DELIVERABLE D4.1
Design for tools for improved data management

H2020-EU: **PIMCity**
Project No. 871370
Start date of project: 01/12/2019
Duration: 33 months

Revision:
Deliverable delivery: 07/12/2020
Deliverable due date: 30/11/2020

Document Information

Document Name: Deliverable 4.1 – Design for tools for improved data management
WP4 - Tools for improved data management
Author: TID and all WP4 Partners

Dissemination Level

Project co-funded by the EC within the H2020 Programme		
PU	Public	☑
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

(Tick the corresponding dissemination level of the deliverable according to Annex I).

Contributions

	Name	Entity	Date
Author	Kleomenis Katevas	TID	30/11/2020
Author	Nicolas Kourtellis	TID	30/11/2020
Author	Panagiotis Papadopoulos	TID	30/11/2020
Author	Diego Perino	TID	30/11/2020
Author	Alvaro Garcia-Recuero	IMDEA	30/11/2020
Author	Roberto Gonzalez	NEC	30/11/2020
Author	Daniel Oñoro	NEC	30/11/2020
Author	Mathias Niepert	NEC	30/11/2020
Author	Bhushan Kotnis	NEC	30/11/2020
Author	Roberto Bifulco	NEC	30/11/2020
Author	Alvaro Garcia-Recuero	IMDEA	30/11/2020
Author	Devris Isler	IMDEA	30/11/2020
Author	Xavi Olivares	LSTECH	30/11/2020
Author	Dimitrios Delopoulos	LSTECH	30/11/2020
Author	Evangelos Kotsifakos	LSTECH	30/11/2020
Author	Alvaro Garcia-Recuero	IMDEA	30/11/2020
Author	Evangelos Kotsifakos	LSTECH	30/11/2020
WP Leader	Nicolas Kourtellis	TID	30/11/2020
Coordinator	Marco Mellia	POLITO	30/11/2020

Document history

Revision	Date	Modification
Version 1	30/11/2020	V1

List of abbreviations and acronyms

Abbreviation	Meaning
G.A.	Grant Agreement
CA	Consortium Agreement
GA	General Assembly
PB	Project Board
PC	Project Coordinator
PrO	Project Office
IR	Interim Reports
PIMS	Personal Information Management Systems
DA	Data Aggregation
DPC	Data Portability Control
DP	Data Provenance
UPS	User Profiling System
QS	Quantified Self
P-DS	Personal Data Safe
D-TE	Data Trading Engine
DP	Data Provenance
DKE	Data Knowledge Extraction
PDK	PIMS Development Kit

Table of Contents

1. Introduction

Data management has been considered in different contexts, and especially in the online world, in an effort to empower data producers (in our case, end-users) to be able to manage the data they produce and/or collect. For online end-users, nowadays, data management typically means front-end User Interfaces that are connected to back-end systems and allow them to query said systems for some visuals of the users' data, to gain insights on the data collected. However, such systems rarely extend basic functionalities to the end-user, including modifying or deleting the data, or allowing them to port their data to another system. Furthermore, the end-users typically do not have any knowledge extraction tools in their disposal, for mining their own data, aggregating them in different dimensions, and thus understanding more about themselves. Additionally, if the system collected some data of the user, they could probably share that data with multiple collaborating 3rd parties, and the end-user has no way of knowing who accessed their data, how many times, etc.

In this context, WP4 aims to provide both Personal Information Management Systems (PIMS) companies and non-PIMS companies the necessary tools to ingest and aggregate data in a secure and privacy-preserving way, while offering users data portability and data verification features.

Therefore, the target audiences of these tools are:

1. Existing PIMS willing to improve their products and achieve compliance with the GDPR and other legislative frameworks
2. Generic non-PIMS companies interested in integrating specific data management components to improve their products
3. Direct end users who will be provided with a comprehensive set of tools to visualize, manage, validate, track and export their data, and finally
4. Advertising companies which will be provided with more accurate and updated datasets for better audience targeting.

2. Deliverable Objectives

The goal of the deliverable D4.1 is to present the detailed design of the tools that will be built in Work Package 4 (WP4) and aim to improve data management in the PIMCity project.

The WP4 includes five separate tools:

1. **Data Aggregation (DA):** This module will provide the tools for the data owners to prepare, anonymize and aggregate their data in order to be able to share them in a privacy-preserving way. This module is implemented under task T4.1.

2. **Data Portability Control (DPC):** The purpose of this tool is to allow users to migrate their data to new platforms, in a privacy-preserving fashion. More specifically, it will provide methods for extracting data from one PIMS, process it by filtering out sensitive information and outport it into other PIMS systems. This module is implemented under task T4.2.

3. **Data Provenance (DP):** This module provides Data Provenance capabilities to the PIMCity platform. Its goal is to provide data provenance about the data uploaded to the Personal Data Safe (P-DS) of PIMCity by users and support the data provenance in the transactions of the Data Trading Engine (D-TE) component in order to verify data orders' fulfilment attending to our Data Provenance (DP) module internals. This module is implemented under task T4.3.

4. **User Profiling System (UPS):** This module is the first component of the Data Knowledge Extraction (DKE). It is able to automatically generate a profile of the user, while considering their browsing patterns. The profile will indicate the interests of the user in each of the categories defined by the IAB. This module is implemented under task T4.4.

5. **Quantified Self (QS):** The purpose of the Quantified-self dashboard is to present to the user the benefits of sharing data in PIMCity and let him/her control permissions to the different datasets or taxonomies involved in generating his/her user profile. This module is implemented under task T4.4.

We use as reference the Deliverable D1.1, in which we precisely defined the requirements of the modules included in the WP4, whose design is described in the next sections. As such, we defer the reader to Deliverable 1.1 for a broader description of the PIMCity project, the other PDK modules and the EasyPIMS architecture. D4.1 was built in coordination with the other technical Work Packages (i.e., WP2 and WP3) and their corresponding deliverables D2.1 and D3.2.

3. Data Aggregation

3.1. Tool/Module Objective

The Data Aggregation (DA) tool will enable data owners (for example an ISP) that hold a bulk of their users' data to perform two important processes on their data: aggregation and anonymization. This will allow them to share these data in a privacy-preserving way. This module will reside on the data owner's side. The input to the module will be the raw data that are available through the initial sources (telco data, sensor data, etc.) and they will be transformed in a predefined schema / metadata model. The user (data owner) is responsible to prepare the data for processing (export from their initial source (internal database), clean them if needed, etc.). Afterwards, through the module, the user will be able to choose the subset of the data to be aggregated / anonymized and set the related algorithmic parameters (for aggregation and anonymization). The output will be the processed (aggregated / anonymized) data that could be exported to the PIMCity marketplace through an API that the module will provide. The data will reside on the data owner side and the interested party will be able to retrieve them through this API.

This will be achieved through designing and implementing the data aggregation and anonymization pipelines in a generic way that will allow different types of data to be processed. The tool will be tested on real datasets provided by TID and are mainly related to the end-user's mobility (location, travelling distances etc.). The user (the data owner) will be able to select a number of parameters and apply aggregation and anonymization methods and check their effectiveness (achieved anonymization) in a simple user interface. This interface will be a simple web interface with options to visualize the processed data, in the form of tables and graphs using also basic controls over them.

The outcome of this task will be the metamodels, the flow and the set of properties of the data to be exported, as well as the functions that can be available and can be applied to the data, in order to allow aggregation in a privacy-preserving way.

It should be noted that the anonymization techniques that are to be developed in this module are for the data owners in order to be able to share their data and is different from the implementation of the respecting anonymization algorithms of the P-PPA module described in D2.1. While this is a parallel implementation, depending on the nature of the data at hand for the testing and evaluation of our modules, both implementations will be considered.

3.2. Background and State of the Art

Data aggregation techniques are used to bring together relevant pieces of information to provide a high-level overview of an entity in a scalable, efficient and reliable manner. However, the design and evaluation of such aggregation algorithms have not received the same level of attention that other basic operators, such as joins, have received in the literature.

The nature and purpose of existing aggregation algorithms/techniques is also platform dependent in some cases, restricting their wider application across different scenarios. For example, the querying mechanism for traditional (e.g., SQL or NoSQL) data platforms are different from Hadoop[1]. The industrial tools for data management and self-service data analytics such as Trifacta Wrangler[2] or Alteryx Designer[3] offer desktop-based environments to prepare, aggregate and analyze data.

[1] http://hadoop.apache.org/
[2] https://www.trifacta.com/
[3] https://www.alteryx.com/products/alteryx-platform/alteryx-designer

Moreover, the influence of Big Data techniques in various application domains is seen in the emergence of hybrid approaches for data aggregation. Examples of hybrid techniques include the use of machine learning, predictive and clustering algorithms to perform data aggregation operations. In addition, some existing frameworks for data aggregation focus on combining data from multiple sources in real-time environments (e.g., Apache Spark[4]). Using such technologies, the approaches for aggregating historic datasets still rely on the traditional CRUD (Create, Read, Update and Delete) operations.

In PIMCity, we aim at using these standard techniques to allow the data owners / providers to prepare their data to be shared in the PIMCity ecosystem. Such techniques will be used for data aggregation to provide statistics like count, sum, average over specific data attributes such as location, age and time. Furthermore, these techniques will support the anonymization part of Task 4.1, supported also from anonymization algorithms to ensure data privacy.

Finding appropriate ways to implement privacy principles in the big data business is the most efficient way to prevent a conflict between privacy and big data. Privacy by design is one of the fundamental mechanisms to address privacy risks from the beginning of the data processing, and apply necessary privacy preserving solutions. Privacy by design was firstly introduced by (Cavoukian, 2011), and elaborated on the notion of embedding privacy measures into the design of Information and Communications Technology (ICT) systems. Consequently, privacy by design empowers the individuals in the big data era, as well as supporting the data controllers' liability and trust.

Anonymization refers to the process of modifying personal data in such a way that individuals cannot be re-identified, and no information about them can be learned. It is applied in several cases of data analysis. Most of the privacy anonymization models fall into two distinct categories. The first includes k-anonymity (Sweeney, 2002) and its extensions (Truta & Vina, 2006) (Machanavajjhala, Kifer, Gehrke, & Venkitasubramaniam, 2007) (Li, Li, & Venkatasubramanian, 2007), while the second consists of the notion of Differential Privacy (Dwork, 2006), alongside with some variations (Gehrke, Hay, Lui, & Pass, 2012) (Machanavajjhala & Kifer, Designing Statistical Privacy for Your Data, 2015). While k-anonymity is a mechanism that is applied on the data to make them a bit fuzzier to prevent revealing individuals, differential privacy is applied on the answers of the queries on the dataset to disclose private information.

Identity disclosure can be prevented with a k-anonymous version of the initial dataset, that is, an original record cannot be distinguished within a group of k records sharing quasi-identifier values. The basic notion of the k-anonymity model and its extensions is to classify the attributes of a dataset into several non-disjoint types:

- **Identifiers:** These are the attributes in the initial dataset that explicitly identify the subject (e.g., passport number). The removal of the identifiers is a requirement to create an anonymized dataset.
- **Quasi-identifiers:** These are the attributes in the initial dataset that in combination can identify the subject (e.g., age, city of residence). Quasi-identifiers cannot be removed, since any attribute is potentially a quasi-identifier.
- **Confidential attributes:** These attributes contain sensitive information on the subject (e.g., salary, health condition).

Differential privacy is the second category of privacy models, which aims to anonymize the answers to interactive queries submitted to a database. Differentially private data sets can be generated by creating a simulated dataset from a differentially private dataset or by adding noise to the attributes of the initial dataset. The main downside of differential privacy is that privacy guarantees deteriorate with repeated use and various techniques to address that need to be considered.

In our PIMCity implementation we will use the K-anonymity approach as it is more popular, easier to implement and there are more implementations available.

[4] https://spark.apache.org/

During our research for the best K-anonymity implementation for our module, we have considered various open-source projects that we describe in the following paragraphs, along with their pros and cons. The current state-of-the-art in anonymization is **ARX**[5]. it is open source, has a GUI version and an API in order to access from external applications. It is fully developed in Java. Its main advantages are that it is the most consolidated implementation of K-anonymity, having a strong community and more online resources for support on the development. The only disadvantage could be its learning curve, since it requires previous knowledge about anonymity algorithms.

