2011). *vCard Format Specification*. RFC 6350. IETF. URL: https://tools.ietf.org/html/rfc6350.

Peterson, D., S. Gao, A. Malhotra, C. M. Sperberg-McQueen, and H. S. Thompson (2012). *W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes*. W3C Recommendation. W3C. URL: https://www.w3.org/TR/xmlschema11-2/.

Poikola, A., K. Kuikkanieni, and H. Honko (2015). *MyData — A Nordic Model for human-centered personal data management and processing*. Tech. rep. Finnish Ministry of Transport and Communications. URL: http://urn.fi/URN:ISBN:978-952-243-455-5 (visited on 01/15/2017).

Polleres, A., A. Passant, and P. Gearon (2013). *SPARQL 1.1 Update*. W3C Recommendation. W3C. URL: http://www.w3.org/TR/2013/REC-sparql11-update-20130321/.

Porter, M. E. and V. E. Millar (1985). "How information gives you competitive advantage". In: *Harvard Business Review* 63.4, pp. 149–160.

PostgreSQL Global Development Group (2017). *PostgreSQL*. URL: http://www.postgresql.org (visited on 01/30/2017).

ProtoGeo (2013). *Moves — Activity Diary for iPhone and Android*. URL: https://moves-app.com/ (visited on 01/05/2017).

Quddus, M. A., W. Y. Ochieng, and R. B. Noland (2007). "Current map-matching algorithms for transport applications: State-of-the art and future research directions". In: *Transportation Research Part C: Emerging Technologies* 15.5, pp. 312–328. DOI: 10.1016/j.trc.2007.05.002.

Ravi, N., N. Dandekar, P. Mysore, and M. L. Littman (2005). "Activity Recognition from Accelerometer Data". In: *Proceedings of the 17th Conference on Innovative Applications of Artificial Intelligence*. IAAI-05. (Pittsburgh, PA, USA). Ed. by N. Jacobstein and B. Porter. Vol. 5. American Association for Artificial Intelligence, pp. 1541–1546.

Reactive Streams Special Interest Group (2017). *Reactive Streams*. URL: http://www.reactive-streams.org/ (visited on 01/30/2017).

Reddy, S., M. Mun, J. Burke, D. Estrin, M. Hansen, and M. Srivastava (2010). "Using Mobile Phones to Determine Transportation Modes". In: *ACM Transactions on Sensor Networks* 6.2, 13:1–13:27. DOI: 10.1145/1689239.1689243.

Regalado, A. (2016). "Life Logging Is Dead. Long Live Life Logging?" In: *MIT Technology Review*. URL: https://www.technologyreview.com/s/601300/life-logging-is-dead-long-live-life-logging/.

— (2013). "Stephen Wolfram on Personal Analytics". In: *MIT Technology Review*. URL: https://www.technologyreview.com/s/514356/stephen-wolfram-on-personal-analytics/.

Resnick, P. W. (2008). *Internet Message Formats*. RFC 5322. IETF. URL: https://tools.ietf.org/html/rfc53222.

Rossini, A. (2012). *The sad story of the vCard format and its lack of interoperability*. URL: https://www.ietf.org/mail-archive/web/vcarddav/current/msg02671.html (visited on 12/28/2016).

Rowley, J. (2007). "The wisdom hierarchy: representations of the DIKW hierarchy". In: *Journal of Information Science* 33.2, pp. 163–180. DOI: 10.1177/0165551506070706.

Schmandt-Besserat, D. (1996). *How writing came about*. University of Texas Press.

Sellen, A. J. and S. Whittaker (2010). "Beyond Total Capture: A Constructive Critique of Lifelogging". In: *Communications of the ACM* 53.5, pp. 70–77. DOI: 10.1145/1735223.1735243.

Seth, S. (2011). *Introducing schema.org: A Collaboration on Structured Data*. URL: http://www.ysearchblog.com/2011/06/02/introducing-schema-org-a-collaboration-on-structured-data/ (visited on 01/30/2017).

Settles, B. (2004). "Biomedical Named Entity Recognition Using Conditional Random Fields and Rich Feature Sets". In: *Natural Language Processing in Biomedicine and its Applications*. Proceedings of the International Joint Workshop. NLPBA '04. Geneva, Switzerland: Association for Computational Linguistics, pp. 104–107. URL: http://dl.acm.org/citation.cfm?id=1567594.1567618.

Shoaib, M., S. Bosch, O. D. Incel, H. Scholten, and P. J. M. Havinga (2015). "A Survey of Online Activity Recognition Using Mobile Phones". In: *Sensors* 15.1, pp. 2059–2085. DOI: 10.3390/s150102059.

Sjöberg, M., H.-H. Chen, P. Floréen, M. Koskela, K. Kuikkaniemi, T. Lehtiniemi, and J. Peltonen (2016). "Digital Me: Controlling and Making Sense of My Digital Footprint". In: *Symbiotic Interaction*. 5th International Workshop, Symbiotic 2016. Vol. 9961. Lecture Notes in Computer Science. Cham: Springer, pp. 155–167. DOI: 10.1007/978-3-319-57753-1_14.

Solove, D. J. (2007). "'I've Got Nothing to Hide' and Other Misunderstandings of Privacy". In: *San Diego Law Review* 44, p. 745. URL: http://ssrn.com/abstract=998565.

— (2008). *Understanding Privacy*. Harvard University Press.

Soules, C. A. N. and G. R. Ganger (2005). "Connections: Using Context to Enhance File Search". In: SOSP '05, pp. 119–132. DOI: 10.1145/1095810.1095822.

Speier, C., J. S. Valacich, and I. Vessey (1999). "The Influence of Task Interruption on Individual Decision Making: An Information Overload Perspective". In: *Decision Sciences* 30.2, pp. 337–360. DOI: 10.1111/j.1540-5915.1999.tb01613.x.

Stenneth, L., O. Wolfson, P. S. Yu, and B. Xu (2011). "Transportation Mode Detection Using Mobile Phones and GIS Information". In: *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. GIS '11. (Chicago, Illinois). New York, NY, USA: ACM, pp. 54–63. DOI: 10.1145/2093973.2093982.

Suchanek, F. M., S. Abiteboul, and P. Senellart (2011). "PARIS: Probabilistic Alignment of Relations, Instances, and Schema". In: *Proceedings of the VLDB Endowment* 5.3, pp. 157–168. DOI: 10.14778/2078331.2078332.

Suchanek, F. M., G. Kasneci, and G. Weikum (2007). "Yago: A Core of Semantic Knowledge". In: *World Wide Web*. Proceedings of the 16th International Conference. WWW '07. Banff, Alberta, Canada: ACM, pp. 697–706. DOI: 10.1145/1242572.1242667.

Sun, C. and C. Ellis (1998). "Operational Transformation in Real-time Group Editors: Issues, Algorithms, and Achievements". In: *Computer Supported Cooperative Work*. Proceedings of the 1998 ACM Conference. CSCW '98. Seattle, Washington, USA: ACM, pp. 59–68. DOI: 10.1145/289444.289469.

Sutton, C., A. McCallum, et al. (2012). "An introduction to conditional random fields". In: *Foundations and Trends® in Machine Learning* 4.4, pp. 267–373. DOI: 10.1561/2200000013.

Team, T. O. D. (2017). *OpenLayers*. URL: https://openlayers.org/ (visited on 01/30/2017).

Teevan, J. (2007). "The Re:Search Engine: Simultaneous Support for Finding and Re-finding". In: *User Interface Software and Technology*. Proceedings of the 20th Annual ACM Symposium. UIST '07. Newport, Rhode Island, USA: ACM, pp. 23–32. DOI: 10.1145/1294211.1294217.

Teevan, J., S. T. Dumais, and E. Horvitz (2005). "Personalizing Search via Automated Analysis of Interests and Activities". In: *Research and Development in Information Retrieval*. Proceedings of the 28th Annual International ACM SIGIR Conference. SIGIR '05. Salvador, Brazil: ACM, pp. 449–456. DOI: 10.1145/1076034.1076111.

The Apache Software Foundation (2017a). *Apache CouchDB*. URL: https://couchdb.apache.org/ (visited on 01/30/2017).

— (2017b). *Apache Lucene*. URL: https://lucene.apache.org/ (visited on 01/30/2017).

The New York Times Company (2012). *OpenPaths*. URL: https://openpaths.cc/ (visited on 01/05/2017).

Thiagarajan, A., J. Biagioni, T. Gerlich, and J. Eriksson (2010). "Cooperative Transit Tracking using Smart-phones". In: *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*. SenSys '10. (Zürich, Switzerland). New York, NY, USA: ACM, pp. 85–98. DOI: 10.1145/1869983.1869993.

Thiagarajan, A., L. Ravindranath, K. LaCurts, S. Madden, H. Balakrishnan, S. Toledo, and J. Eriksson (2009). "VTrack: Accurate, Energy-aware Road Traffic Delay Estimation Using Mobile Phones". In: *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*. SenSys '09. Berkeley, CA, USA: ACM, pp. 85–98. DOI: 10.1145/1644038.1644048.

Thudt, A., D. Baur, and S. Carpendale (2013). "Visits: A Spatiotemporal Visualization of Location Histories". In: *Visualization*. Proceedings of the 15th Eurographics Conference. EuroVis 2013. The Eurographics Association. DOI: 10.2312/PE.EuroVisShort.EuroVisShort2013.079-083.