Alternatively, to ARX, there is an open-source project called **PrioPrivacy**[6]**,** which is developed by Alexandros Bampoulidis and it is being used in the safe-DEED project[7]. It performs better than ARX with extra arguments by slightly relaxing k-anonymity. It is developed in Java using ARX libraries which is its main advantage. It is based on an academic research, accompanied by the related paper and it is open source. Its main disadvantage is its tough learning curve.

There are also **lighter implementations of K-anonymity**[8] algorithms in Github. However, they are less recognized and less standardized. These are not tested enough and are small projects but can be good alternatives in case we require early results or faster tests. Another viable option regarding K-anonymity methodology is Mondrian14. It is an anonymization algorithm based on the generalization field of K-anonymity. Currently, it is the fastest when processing large datasets, but the amount of distortion is much superior to standard K-anonymity software such as ARX, meaning that the data loss is higher. The most popular project involving this algorithm is developed in Python[9]. Also, there is not enough existing documentation online of the methods that it uses over K-anonymity.

As the last approach on anonymity algorithms, Microsoft released **Presidio**[10], an open-source project designed to anonymize text, images and data. It consists of a local API with multiple endpoints. It already has a Docker distribution uploaded and it is practically a plug-and-play service. It was developed in Python and Golang and is still an active project. The main advantage is the documentation of the entire project, as well as the resources that are already available, such as the docker container. This would save developing time and ease the testing phase. Also, it is widely known in the developer's community, under continuous development with a lot of related resources online. As the main disadvantage, we found out that it lacks parameterization options.

After evaluating the above options in terms of completeness, parametrization, ease of implementation and available expertise, we decided to deploy the ***ARX*** implementation.

Furthermore, the z-anonymity approach that is proposed under WP2 and is described in the deliverable D2.1, section 5.3, will be considered due to its suitability on location and continuously changing data.

Our development on T4.1 will be based on the Python programming language due to our expertise.

3.3. Tool/Module Technical Design

The tool will follow the typical input-processing-output procedure. The first step is the preprocessing of the data that should be done by the user before using the module. The users of the module (the data owners) should pre-process their raw data (transactions, CDRs, etc.) to select and export the attributes they want to aggregate / anonymize and then provide them in a comma-separated value (CSV) file for input to the module. The *input* for this module will be a set of attributes and their data type and description that will be defined by the user. These metadata will be used for the *processing* of the aggregation and the anonymization. Along with these metadata, the actual data to be

[5] https://arx.deidentifier.org/
[6] https://github.com/alex-bampoulidis/prioprivacy
[7] H2020 grant agreement No 825225 https://safe-deed.eu/
[8] Implementation in Python: https://github.com/Nuclearstar/K-Anonymity & Java: https://github.com/ScottWalkerAU/KAnonymity
[9] https://github.com/qiyuangong/Mondrian
[10] https://microsoft.github.io/presidio/

processed will be provided in a flat text-based file (e.g., CSV). The file should have headers that correspond to the metadata provided. Finally, the *output* of this module will be the processed, aggregated, anonymized data in a common data interchange format (e.g., CSV, JSON etc.). A metadata file / data model description will also be provided.

The following diagram (Figure 1) summarizes the process.

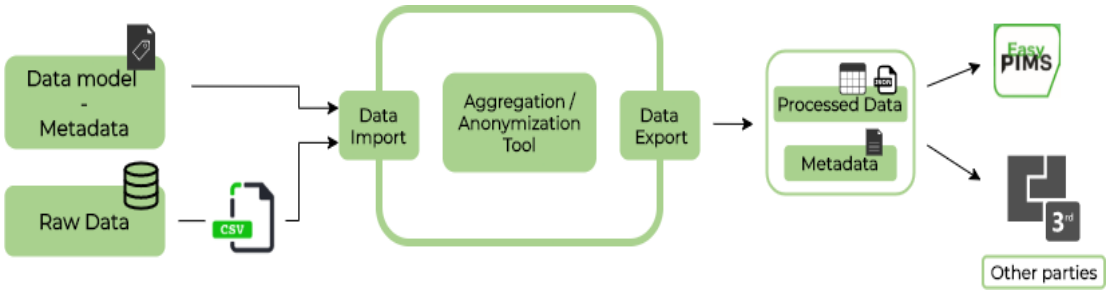

Figure 1: Data Aggregation module design diagram

3.3.1 Tool/Module Operation

This tool is aimed to be used from the data owners to process their data before being able to share them. The user will be able to view a portion of the data and also all the metadata defined on them. The user will choose aggregation options and will be also able to set K-anonymity parameters like the k-value and the quasi-identifiers. Basic graphs can be provided to the user to better understand the effectiveness of the aggregation / anonymization over the raw data.

3.3.2 Tool/Module Interfacing

The module will provide an API to share the aggregated / anonymized data to external services that will use them for their own purposes. These external services can be other modules of the PIMCity implementation or any other third-party application. Their use can vary from re-selling the data to exploit them to provide to the users related analytics.

An output RESTful API will be developed that will allow the data to be exported. The output will have two main parts, the metadata - data model that will describe the data structure and attributes and a part that will include the data themselves, in a common data interchange format like CSV or JSON. At the current stage we cannot define in detail this API, since it depends on the implementation of the modules, as well as the respective needs that will be explored during the project implementation.

4. Data Portability Control

4.1. Tool/Module Objective

The right to data portability is a novelty of the EU's General Data Protection Regulation (GDPR), that allows individuals to obtain and reuse personal data from one environment to another (Hert, Papakonstantinou, Malgieri, Beslay, & Sanchez, 2018). Some platforms now offer an option for exporting user data in structured formats. For instance, Facebook allows users to export their social activity in an archived format. Such approaches suffer from:

a) inconsistent data format between systems, with data structure being limited to simplistic JSON or CSV archives
b) complicated configurations that depend on technical knowledge (e.g., implement OAuth2 connections) or are slow due to the required manual user actions (e.g., type password, request archive, wait for 24 hours and then download it)
c) lack of privacy preserving mechanism, when extracted data includes sensitive information
d) lack of support for data importation

The purpose of this tool is to allow individual users to migrate their data to new platforms, in a privacy-preserving fashion. More specifically, it will provide methods for extracting data from one PIMS (e.g. Facebook, bank, mobile phone), process it by filtering out (e.g., by applying differential privacy) sensitive information such as platform-inferred data (e.g., user unavailability due to event attendance) or user-inputted data (e.g., remove login credentials or debit card numbers), and outport it into the PDK module, a new PIMS (e.g., EasyPIMS) or an exported file in a common data interchange format (e.g., JavaScript Object Notation (JSON)).

4.2. Background and State of the Art

Various solutions for data portability between cloud providers exist. (Shirazi, Kuan, & Dolatabadi, 2012) presented the design patterns for enabling portability between different type of databases and cloud services. (Alomari, Barnawi, & Sakr, 2014) proposed a framework that allows system developers to easily migrate data from one cloud service to another. However, these approaches are platform specific and do not allow user-defined data extraction and processing to new platforms.

Semantic Web[11] was an approach proposed from W3W for modelling user data (usually in the form of social networks) in common semantics that could be easily migrated or used by other platforms. However, these proposals have typically been considered as academic exercises and mostly failed to take off, since there have been no incentives embedded for data processors that operate under information lock-in by building data silos and hoping to maintain their competitive edge against others to enable data portability.

Some examples of systems that allow full data exportation and importation do exists but are usually only limited to Email providers (e.g., Google Workspace, Microsoft 365 Business). Email providers often offer free data importation for migrating Emails, Calendar and Contacts from alternative providers. To our knowledge, no such wholistic tool exist that allows you to connect to some platform, download data, adapt them or filter our sensitive information and then migrate data into another platform.

4.3. Tool/Module Technical Design

The Data Portability Control (DPC) tool will incorporate the necessary tools to import data from multiple platforms (through the available Data Sources), process the data to remove sensitive information (through the Data Transformation Engine), and outport into other platforms (through the Data Export module). Since the tool will not have a UI for interacting with the users, it will provide an interface in a form of a generic Control API for controlling all operations from other modules of the

[11] https://www.w3.org/standards/semanticweb/

PIM system.

The figure below (Figure 2) shows an overview the design of the DPC tool:

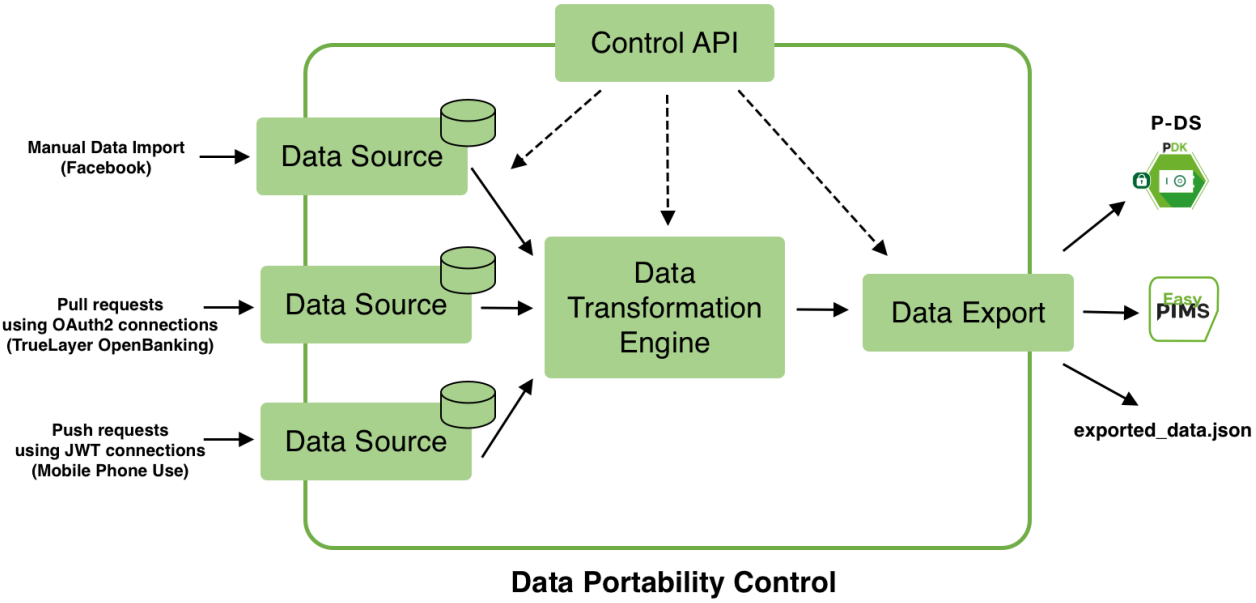

Figure 2: Design Overview of Data Portability Control tool

The next subsection explains all supported operations and provides details about all components of the DPC tool.

4.3.1 Tool/Module Operation

The DPC tool will have a modular design that will make it easily extensible to support different data sources and platforms. It will consist of a series of Data Sources, a Data Transformation engine, and a Data Export module.

The **Data Sources** are drivers for supporting specific platforms (e.g., Facebook, TrueLayer Open Banking, Mobile Phone Use from a smartphone). In its initial release, we will provide three Data Sources that support different types of data importation and cover the majority of such available functions:

a) manual data importation (e.g., Facebook data)
b) pull requests authenticated over the industry-standard protocol OAuth2 for periodically retrieving data (e.g., bank data pulled using the TrueLayer Open Banking service)
c) push requests authorized using the JSON Web Token (JWT) internet standard that a platform can use to submit data to the tool (e.g., Mobile Phone Use pushed from a smartphone)

All provided Data Sources, while specific to a particular platform (i.e., Facebook for manual data import, TrueLayer Open Banking for pull requests, and a smartphone app for push requests), could later be used as templates to support other platforms based on the needs of each partner. All data will be saved in an internal datastore per data source module and will be available for further processing by the Data Transformation Engine when requested.