Tilde Inc. (2017). *Ember.js – A framework for creating ambitious web applications*. URL: https://emberjs.com/ (visited on 01/30/2017).

Trevithick, P. and M. Ruddy (2012). *Higgins — Personal Data Service*. URL: http://www.eclipse.org/higgins/ (visited on 01/15/2017).

Tridgell, A. and P. Mackerras (1996). *The rsync algorithm*. Tech. rep. The Australian National University. DOI: 1885/40765.

Tulving, E. (1972). "Episodic and Semantic Memory". In: *Organization of Memory*. Ed. by E. Tulving and W. Donaldson. Academic Press, pp. 382–404.

Uldis, B. and J. G. Breslin (2010). *SIOC Core Ontology Specification*. Tech. rep. The SIOC initiative. URL: http://rdfs.org/sioc/spec/.

Vassiliadis, P. (2009). "A Survey of Extract–Transform–Load Technology". In: *International Journal of Data Warehousing and Mining* 5.3, pp. 1–27. DOI: 10.4018/jdwm.2009070101.

Venkatraman, N. (1994). "IT-Enabled Business Transformation: From Automation to Business Scope Redefinition". In: *Sloan Management Review* 35.2, pp. 73–87.

Vianna, D., A. Yong, C. Xia, A. Marian, and T. D. Nguyen (2014). "A tool for personal data extraction". In: *Data Engineering Workshops*. 2014 IEEE 30th International Conference. ICDE 2014. Chicago, IL, USA, pp. 80–83. DOI: 10.1109/ICDEW.2014.6818307.

Vrandečić, D. and M. Krötzsch (2014). "Wikidata: A Free Collaborative Knowledgebase". In: *Communications of the ACM* 57.10, pp. 78–85. DOI: 10.1145/2629489.

W3C (2017). *ConverterToRdf – W3C Wiki*. URL: https://www.w3.org/wiki/ConverterToRdf (visited on 01/30/2017).

Wache, H., T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner (2001). "Ontology-Based Integration of Information — A Survey of Existing Approaches". In: *Ontologies and Information Sharing*. Proceedings of the IJCAI-01 Workshop, pp. 108–117.

Walker, C. and H. Alrehamy (2015). "Personal Data Lake with Data Gravity Pull". In: *Big Data and Cloud Computing*. Proceedings of the Fifth IEEE International Conference. BDCloud 2015. Dalian, China, pp. 160–167. DOI: 10.1109/BDCloud.2015.62.

Wang, S., C. Chen, and J. Ma (2010). "Accelerometer based transportation mode recognition on mobile phones". In: *2010 Asia-Pacific Conference on Wearable Computing Systems*. APWCS 2010. (Shenzhen, China). Los Alamitos, CA, USA: IEEE Computer Society, pp. 44–46. DOI: 10.1109/APWCS.2010.18.

WhatsApp Inc. (2016). *One billion*. URL: https://blog.whatsapp.com/616/One-billion (visited on 01/30/2017).

— (2017). *WhatsApp*. URL: https://www.whatsapp.com/ (visited on 01/30/2017).

Whittaker, S., V. Bellotti, and J. Cwizdka (2011). "Everything through Email". In: *Personal information management*. Ed. by W. Jones and J. Teevan. Seattle, Washington, USA: University of Washington Press. Chap. 10, pp. 167–189.

Whittaker, S. and C. Sidner (1996). "Email Overload: Exploring Personal Information Management of Email". In: *Human Factors in Computing Systems*. Proceedings of the SIGCHI Conference. CHI '96. Vancouver, British Columbia, Canada: ACM, pp. 276–283. DOI: 10.1145/238386.238530.

Wiesmann, M., A. Schiper, F. Pedone, B. Kemme, and G. Alonso (2000). "Database Replication Techniques: A Three Parameter Classification". In: *Reliable Distributed Systems*. Proceedings 19th IEEE Symposium. SRDS 2000. IEEE. Nürnberg, Germany, pp. 206–215. DOI: 10.1109/RELDI.2000.885408.

Wikipedia contributors (2017). *Intelligent personal assistant – Comparison of assistants*. URL: https://en.wikipedia.org/w/index.php?title=Intelligent_personal_assistant&oldid=759875344#Comparison_of_assistants (visited on 01/15/2017).

— (2016). *List of search engines – Desktop search engines*. URL: https://en.wikipedia.org/w/index.php?title=List_of_search_engines&oldid=755820923#Desktop_search_engines (visited on 01/15/2017).

Wolfram, S. (2012). *The Personal Analytics of My Life*. URL: http://blog.stephenwolfram.com/2012/03/the-personal-analytics-of-my-life/ (visited on 12/28/2016).