The **Data Transformation Engine** is a customizable component, responsible for retrieving the data from selected data sources and process the data by filtering out sensitive information. The type of transformation – anonymization – will be based on plugins in the form of Python scripts, called Transformation Programs (e.g., a program can apply differential privacy by injecting random noise into each geo-location point to protect the user's exact location). When processing is completed, data will be passed to the Data Export module for further exportation to other platforms. Note that existing anonymization functions from Data Aggregation tool (Section 3) and Personal Privacy

Preserving Analytics (P-PPA) tool (D2.1) will be compatible with the Data Transformation Engine.

The **Data Export** module will be the final component in the tool's pipeline responsible for outporting the data to supported platforms. It will support three main options for outporting the data:

a) send the data to the P-DS using its interfacing API (P-DS will provide its own data-schema that DPC tool will support for exporting data)
b) send the data to other PIMS (e.g., EasyPIMS)
c) save the data in a common data interchange format (e.g., JavaScript Object Notation (JSON)).

4.3.2 Tool/Module Interfacing

By default, the DPC tool will not have a UI for interacting with the users. Instead, it will provide an interface in the form of a generic Control API for controlling all user-based actions explained above (e.g., authenticate / configure the data sources, start the data import, export and delete data). Any PIMS component can implement this interface and provide an actual UI for the user.

The **Control API** will be implemented as a secure HTTPS service that comforts to the REST architectural style, in order to be compatible with all other PIMS modules. More specifically it will provide the following functions:

- List available Data Stores
- Install / uninstall / configure / clear a Data Store
- List available Transformation Programs
- Install / uninstall / configure a Transformation Program
- Authenticate with external PIMS
- Start / stop data portability
- Download outported data
- Clear all data stores

The Control API will communicate with Python-based internal APIs of all core components of the tool:

A **Data Source API** will exist per available Data Store, that is responsible for authenticating with external data sources, or providing the data in the case of manual data import or push requests, configuring the parameters of the data import (e.g., fields to be imported), start the data import process, and clear the datastores when needed.

The **Data Transformation API** is responsible for installing / uninstalling transformation programs, configuring the available parameters, and finally execute a transformation.

The **Data Export API** is responsible for authenticating with external PIMS for data exportation and start the data exportation.

5. Data Provenance

5.1. Tool/Module Objective

This module will provide Data Provenance capabilities to the PIMCity platform. A common technique in the data protection and/or copyright enforcement in the Big Data business is *Digital watermarking.* It is a technique used for media content distribution with ownership protection in digital markets as those for music or video. To allow users to download purchased content, the content owners insert a mark in the content to trace who leaked the content or prevent future leakage. However, this often also comes at the cost of degrading the quality of the content (e.g., Spotify[12]) or forcing users to use a given application in a sandbox to reproduce the content they purchase (e.g., Netflix[13]), which can be both impractical, but also unethical due to the use of a particular technique. *Database watermarking* deals with heterogenous data of independent objects and tuples. Generally, database watermarking uses a watermarking key for watermark insertion, which can be private or public; whereas for watermark detection the technique also requires use of such key in order to verify the watermarked information belongs to the owner of the key. In general, database watermarking has been approached with robust (an attacker attempts to remove or hide the embedded watermark while keeping the database useful) watermarking techniques that are not distortion free (modify the least significant bits) and initially only worked for numeric data, for which the latter two are a significant limitation. In order to realize our technical requirements for our DP component/module, the next paragraphs discuss the minimum set of requirements (functional and non-functional) the watermarking technique in PIMCity should comply with.

Objective

The goal of this module is to provide data provenance about the data uploaded to the Personal Data Safe (P-DS) of PIMCity by users in order to verify data orders' fulfilment attending to the Data Provenance (DP) module internals. We assume that data consent is automatically informed here or simply already agreed with the user at the Personal Consent Manager (P-CM) module previously.

Watermarking requirements

Regarding requirements, we assume incoming data are already anonymized and consent is approved. Also, we assume our system will not dispute to a true data owner authorship or validity of his/her watermark. Likewise, a data buyer who bought a dataset should be able to provide proof of it. The list of requirements we rely on to develop the DP module are the following:

- Functional requirements: a requirement that specifies what this module/component "should" do. These were defined in deliverable D1.1[14].
- Non-functional requirements: a requirement that specifies "how" this module/component performs a certain function or functionality.

Naturally, regarding Functional requirements we consider the previously listed requirements in deliverable D1.1 for the DP module, namely requirements from R20 (basic output functionality of the DP module) to R24. Here we make them specific to the module components, namely watermarking and fingerprinting techniques in the literature. Regarding R22 (type) and R23 (size) watermarking requirements in D1.1, we will incrementally add those as features as we build the framework, thus becoming option parameters in the operation of the DP framework. R21 and R24 are covered by tracing or preventing data leakage from the watermarking state-of-the-art, whereas fingerprinting may be used together with watermarking. For data leakage prevention, an appropriate solution among public-key watermarking, fingerprinting, traitor tracing, dual watermarking-fingerprinting, DRM[15], or similar one will be chosen if necessary (just tracing may suffice for our use case). Finally, the chosen technique should be compatible with making the watermark specific for each buyer.

As for Non-Functional requirements, we have the following:

[12] https://community.spotify.com/t5/Content-Questions/Audible-watermarks-degrading-sound-on-Spotify-premium/td-p/1300815
[13] https://medium.com/pallycon/how-netflix-protects-contents-part-2-33c1b60002a3
[14] D1.1, PIMCity Requirements and Specifications, https://www.pimcity-h2020.eu/app/uploads/2020/06/D1.1_PIMCity-requirements-and-specifications.pdf (Confidential). https://www.pimcity-h2020.eu/dissemination/
https://www.techopedia.com/definition/3986/digital-rights-management-drm

- **Distortion-free:** watermarks should be as much as possible distortion free though some data types may not avail this so easily.
- **Robustness**: the watermark technique should be strong enough against any processing, malicious or not, that modifies, destroys or erases the watermark.
- **(Un)blindness:** with/without having the original piece of watermarked data, data owners should be able detect their own watermark.
- **Trace/prevent data leakage** (key base watermarking, traitor tracing, fingerprinting)**:** This is equivalent to R21 and R24 in D1.1, enabling us to know who bought and/or leaked the data, which essentially requires the same or similar scheme functionality for both client - buyer.

As stated above, a technique that can fulfil our goal and cover most of these requirements above is watermarking, which works by embedding a hidden piece of information about the source or owner of a piece of data. An advantage of this approach is being able to trace back the responsible for the possible data leakage, namely a data buyer. The disadvantage is that it is difficult to prevent such leakage unless content-screening systems are used together with the watermarking. In order to provide a practical application of our approach using this technique to the problem of data ownership and provenance, we need to ensure the watermarking is compatible and can be used together with the many types of data according to literature (Panah, Van Schyndel, Sellis, & Bertino, 2016) (Bianchi & Piva, 2013) and possibly with screening methods also for leakage prevention (Kirovski, Malvar, & Yacobi, 2002). Also, and in order to fulfil orders from the D-TE and responding to the Data Marketplace with data that is not broken, it should be easy for data owners to verify watermarked data, while being difficult to detect and remove a watermark for data buyers.

5.2. Background and State of the Art

There are three main types of watermarking techniques, whereas each have different applications and share some major properties. The robust watermarking techniques are suitable for applications that require ownership protection, while the fragile ones serve the purpose of data tamper-proofed and integrity. Finally, the dual watermarking-fingerprinting approach provides a mechanism to identify the guilty agent who was responsible for the data leakage.

Main properties

Robustness vs *Fragile:* robust is for resisting malicious or benign watermarked data updates (copyright protection), whereas fragile is used for tamper detection and in order to identify and report every possible region in which someone has tampered with the watermarked data (integrity).

A good watermarking technique should also be *blind*, that is, it should not need the original data or watermark to detect the watermark. Moreover, the embedded watermark should only be detectable by the data owner. Furthermore, the watermark should not deteriorate the original data and data usability should be ensured during the process of watermark insertion (Kamran & Farooq, 2018). Finally, it should be distortion free, meaning that updates or insertions into the actual data do not affect it.

Database Watermarking

The state of the art in Database Watermarking is from the VLDB seminal paper in 2002 (Agrawal & Kiernan, 2002), which basically presents a novel scheme that watermarks a database of tuples, each one with its attributes, using a bit pattern given by a private watermarking key. This technique is only compatible to mark attributes in tuples that are numeric attributes and thus assumes that the marked attributes can tolerate some slight change in value; namely LSB (least significant bits) changes the numeric value insignificantly. It is also compatible with and without primary keys. The marking algorithm here takes the following parameters, out of which three and the private key are kept by the owner (private):

η	Number of tuples in the relation.
v	Number of attributes in the relation available for marking. (private)

ξ	Number of LSB available for marking in an attribute. (private)
$1/\gamma$	Fraction of tuples marked.
ω	Number of tuples marked.
α	Significance level of the test for detecting a watermark.
τ	Minimum number of correctly marked tuples needed for detection.

Figure 3: Insertion in SoA of Database Watermarking

To watermark a database relation *R (P, A_0, …, A_{v-1})* with primary key *P* and *A* number of attributes *v* the algorithm goes as follows:

Setup:

- The private key *k* is known only to the owner of the database.
- The private parameters above are also privately held by the owner.
- Let *F* be a MAC (Message Authenticate Code) function that randomizes the values of the primary key attribute *P* of tuple r (r.*P)* and returns an integer value in a wide range. *F* is seeded with a private key *k* known only to the owner.

We compile the state-of-the-art algorithm for watermarking databases, which works as follows in the watermark insertion phase (Algorithm 1 is in Figure 4 below):

Algorithm 1 Watermaking Insertion
From "Watermarking Relational Databases" in Agrawal, VLDB'02.

```
1: for tuple r ∈ R do
2:    if ( F(r.P) mod γ) = 0 then
3:       attribute index i = F(r.P) mod υ        {//This line marks an attribute}
4:       bit index j = F(r.H) mod ε              {//Mark j_th bit}
5:       r.A_i = mark(r.P, r.A_i,j)
6:    end if
7: end for

   F(mark(primary_key P, number υ, bit_index j)){ first_hash = H(κ || P)
8: if first_hash is even then
9:    set the j_th bit of υ to 0
10: else
11:   set the j_th bit of υ to 1
12: end if
13: return υ              {//Func. returns # of attributes available for marking.}
   }
```

Figure 4: State of the art Algorithm for Insertion of watermark

The idea here is to ensure some bit positions of some of the attributes of some of the tuples contain specific values (bit pattern is the watermark), which are algorithmically determined by the watermarking private key *k* of the data owner. The insertion of the watermark in Algorithm 1 works as follows: we iterate over all tuples in relation *R* and in line 2 we decide if the tuple is going to be marked. To do so, we use a one-way hash function as MAC to define a function *F* in line 2 that will map values of the primary key r.*P* of tuple r to a random value in a wide range due to the private watermark *k* used as seed, and then check it is mod zero to the fraction of tuples selected in the gamma parameter. The scheme requires this private key also to detect the watermark with high probability, but on the other hand it does not require the original data nor the watermark. Line 3 will choose the attributes marked in the attribute space. Line 4 selects the actual bit position for a selected attribute among epsilon least significant bits parameter that are marked. These two lines depend on the private key *k*, so it is difficult to use such parameters successfully even if guessed at random. Guessing the marked attributed and bit positions is not easy. Finally, line 5 decides to insert using the marking function (0 or 1) on the selected bit depending on an even first hash value at line 8 of the mark function for a given private key and primary key.

Regarding watermark detection, we refer to the paper in VLDB for the sake of brevity here, but the main idea as said, is to require the private watermarking key in order to figure out the attributes of the tuples and the bit positions into them when selected and thus marked. Naturally, the detection algorithm must implement a subroutine that determines the attribute and bit position that must have been marked and then compares the current bit value with the value that must have been set for that bit by the watermarking algorithm. The detection algo is probabilistic so among all the lines, we count in how many of them have the expected bit set correctly. In summary, we can find out how many tuples were checked (totalcount) and how many contain the expected value (correct). The correct are compared with the minimum count returned by the threshold function τ = threshold (totalcount, α) for the test to succeed at the chosen level of significance.

Naturally, authors of this work tested the algorithms (insertion, detection) in real world datasets (e.g., ~500k rows with 61 attributes in the Forest Cover Type dataset, available from the University of California–Irvine KDD Archive[16] and the performance overheads of each of them is closely related to the cost of computing hash values needed either to determine the mark of each tuple or determine the presence of such mark in each tuple.