Wood, D., R. Cyganiak, and M. Lanthaler (2014). *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation. W3C. URL: http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

Wright, A. (2016). "Self-Tracking: Reflections from the BodyTrack Project". In: *Science and Engineering Ethics*. DOI: 10.1007/s11948-016-9801-2.

Xamarin Inc. (2017). *Mobile App Development & App Creation Software – Xamarin*. URL: https://www.xamarin.com/ (visited on 01/30/2017).

Zheng, Y., Y. Chen, Q. Li, X. Xie, and W.-Y. Ma (2010). "Understanding Transportation Modes Based on GPS Data for Web Applications". In: *ACM Transactions on the Web* 4.1, 1:1–1:36. DOI: 10.1145/1658373.1658374.

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Une base de connaissance personnelle intégrant les données d'un utilisateur et une chronologie de ses activités
Mots clefs : gestion de données personnelles, reconnaissance d'activité, intégration de données, reconnaissance de mode de transport, bases de connaissance, données des capteurs d'un téléphone intelligent

Résumé : Aujourd'hui, la plupart des internautes ont leurs données dispersées dans plusieurs appareils, applications et services. La gestion et le contrôle de ses données sont de plus en plus difficiles. Dans cette thèse, nous adoptons le point de vue selon lequel l'utilisateur devrait se voir donner les moyens de récupérer et d'intégrer ses données, sous son contrôle total. À ce titre, nous avons conçu un système logiciel qui intègre et enrichit les données d'un utilisateur à partir de plusieurs sources hétérogènes de données personnelles dans une base de connaissances RDF. Le logiciel est libre, et son architecture innovante facilite l'intégration de nouvelles sources de données et le développement de nouveaux modules pour inférer de nouvelles connaissances. Nous montrons tout d'abord comment l'activité de l'utilisateur peut être déduite des données des capteurs de son téléphone intelligent. Nous présentons un algorithme pour retrouver les *points de séjour* d'un utilisateur à partir de son historique de localisation. À l'aide de ces données et de données provenant d'autres capteurs de son téléphone, d'informations géographiques provenant d'OpenStreetMap, et des horaires de transports en commun, nous présentons un algorithme de reconnaissance du mode de transport capable de retrouver les différents modes et lignes empruntés par un utilisateur lors de ses déplacements. L'algorithme reconnaît l'itinéraire pris par l'utilisateur en retrouvant la séquence la plus probable dans un champ aléatoire conditionnel dont les probabilités se basent sur la sortie d'un réseau de neurones artificiels. Nous montrons également comment le système peut intégrer les données du courrier électronique, des calendriers, des carnets d'adresses, des réseaux sociaux et de l'historique de localisation de l'utilisateur dans un ensemble cohérent. Pour ce faire, le système utilise un algorithme de résolution d'entité pour retrouver l'ensemble des différents comptes utilisés par chaque contact de l'utilisateur, et effectue un alignement spatio-temporel pour relier chaque point de séjour à l'événement auquel il correspond dans le calendrier de l'utilisateur. Enfin, nous montrons qu'un tel système peut également être employé pour faire de la synchronisation multi-système/multi-appareil et pour pousser de nouvelles connaissances vers les sources. Les résultats d'expériences approfondies sont présentés.

Title: A personal knowledge base integrating user data and activity timeline
Keywords: personal information management, activity recognition, data integration, transportation mode recognition, knowledge bases, mobile device sensor data

Abstract: Typical Internet users today have their data scattered over several devices, applications, and services. Managing and controlling one's data is increasingly difficult. In this thesis, we adopt the viewpoint that the user should be given the means to gather and integrate her data, under her full control. In that direction, we designed a system that integrates and enriches the data of a user from multiple heterogeneous sources of personal information into an RDF knowledge base. The system is open-source and implements a novel, extensible framework that facilitates the integration of new data sources and the development of new modules for deriving knowledge. We first show how user activity can be inferred from smartphone sensor data. We introduce a time-based clustering algorithm to extract *stay points* from location history data. Using data from additional mobile phone sensors, geographic information from OpenStreetMap, and public transportation schedules, we introduce a transportation mode recognition algorithm to derive the different modes and routes taken by the user when traveling. The algorithm derives the itinerary followed by the user by finding the most likely sequence in a linear-chain conditional random field whose feature functions are based on the output of a neural network. We also show how the system can integrate information from the user's email messages, calendars, address books, social network services, and location history into a coherent whole. To do so, it uses entity resolution to find the set of avatars used by each real-world contact and performs spatiotemporal alignment to connect each stay point with the event it corresponds to in the user's calendar. Finally, we show that such a system can also be used for multi-device and multi-system synchronization and allow knowledge to be pushed to the sources. We present extensive experiments.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France