Security

There are several types of attacks in watermarking, insertion, deletion, alteration, multifaceted, additive. Each of the above techniques has its shortcomings in this context.

Regarding attacks on robust watermarking techniques, the main attacks here are based on alteration, deletion, mix-match, and sorting. Generally speaking, watermarking techniques defeat those attacks by the secrecy of a set of inputs to the algorithm that the attacker is not able to obtain, as the embedding secret key and other additional ones. The result is that it is not trivial to detect in which tuples the watermark was inserted and thus how to modify it beyond just random guessing.

Regarding attacks on fragile watermarking techniques, Hamadou et al. (Hamadou, Sun, Gao, & Shah, 2011) proposed a zero-watermarking technique as countermeasure in additive watermarking attacks. The technique is named zero-watermarking because the watermark is not actually inserted in the database but instead its information is registered with a certification authority (CA).

Regarding attacks on dual watermarking-fingerprinting techniques, generally the most common and studied in the literature are collusion attacks (Ergun, Kilian, & Kumar, 1999). Comparing their databases, the attackers find some of the hash entries which do not appear in all colluders' databases. These hashes are usually known as dummy hashes or individualization hashes. However, some more hashes appear as well in each of the attacker's copy of the database to defeat the purpose of attackers guessing their unique database hashes, namely undetected dummy hashes. Based on the percentage of these undetected dummy hashes in the database, we can ascertain some security guarantees of the scheme as explained in the paper of Steinbach et. al (Berchtold, Schäfer, & Steinebach, 2013) of given that such information is enough to trace back the colluders.

5.3. Tool/Module Technical Design

The operational steps of the DP module are depicted in the next Figure 5 below. First, users will send their data from their P-DS module to the Data Provenance module with the objective of adding a mark to their data. First the **Watermark Control** submodule will receive and queue requests from the D-TE component. Once a request is ready to be processed, the **Watermark Printe**r submodule will fetch the raw data from the P-DS and communicate with the Watermark Control submodule, which will satisfy the data trading request coming from the D-TE. Note the Watermark Printer will need to fetch raw data according to the D-TE request from the P-DS and insert a watermark on it. We will also save the watermarks' metadata to a database for later retrieval and tracking of the time, ownership, and so forth for each such piece of data. Therefore, two possible approaches will exist

[16] http://kdd.ics.uci.edu/databases/covertype/covertype.html

for the actual watermarking operation at hand: data is watermarked prior to any transaction or the watermarking occurs once a transaction is processed and thus the requestor includes such a reference to the raw data at the transaction level. We will consider the latter to be most optimal approach as in fact we propose to have a complete view of transaction's metadata coming from the D-TE module. Here, we aim to keep the original watermarked metadata as intact as possible and de-duplicated so that the transaction history lives in a separate database that supports our DP submodules above. The latter will allow us to log transactions in a private and immutable database of D-TE requests, which effectively is a history of transactions also useful for later verification of watermarked metadata as required on demand.

The DP module will incorporate the necessary tools to show data ownership for users uploading data and for the data buyers to be able to request transactions through the D-TE module. Also, to confirm the validity and ownership of the data purchased we may verify such a watermark by simply comparing the watermarked data of a given data buyer with the secret key that only a data owner holds, so that revealing such secret or private key can convince a judging party if the data owner or buyer is correct. Ideally, this could be done without the judge or service provider learning the secret directly but indirectly in an outsourced protocol.

In summary, our DP module will be responsible for marking data and keeping a historical log or marking transactions in the system before data is delivered to the data buyer. In this manner, we can count how many times a particular item has been seen and by whom in the system. Additionally, we can consider holding metadata about the identity of the watermarking source and destination in the immutable database. Naturally, the watermarked data will be released to the marketplace as seen below, so the DP module does not hold data in this database, the reason being that a user that buys a dataset could store the data themselves once purchased without the risk of forging our watermark. This presents some challenges and choices to be made for operating a content-screening system, but this is beyond the design in this deliverable.

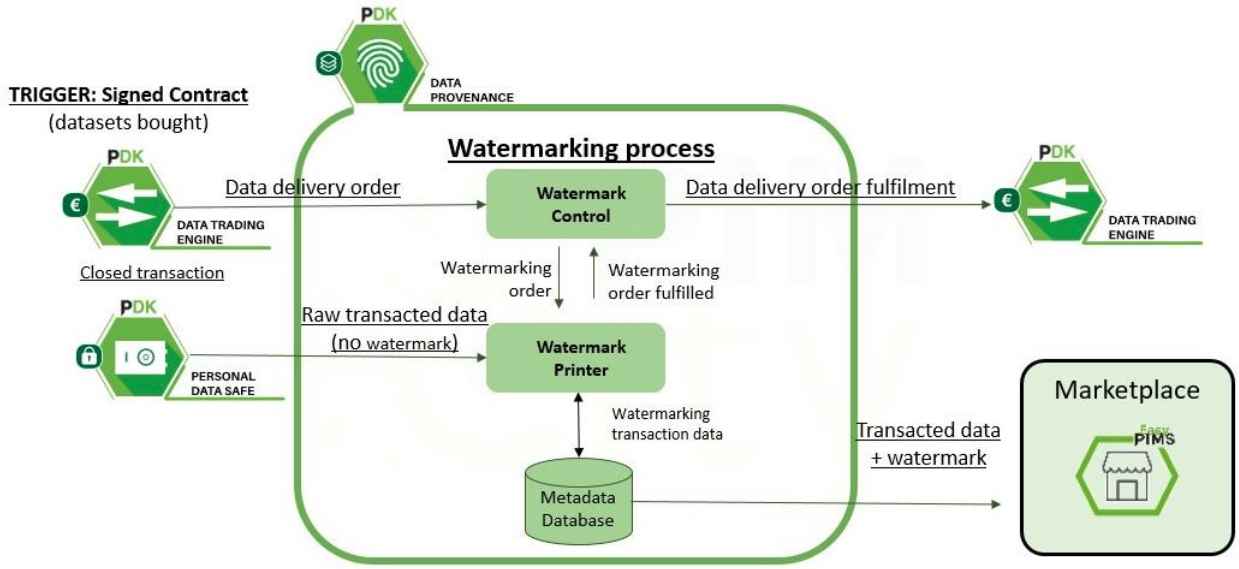

Figure 5: Watermarking Process

5.3.1 Tool/Module Operation

The Data Provenance module will interact with several other modules of the PIMCity platform, namely the D-TE and P-DS. The PIMS Marketplace will be responsible for interfacing with the DP module to deliver data to the data buyers once the transaction is fulfilled by the D-TE.

The DP module will be watermarking data, initially as a centralized component. The first step towards implementing the Watermarking module is following the service architecture as in the previous Figure 5 of the module architectural design. The submodules are supported by an immutable database for historical transaction requests coming from the DP module. We should consider a key manager at the PD-S or DP side to manage watermarking with security and privacy guarantees when possible.

Overall, the DP module will operate as a black box system that provides support to watermark data and return it to the Data Marketplace once the transaction at the D-TE is completed, thus storing an historical log of transactions in the system and counting how many times a dataset has been purchased, for example.

A URL dataset use case:

https://www.ccn.com
https://www.facebook.com
https://www.nytimes.com/2020/10/08/us/politics
https://www.networks.imdea.org/research/projects
https://www.pimcity-h2020.eu/
https://www.instagram.com/imdea_networks

Table 1: URL example list

We explore a use case for datasets we have based on list of URLs that users post or visit. In the real-world Big Data market, lists of visited URLs are valuable information (see Table 1), which are commonly purchased by data buyers for commercial purposes from private data sources (e.g., ComScore). Such purpose serves different business goals (e.g., ad placement for reaching audiences, demographics, sociological studies). Click streams themselves do not contain any proof of data ownership. Therefore, the data owners cannot detect if their click streams are leaked by data buyers without their permission. Existing watermark solutions in the literature cannot be applied easily to click streams because click streams including URLs do not tolerate any distortion. To overcome this problem, we will introduce a new watermarking technique. The general idea is that our solution plays with characters at the URL level by capitalizing some characters as the URL must still be valid. However, deciding which characters to capitalize, our algorithm will generate a random set of bits of the same length of the URL in question. We explain in detail how our watermarking technique works in the following, Algorithm 2 and 3 (Figure 6 and 7 respectively):

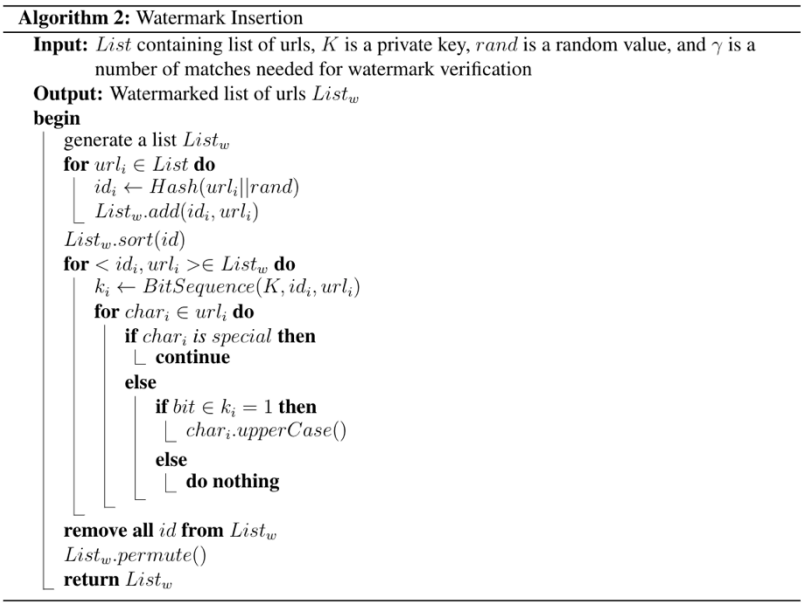

Figure 6: URL Watermark insertion

Watermark Insertion (Algorithm 2): A data owner holds a list of URLs that visited for a period of time (e.g., one week of URL browser history) and inserts a watermark on the list so that the owner can claim ownership in case of illegal distribution. Adding a watermark on the list of URLs should change as little as possible because data distortion is not always tolerable here. In a nutshell, the owner capitalizes some of chars of the URL in a random way. To decide which chars should be capitalized, for each URL, a random bit sequence of the same size of the URL is generated. If the corresponding bit is 1, then the char is capitalized; otherwise, it is left as it is. For example, considering a URL and assuming the bit sequence is *101100110101001010*, a watermarked URL is generated as *HtTPs://wWW.nYtImeS.cOm*. The watermarked URL is still a valid URL and useful. In details, watermark insertion algorithm works as follows: *https://www.nytimes.com* and assuming the bit sequence is *101100110101001010*, a watermarked URL is generated as *HtTPs://wWW.nYtImeS.cOm*. The watermarked URL is still a valid URL and useful. In details, watermark insertion algorithm works as follows:

- The owner generates a private key *K*, a random value *rand*, and *γ* which is a minimum number of matches needed for watermark detection.

  Remark: These three values are private and only known by the owner.
- Generates a list $List_w$ with the same size of URL list *List*.

  Remark: $List_w$ is the watermarked version of *List*.
- For each url_i in the list, a random *id* is generated as $id_i \leftarrow Hash(url_i\ rand)$, and $< id_i, url_i >$ is inserted into $List_w$.

  Remark: because the hash is randomized using the *rand*, it makes hard for an attacker to guess id_i ; *rand* is a random scalar the client can use for the list of URLs. A problem with this approach is that if there are two (or more) identical URLs then ids will be the same. We can resolve this problem by generating a set of random scalar values with the size of $List_w$ as $\{rand_i\}_{i=1,\ldots,List.size()}$. Now, we would have a random value for each URL; hence generated *id* would be different (because of collision resistant hash function) even if any URL pair in the list $List_w$ have identical values.
- Then, we sort the $List_w$ by *id* as $List_w.sort(id)$ to prevent guessing attacks on the position of each of the tuples with the hashed URL and URL pair, namely (id_i, url_i), sorting by id. However, since for marking our scheme exposes the url_i to the data buyer when he obtains the dataset, a malicious buyer could observe positions of a new URL in the sorted list.
- *BitSequence* generates a set of bits with the same size of url_i (excluding special chars) as $k_i \leftarrow BitSequence(K, id_i, url_i)$.

  Remark: Capitalization of chars in a url_i should be random. *BitSequence* (pseudorandom bit

sequence) generates a bit sequence which includes 0s and 1s. *BitSequence* is deterministic meaning that it generates the same output with the same inputs. However, it is computationally hard to find inputs given output. This prevents an attacker to generate a forged watermark (see security discussion for further details.)

- A char is capitalized if the corresponding bit in the bit sequence is 1. If the bit is 0 then char is left as it is.
- *id*s are removed from $List_w$.
- $List_w$ is permuted and returns $List_w$.

Algorithm 3: Watermark Detection

Input: $List'$ containing list of suspected URLs, K is a private key, $rand$ is a random value, and γ Minimum number of correctly marked URL needed for verification
Output: Watermark detected or not
begin
  generate a list $List_s$
  $count = 0$
  for $url_i \in List'$ **do**
    $allLowerCase(url_i)$
    $id_i \leftarrow Hash(url_i||rand)$
    $List_s.add(id_i, url_i)$
  $List_s.sort(id)$
  for $< id_i, url_i > \in List_s$ **do**
    $k_i \leftarrow BitSequence(K, id_i, url_i)$
    $k_i'.sizeOf(k_i)$
    for $char_i \in url_i$ **do**
      if $char_i$ *is special* **then**
        continue
      else
        if $char_i$ *is* $UpperCase$ **then**
          $k_i'.append(1)$
        else
          $k_i'.append(0)$
    if $k_i == k_i'$ **then**
      $count++$
  if $count \geq \gamma$ **then**
    detected

Figure 7: URL Watermark detection

Watermark Detection (see Algorithm 3): A data owner suspects a list of URLs (*List'*) is the dataset he/she sold. In order to detect if the suspected list belongs to the true data owner or not, a watermark detection algorithm is presented. In a nutshell, the owner regenerates bit sequences from a suspected (watermarked) list by checking upper and lower cases in a URL. She later compares it with bit sequences computed from watermark key *K*, *id*, and *url*. The owner starts counting matches between bit sequences. If the number of matches exceeds the certain threshold then the algorithm returns detected. For illustration, assume the owner wants to know if *HtTPs://wWW.nYtImeS.cOm* belongs to her. The watermark detection algorithm first retrieves the bit sequence used while inserting the watermark. The algorithm appends 1 when it sees uppercase char, 0 otherwise, resulting in the bit sequence: *101100110101001010.* Later, it computes a *BitSequence* function as $BitSequence(K, id_i, url_i)$. If *K*, *id*, and *url* are the same inputs while computing insertion algorithm then the output of bit sequence function should be the same. The output of the bit sequence function is compared with the bit sequence reconstructed from the URL (nytimes.com). If they are equal then the watermarked URL belongs to the owner.

For example, for *HtTPs://wWW.nYtImeS.cOm*, the bit sequence is *101100110101001010.* Later, it computes the bit sequence by running the bit sequence . If *K, id*, and *url* are the same inputs while computing the insertion algorithm then the output of bit sequence function should be deterministic. The output of bit sequence function is compared with the bit sequence reconstructed from the URL (nytimes.com). If they are equal, the watermarked URL belongs to the owner. In details, watermark detection algorithm works as follows;

- Generates a list $List_s$ with the same size of *List'* (url list) and initializes counter as 0.
- Computes lowercase version of each URL (*allLowerCase(url)), id (id←Hash(url_i,rand)*), and adds it to the $List_s$.
- Sort the $List_s$ by id as $List_s$.sort(id)

- For each URL in Lists, a bit sequence k_i is generated a set of bits with the same size of url_i (excluding special chars) as $k_i \leftarrow BitSequence(K, id_i, url_i)$.
- $k_{i'}$ is generated with the same size of k_i in line $k_{i'}$.SizeOf(k_i).
  Remark: k'_i is generated from the watermarked URL from the suspected list. k'_i should be the same as k_i if the watermark is a valid one (see security discussion for further details).
- For each selected URL url_i, checks if a char is lowercase or uppercase (omits special chars as in the insertion algorithm). If the char is uppercase then 1 is appended to k'_i, otherwise 0 is added.
- Checks if k'_i and k_i are identical. If they are, it increases the counter by one.
- After repeating previous steps on each URL, the counter is checked if it is greater than or equal to the threshold γ. If it is, then it returns **detected**.
  Remark: The threshold γ and counter can be omitted in case the owner wants to detect if a watermarked URL belongs to her. This approach secures our technique against subset-attacks (see security analysis for more details).

Algorithms 2 and 3 (Figure 6 and 7) for URL watermarking differ from the state-of-the-art Algorithm 1 in the fact that the latter applies to datasets with numeric values only as seen in the VLDB work. In the VLDB work, they have to skip any other data types at the time of choosing potential candidate attributes for insertion of watermarks. Therefore, in Algorithm 1 (Figure 4) they are unable to watermark a string of characters. We propose a new method to overcome this limitation here in a strawman version of our preliminary protocol that we are developing at IMDEA Networks to watermark URLs, which is presented in Algorithms 2 and 3 (Figure 6 and 7). Naturally, we will address some of the security issues and limitations we encounter in the above strawman approach by extending the protocol using for instance a robust and/or signed watermarking version of it to overcome attacks that can easily remove the watermark in batch by converting all URLs to lowercase.. Other techniques may be combined on demand with watermarking as needed with the goal of publicly and securely validating data ownership as well as tracing data buyers during the construction of the watermark previous to any data release into a marketplace. The goal as explained is to apply this watermark on the datasets that data buyers requests before being obtained through the PIMCity marketplace.

5.3.2 Tool/Module Interfacing

This section describes the necessary technical resources to implement for the first prototype of the Watermarking module in PIMCity.

Data Watermarking Control: The Watermarking Control will handle the incoming requests as they arrive sequentially. In order to make sure the DP module includes the corresponding watermark we will invoke here the Watermarking Printer.

Data Watermarking Printer: In order to fulfil data orders from the D-TE, the DP module will insert a watermark before the dataset delivery to the Data Marketplace. However, the Data Marketplace must provide the necessary API endpoint or means to send this data. The DP module will not store the data locally, just transactions metadata in the immutable database. D-TE will invoke this module before any piece of data becomes available to the Data Marketplace and thus the buyer downloads any piece of data from the system after this process is completed.

Immutable Database: this database holds an immutable registry of data watermarking transactions in the system. However, the raw datasets will be held in an aggregated form in a centralized component (data lake) or a decentralized one (P-DS) of PIMCity and have gone through anonymization procedures already to ensure privacy and adherence to GDPR requirements as well.

Internal queries to the DP module will be supported by an internal REST API (implemented in WP5) with an output similar to one following in Table 2 below. We have not yet decided whether for verification we will provide another API endpoint, or it will be a decentralized process local to the user and their key as to verify the watermark privately. We do not have any externally facing interfaces to design according to platform requirements in D1.1.

Internal APIs:

- POST
  - Example:

    D-TE module sends to the DP module a request to fulfil a transaction (possibly including references to the row identifiers that makes up the data requested), so that the DP module proceeds to fulfil the watermarking requirements for it or simply retrieve the confirmation that the data is already watermarked.

    DP responds with:
    - A 200-Ok response with a body indicating a unique *id* for the transaction requested by the D-TE. The id is used for watermarking of the chosen piece of data is being processed correctly.
    - A 409-Conflict response: Indicates that the request could not be processed because this request has been already scheduled for processing.

- GET: this endpoint requires the *id* as parameter key with a value to identify the watermarking unique transaction id initiated by the D-TE to the DP in the previous POST request/endpoint.
  - Example: D-TE obtains the metadata of the transaction being watermarked as specifies the *id as* parameter from the returned body of the POST above.
  - Returns a 200-Ok response with a JSON formatted body file indicating results as follows:

```
{
results:
        [
        {id: “8d7b0f47-55a7-4795-8ec4-2d33af2bf0e5”,
        ‘createdAt’: “2020-09-30T12:39:22Z”,
        ‘updatedAt’: “”,
        ‘status’: “WATERMARK PROCESSED”,
        ‘transactionReport’: “NO ERRORS”}
        ]
}
```

Table 2: API output

6. User Profiling System

6.1. Tool/Module Objective

The user profiling system is a module developed under the umbrella of the Data Knowledge Extraction (DKE) component. This component is the means to extract knowledge from the raw data. One of the biggest challenges in big data and machine learning is the creation of value out of the raw data. When dealing with personal data, this must be coupled with privacy preserving approaches, so that only the necessary data are disclosed, and the data owner keeps the control on them. The DKE consists of machine learning approaches to aggregate data, abstract models to predict future data (e.g., predict user's interests in recommendation systems), fuse data coming from different sources to derive generic suggestions (e.g., to support decision by users, providing suggestions based on decisions taken by users with similar interests).

In the case of the User Profiling system, the final goal is to create meaningful user profiles that can be used for companies involved in the online advertising ecosystem. To this end, the profiles will be generated using the taxonomy defined by the IAB[17]. Moreover, the system will be able to incorporate different data sources to create a profile as comprehensive and representative of the user as possible.

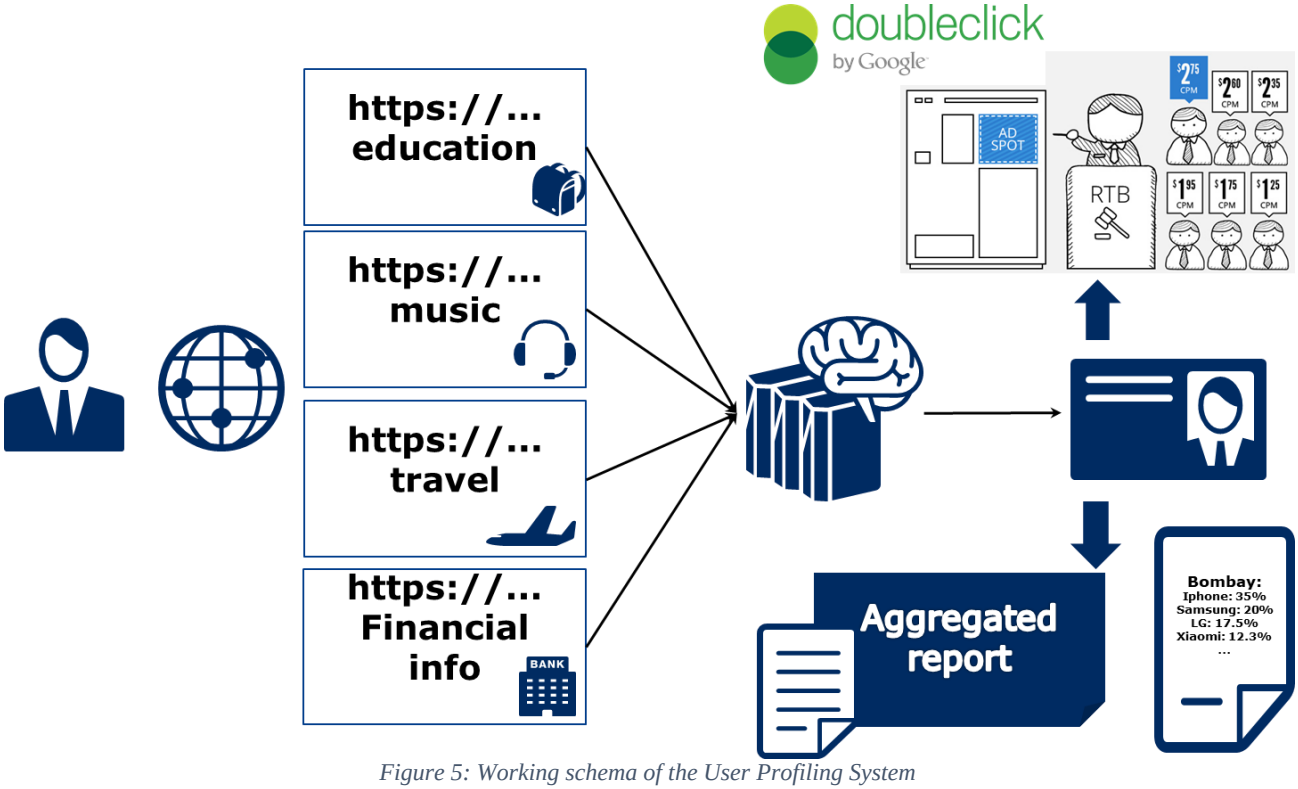

Figure 5: Working schema of the User Profiling System

The figure above (Figure 6) presents the idea of the whole module that can be used to generate individual profiles, for example, for its uses in the RTB ecosystem or can be used to generate reports for specific audiences.

The system is designed to work in a wide variety of scenarios, allowing the development of multiple use cases (by adding different analysis algorithms) and the addition of custom data sources. However, during the PIMCity project, we will develop those connectors and algorithms that use the network browsing history of the users as input to generate a profile for the online advertising ecosystem. This component will be a key piece of the Personal Data Avatar and will be integrated into the EasyPIMS platform.

[17] https://www.iab.com/guidelines/content-taxonomy/

6.2. Background and State of the Art

Targeted online advertising is a multi-billion-dollar business based on the ability of profiling and delivering targeted ads to a wide range of users. Due to the privacy erosion associated with such business, researchers are trying to understand how profiling works and anti-tracking applications are becoming popular among users. Both research and privacy-enhancing apps, however, target ad-networks or over-the-top providers that have unrestricted access to users' online activity. There seems to be little interest in potential profiling activities by “network observers” like ISPs or VPN providers. On the one side, this may be explained by the pervasiveness of TLS that secures connections end-to-end. On the other side, TLS does leak some information, and it is not clear what an eavesdropper can learn about a user, despite her traffic being encrypted.

Some previous studies (mainly carried out by members of the PIMCity consortium (Gonzalez, Soriente, & Laoutaris, 2016)) have demonstrated the possibility of obtaining user profiles from network data in a comparable fashion to the profiles generated by OTT providers. However, the collection of the data from telecom operators is problematic, since even when it may be legal under new laws like the GDPR, it may damage the image of the companies. We expect PIMCity, and, in particular, the EasyPIMS platform to solve this problem by directly involving the user in the profiling system.

The main component of this module is the Net2Vec technology (Gonzalez, et al., 2017) developed by NEC. This technology is a flexible high-performance platform that allows the execution of deep learning algorithms in the communication network. Net2Vec is able to capture data from the network at more than 60Gbps, transform it into meaningful tuples and apply predictions over the tuples in real time. This platform can be used for different purposes ranging from traffic classification to network performance analysis.

The Net2Vec technology has been already tested in different use cases in network operators. Among those, we highlight its use to detect dangerous behavior from final users (Siracusano, Trevisan, Gonzalez, & Bifulco, 2019) or to predict the churn of users among companies.

6.3. Tool/Module Technical Design

The main objective of this tool is to provide a framework that allows the analysis of different types of data to obtain user profiles (or to extract other knowledge from stream data). Network traffic consists of high-speed packet sequences (or data derived from them, such as log files) carrying high-level information such as sequences of text, images, speech and so on. State of the art (deep) learning algorithms have shown remarkable performance for analytics tasks on such sequences of objects, and so this tool aims to bring these state-of-the-art ML methods to bear on communication networks.

At a high level, this tool will be in charge of capturing packet data, filtering it, constructing tuples from it, and feeding those tuples to the machine learning algorithms in charge of the analysis. More specifically, the architecture will consist of a set of five components, each containing pluggable modules (see Figure 9):

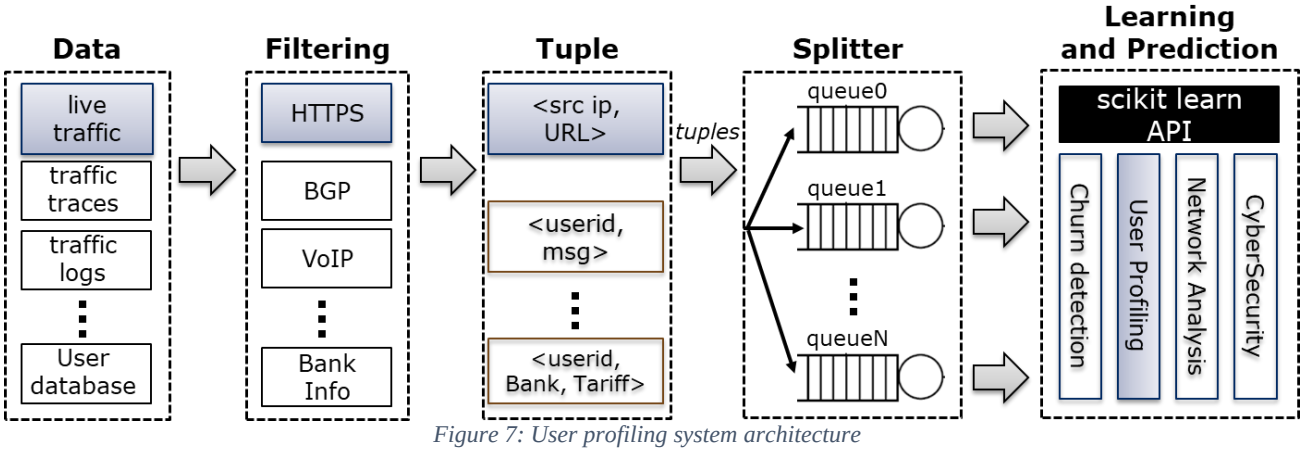

Figure 7: User profiling system architecture

Data Capture: This component is in charge of capturing network data from a number of sources including network interfaces, traffic trace files or log files. For the demonstration of this tool in the PIMCity project, we will use two sources: 1) traffic logs coming from the network operators participating in the project and 2) a browser plugin developed as part of the Data Portability task that will collect the hosts visited by the users.

Filtering: A set of modules responsible for filtering out uninteresting traffic. This component is optional: depending on the data capture module, it may be that all traffic is relevant. In our evaluation we implement a module that filters for HTTP traffic.

Tuple Generation: This component transforms the incoming packet data stream into a set of tuples suitable for the representation learning component. For example, in the user profiling use case we will demonstrate, the tuples are of the form <src ip, hostname>.

Splitter: This component takes, as input from users, a subset of attributes and uses it to split the incoming tuple stream into separate sequences of tuples. Analogous to the relational database terminology, we refer to these subsets of attributes as keys. The individual sequences become the input to the representation learning methods.

Representation Learning and Prediction: This component supports any algorithm that complies with the basic methods of the scikit-learn[18] API. In addition to the standard ML algorithms, most deep learning frameworks have wrappers that expose these methods. The input to the ML models is fixed size (possibly padded) sequences of tuples, one per key value.

Users of the platform can implement data analysis use cases by choosing modules for the different components described above, along with a specification of a key to be used by the splitter. Extending the platform's functionality can be done by developing additional modules. In the rest of this section, we give a more detailed explanation of the various components, as well as a performance evaluation of the data capture, filtering and tuple generation components.

6.3.1 Tool/Module Operation

For the user profiling use case, we assume a network operator (or other actor with access to the network traffic) capturing customer HTTP(S) traffic. The HTTP(S) requests are captured by Net2Vec and mapped into tuples of the form (src ip/IMSI, hostname). Each tuple represents a hostname visited by a user in the network.

In addition, the operator maintains a set of product-related categories C and a subset of the hostnames is associated with some categories from C. Note that the number of hostnames for which categories are known is small compared to all known hostnames since many hostnames do not correspond to webpages but to CDNs, trackers, and mobile application APIs. The objective is now to assign, in real-time, a given user (here: IP address or IMSI) to a set of product-related categories based on her current URL request sequence.

The standard approach would be to follow a strategy similar to the one used by online trackers, where all hostnames visited are stored and used to assign categories to users. This approach, however, has several drawbacks. First, it does not scale well since the amount of data per user grows continually. Second, storing the browsing history of users raises important difficult issues and is even illegal in several countries.

[18] https://scikit-learn.org/stable/

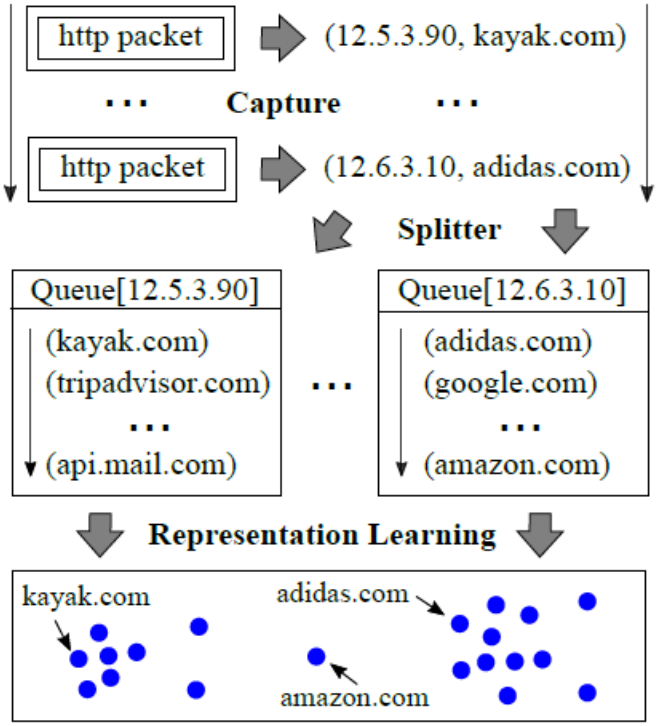

Figure 8: Illustration of the proposed framework using the user profiling system.

To overcome these problems, we will use the User Profiling System to train a neural network model that learns the behavior of users. It is then deployed to predict, given a request sequence of a user, the product-related categories the user is likely to be interested in. The neural network learns a representation of the hostnames from a large number of request sequences. For each input sequence, the model is trained to reconstruct one-hot encoding of the hostname from the rest of the sequence. This ML model is similar to the word2vec model for learning word embeddings (Mikolov, Chen, Corrado, & Dean, 2013) Figure 10 depicts the Net2Vec system for user profiling. This way, the model is able to infer the categories associated to unknown hostnames.

The module requires to be periodically trained using the data available in the system (from previous days). Then, it may be queried in a very efficient way when a profile should be generated.

6.3.2 Tool/Module Interfacing

The User Profiling System may interact with different external components, both from the PDK and from external data sources. Among those, we find:

- The component generating the data input: this component may provide access to the browsing data of the users. In a production environment it would be the physical connection to the network. For the demonstration of PIMCity (and the EasyPIMS platform) we will develop a plugin for the Chrome browser that will collect the data. This component will be developed as part of the Data Portability task. This is the main input of the system.
- The categories for the websites: a list of hostnames with associated categories is required. We will use a set of categories obtained from the Google Adwords system that uses the categories defined by the IAB. This input is required and could be adapted to different use cases by only changing the ontology used.
- The user dashboard: The User Profiling System will generate a profile that can be presented to the user. It will allow the user to modify the profile or to block parts of it (i.e., a user whose profile indicates and interest in medical treatments may want to block that part when the profile is sold to third parties).

Figure 11 shows the relations between the different components of the PDK with the User Profiling System.

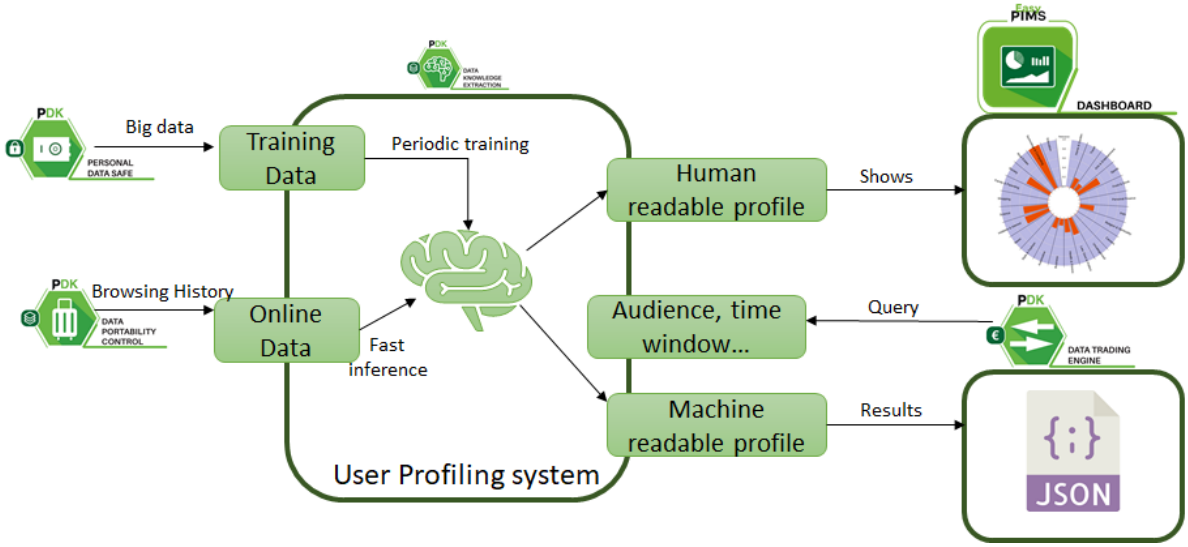

Figure 9: User Profiling System

7. Quantified Self

7.1. Tool/Module Objective

The purpose of the Quantified-self dashboard is to present to the user the benefits of sharing data in PIMCity. Self-quantified, also known as self-tracking, is a way of measuring and monitoring elements of someone's body and life as a way of self-improvement or self-reflection since ancient times (Lupton, 2016). In PIMCity, users choose to upload data to the PIMCity platform because they obtain a financial benefit, unlike in the current landscape of wearables, IoT sensors, mobile sensors, which report data about the individual without any explicit reward presented to the user, beyond self-improvement as a service. Therefore, in PIMCity Quantified-Self we rely on the consent and the data being already anonymized for each individual when it comes to be displayed at the dashboard. That is, a user in the Quantified-Self dashboard can only see data at the aggregated user level when it is about his/her own profile, the rest must be displayed to him/her at a global level among all users with no specific characteristics that allow users re-identification.

7.2. Background and State of the Art

The consequence of the history of self-tracking has led us to a point where users are constantly generating data about their trajectories, transactions, personal preferences and the like, without even having access to the data generated in the first place. Effectively, this is given away freely to untrusted digital applications with little or no privacy considerations from the applications. In the literature, we find some self-tracking techniques that usually collect information for one or two dimensions of the activity of a user. "Personal informatics and personal analytics are terms that are used most often in academic literature on human–computer interaction" according to "The Quantified Self" (Lupton, The Quantified Self, 2016) . However, there are other approaches that collect many more traces of user behavior and for longer periods of time. For instance, telecom operators can identify users' mobile phones just because they are their mobile carrier. This includes track their phone identity numbers as the phone IMEI number, triangulate latitude, longitude through cell towers and connections to WiFi networks (Razaghpanah, et al., 2018). There are examples of the latter and research has shown the privacy issues that this approach creates, sometimes inadvertently for the user as reported by tools as the *Lumen Privacy Monitor* (Vallina-Rodriguez, 2017). Besides, self-tracking is encouraged[19] or even enforced in the case of mobile applications in authoritarian regimes[20], this gets worse, despite of technology not working in all contexts[21]. However, in this Quantified-Self dashboard we rely on the sociological perspective and assumption from the book of Lupton in that the user is aware of this self-tracking, knowingly and purposely collecting such information to upload to PIMCity. Therefore, "Self-tracking differs from covert surveillance or means of collecting information on people that result in data sets to which the subjects of monitoring do not have access" -- and where privacy is not considered by design.

Our Dashboard visualization of our Quantified-Self shall allow for private and transparent data analytics of one's personal data. Related to that, User Privacy Preferences are being explored in recent and ongoing EU H2020 projects[22]. At a first glance in our Dashboard, each user would just choose to grant (or not) access to certain types of data from a taxonomy and we would just want to give each individual user the ability to visualize access to their data (our Quantified-Self dashboard) according to those choices, plus naturally avoiding a view of the rest of the user population in the system by design of the platform itself. Therefore, only in case of needing to display global metrics, we would propose to use aggregated measurements for the dashboard in the Quantified-Self. This is because there may still exist a risk for visualizations displaying far too much granular information about the whole user population in the entire system. For example, even if we show a heatmap of

[19] Singapore already planning version 2.0 contact-tracing wearable: https://www.theregister.com/2020/06/17/singapore_contact_tracing_wearable_2

[20] Contact Tracing Part of Daily Life for Chinese: https://www.racmonitor.com/contact-tracing-part-of-daily-life-for-chinese

[21] Google and Apple's Contact-Tracing API Doesn't Work on Public Transport, Study Finds: https://www.vice.com/en/article/z3epje/google-and-apples-contact-tracing-api-doesnt-work-on-public-transport-study-finds

[22] PlAtform for PrivAcY preserving data Analytics PAPAYA, H2020-DS-SC7-2017 DS-08-2017: Cybersecurity PPP: Privacy, Data Protection, Digital Identities. Deliverable D3.4 Transparent Privacy preserving Data Analytics (Karlstad University), https://www.papaya-project.eu/node/158 in progress.

aggregate user activity to a specific user of the dashboard, in some areas where the activity of user location data is scarce, in the heatmap a few data points will unequivocally identify a person known to the user for living in that area.

7.3. Tool/Module Technical Design

The Quantified-Self is designed into several screens that showcase its functionality in PIMCity. Depending on the available dataset the user can have various insights of the user data. We aim to use user mobility data from telco provider in order to exploit and present mainly location patterns and behavioral patterns of the user, also in comparison with other users with similar profile (based on age, gender, location, etc.). Following are the main operations that we envisage for this module, along with the related initial mock-up screens.

The user, after logging-in will be able to access 3 main operations that are listed on the left of the screen. The Location patterns, the behavioral patterns and the dataset or taxonomy patterns.

An initial screen will welcome the user and present the available options.

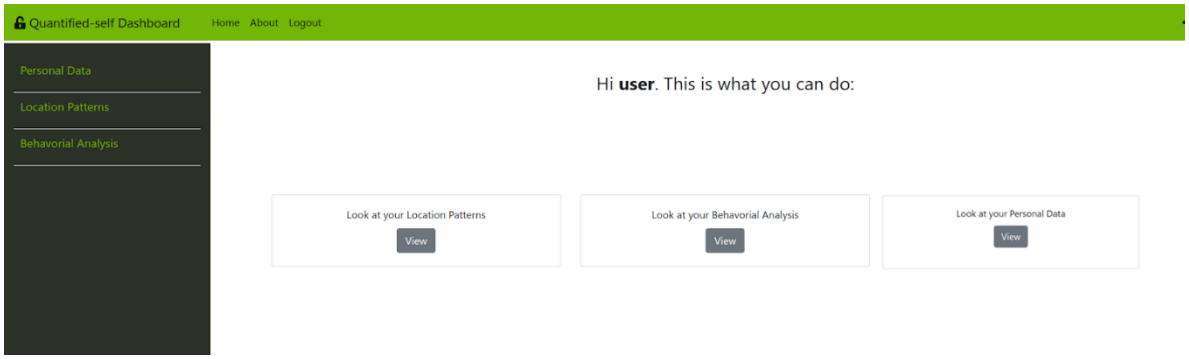

Figure 10: Home screen

The **Locations Patterns** screen is about geo-location patterns the user can see about the contribution he has chosen to make into the system by uploading location data, which he can visualize in a different color as it is his private data (granular). Besides, at the global level he can see an aggregated heatmap of the world/city/town activity.

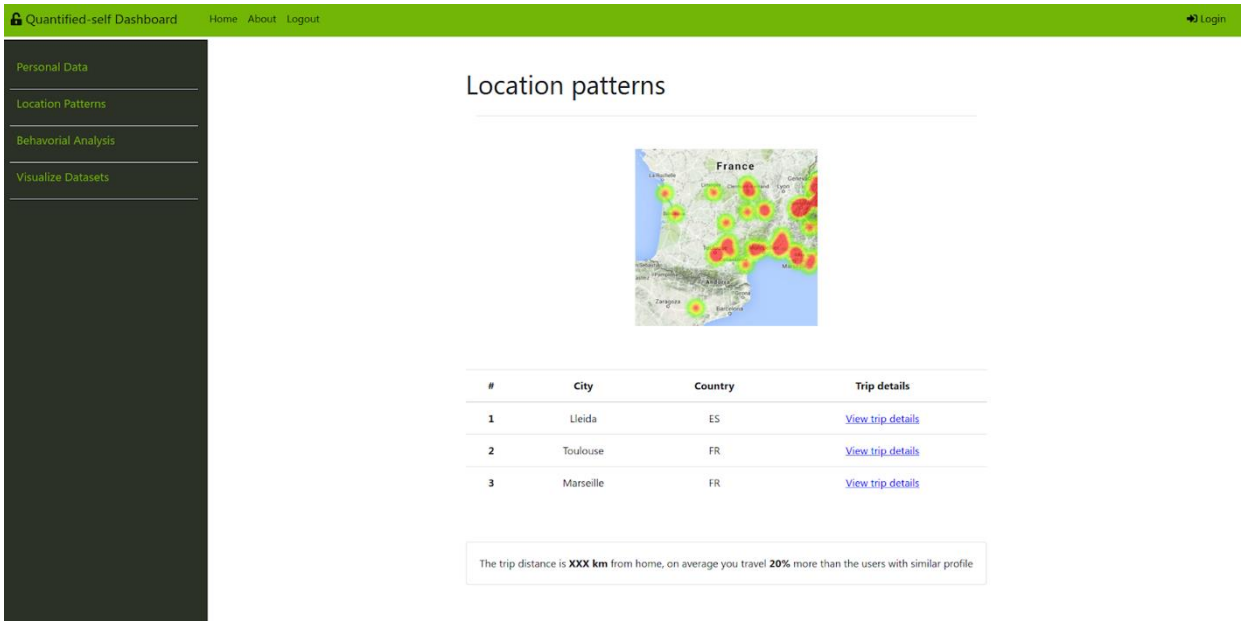

Figure 11: Location Analysis

The **Behavioral Patterns** screen can be based on mobile sensor activity in the system that the user has contributed or in data coming from the use/ location of the mobile device from telco provider. By choosing the “Behavioral Patterns” tab the user can see his data/activity in a bar plot or simple quantitative graph, for example.

Figure 12: Behavioral Analysis

User’s behavioral patterns can be compared with other user’s patterns with similar profile, if available, and a respective chart/table will be provided to the user.

The **Visualize Dataset** screen shows the distribution of contributions to a dataset type (location or behavioral) in the user platform. We may also calculate a global rank among user contributions according to the local distribution of each of them, with local being the pie chart here in darker grey for the type of data.

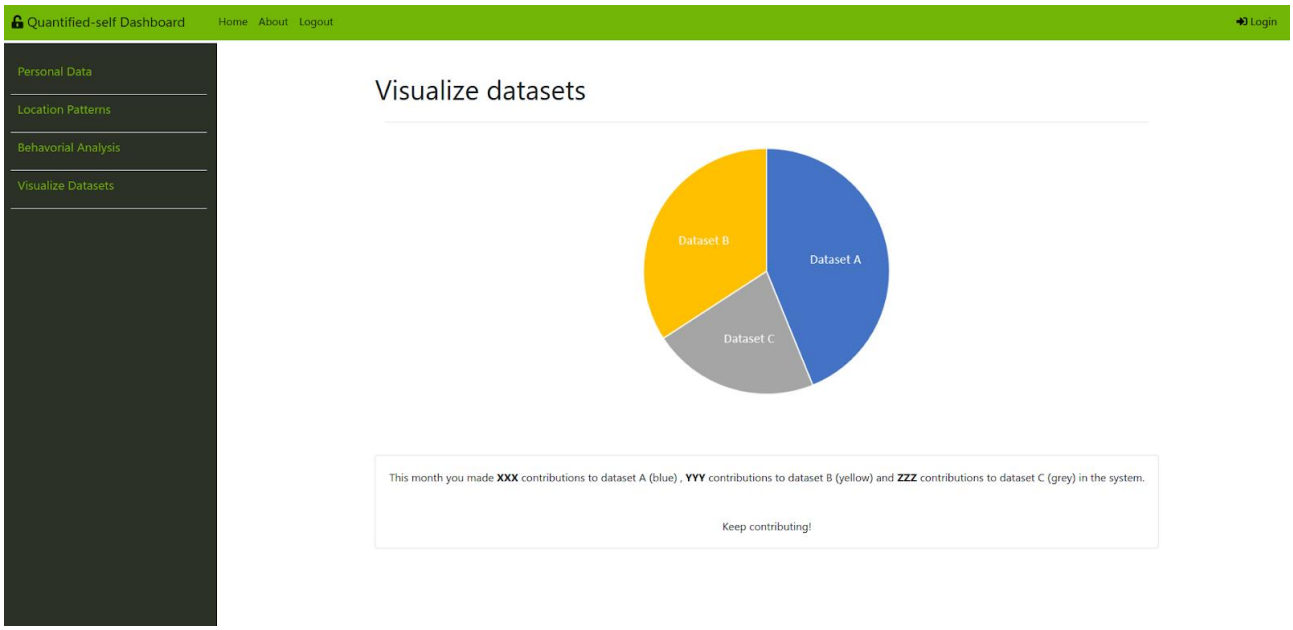

Figure 13: Data visualization screen

7.3.1 Tool/Module Operation

For this task we are leveraging existing technologies as Django and Python. In particular, we leverage a web framework that is based on well-known tools and technologies. We adapt the Django web framework to provide suitable data models in the Quantified-Self, so that it fits into the generic screens designed. This entails modifying all the parts of a web framework based in the MVC (Model, View and Controller[23]) software design pattern. Naturally, MVC use has evolved since its early days of use in SmallTalk[24] objects and now it is widely applied in web frameworks as Django, plus it is widely adopted across several other technologies and frameworks for the WWW (e.g., Rails with Ruby, Spring with Java).

- **Model:** the model represents the data or state and so the business logic of the web application.
- **View:** the view is responsible to render or visualize the model data, so it is about the presentation of the web application to the user.
- **Controller:** the controller is essentially an interface between model and view to update things accordingly in the web application.

In the frontend, we have a layout with buttons and tabs based on the design of the tools in deliverable D2.1[25] for consistency across project dashboards, but we tailor it to our specific requirements for the visualizations that are required in the Quantified-Self screens (Personal, Location, Behavioral and Dataset Visualization tabs). Using MVC and a common graphical design produces common ground for building comprehensive dashboards across the PIMCity project. Also, using a well-accepted common software design pattern for developers and using an accessible programming language, namely Python, benefits the extensibility and acceptability of our Quantified-self dashboard tool. In the next iterations of the Quantified-Self, we plan to change internals of the MVC of the web framework in order to adapt the data **model** to the incoming feeds, the **view** to the above screens and the **controller** specific actions for complete functionality of the Quantified-Self navigation and options available to the end-user. Besides, the framework should be compatible with any type of data storage, should we choose a different database than Mongo (e.g., a time-series database may be more convenient than a MongoDB database in order to calculate daily, weekly, monthly, or yearly average counts of user mobility). Future functionality will be considered according to the D1.1[26] requirements for the Quantified-Self dashboard (e.g., average distance travelled, areas visited, etc.). The reference architecture for the Quantified-Self framework is shown below in Figure 14. Note: we assume built-in authentication into the Django web framework[27].

[23] Model-View-Controller https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

[24] GUI Architectures, Fowler. M. https://martinfowler.com/eaaDev/uiArchs.html

[25] PIMCity Grant agreement ID: 871370, H2020-ICT-2019-2. Deliverable D2.1 Design of tools to improve user's privacy, WP2. Public M12 - 30/11/2020 https://www.pimcity-h2020.eu/dissemination/deliverables/

[26] PIMCity Grant agreement ID: 871370, H2020-ICT-2019-2, D1.1, PIMCity Requirements and Specifications, https://www.pimcity-h2020.eu/app/uploads/2020/06/D1.1_PIMCity-requirements-and-specifications.pdf

[27] https://docs.djangoproject.com/en/3.1/topics/auth/

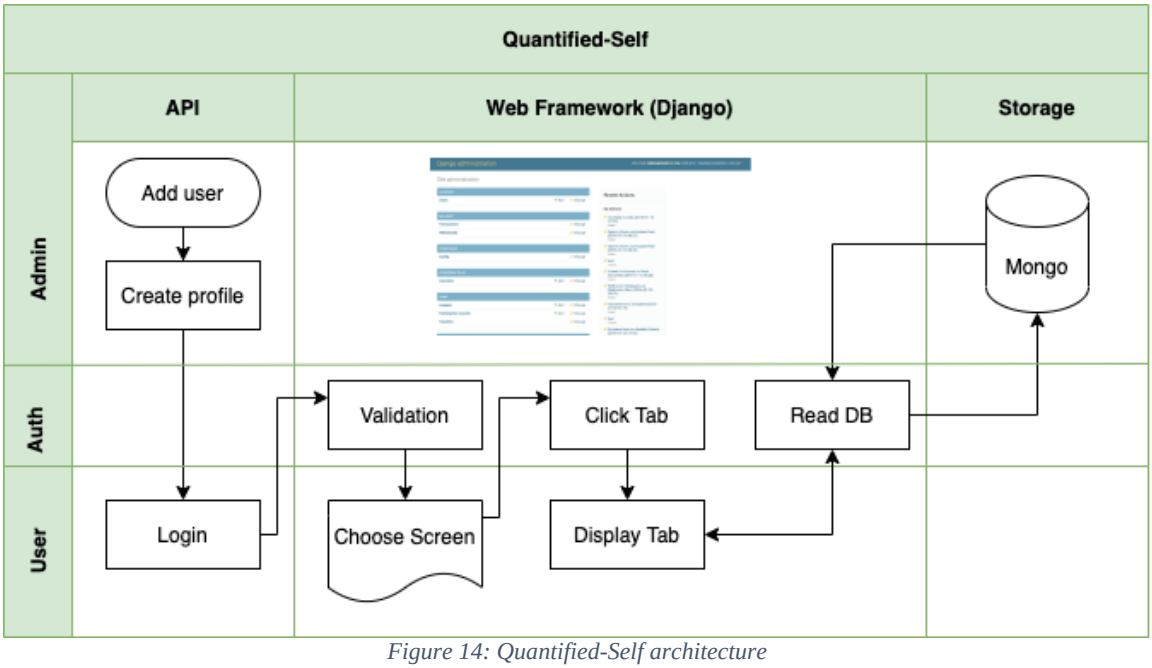

Figure 14: Quantified-Self architecture

7.3.2 Tool/Module Interfacing

This module will be able to retrieve input using the related APIs from

- Individual user data through the P-DS.
- Data Knowledge Extraction module
- User Profiling Data
- Data Aggregation module

In order to combine this information and create the visualizations that will be provided to the user. These visualizations will help the user to better understand their behavioral patterns.

Apart from these visualizations, this module does not provide another output to other modules. The following figure (Figure 15) depicts the data flow related to this module.

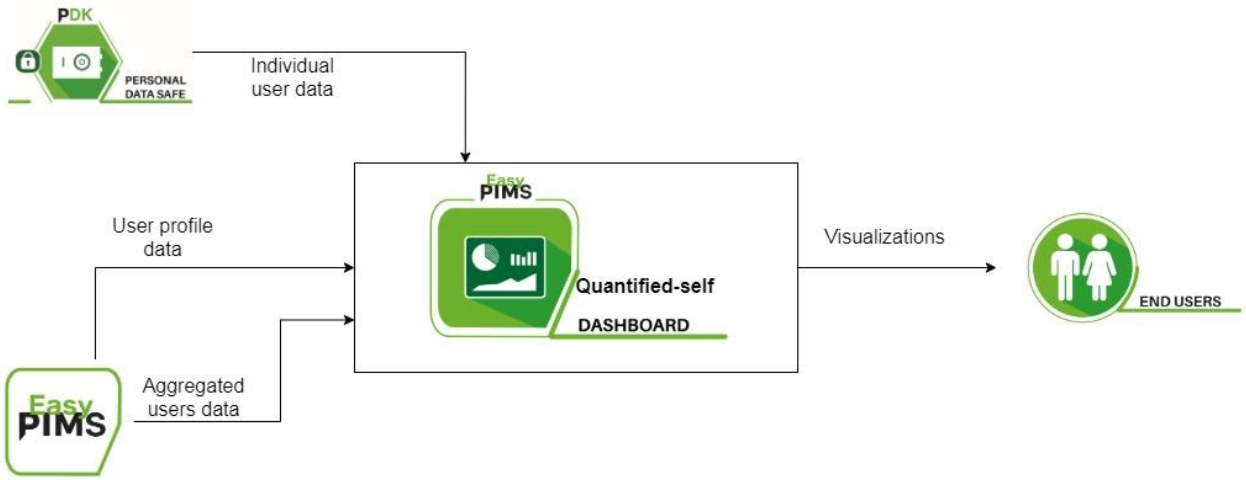

Figure 15: Quantified-self Dashboard module input and output flows

8. CONCLUSION

This document describes the high-level design of the PIMCity modules in charge of improving data management in the PIMCity project, as part of the WP4 deliverable. This deliverable defines the tools for ingesting and aggregating data in a secure and privacy-preserving way, while offering users data portability and data verification features. The tools of which the initial design has been presented in this document are the Data Aggregation (DA), Data Portability Control (DPC), Data Provenance (DP), User Profiling System (UPS) and Quantified Self (QS). Note that both UPS and QS tools are under the umbrella of Data Knowledge Extraction (DKE) component.

The overall architecture of each module has been discussed with all the project partners to allow easy integration in the PIMCity PDK. Then, the partner responsible for each module has finalized the design in cooperation with the other partners of the WP4. Finally, this document has been reviewed by all the WP4 partners and cross-reviews by partners from other WPs.

We refer the reader to the Deliverable 1.1 to have a broader description of the PIMCity project, the other PDK modules and the EasyPIMS architecture. The five tools included in the WP will be implemented in the next coming months, and the final design, as well as the preliminary implementations, will be described in the Deliverable 4.2.

9. REFERENCES

Cavoukian, A. (2011). *Privacy by Design - The 7 Foundational Principles.*

Sweeney, L. (2002). K-anonymity: A Model for Protecting Privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based, vol. 10, no. 5*, (pp. 557-570).

Truta, T. M., & Vina, B. (2006). Privacy Protection: p-Sensitive k-Anonymity Property. *22nd International Conference on Data Engineering Workshops.* Atlanta.

Machanavajjhala, A., Kifer, D., Gehrke, J., & Venkitasubramaniam, M. (2007). L-diversity: Privacy Beyond K-anonymity. *ACM Transactions on Knowledge Discovery from Data, vol. 1, no. 1.*

Li, N., Li, T., & Venkatasubramanian, S. (2007). t-Closeness: Privacy Beyond k-Anonymity and l-Diversity. *IEEE 23rd International Conference on Data Engineering.* Istanbul.

Dwork, C. (2006). Differential Privacy. *33rd international conference on Automata, Languages and Programming - Volume Part II.* Venice.

Gehrke, J., Hay, M., Lui, E., & Pass, R. (2012). Crowd-Blending Privacy. *32nd Annual Cryptology Conference.* Santa Barbara.

Machanavajjhala, A., & Kifer, D. (2015). Designing Statistical Privacy for Your Data. *Communications of the ACM, vol. 58, no. 3*, (pp. 58-67).

Hert, P. D., Papakonstantinou, V., Malgieri, G., Beslay, L., & Sanchez, I. (2018). The right to data portability in the GDPR: Towards user-centric interoperability of digital services. *Computer Law & Security Review, Volume 34, Issue 2.*, 193-203.

Shirazi, M. N., Kuan, H. C., & Dolatabadi, H. (2012). Design Patterns to Enable Data Portability between Clouds' Databases. *12th International Conference on Computational Science and Its Applications*, (pp. 117-120).

Alomari, E., Barnawi, A., & Sakr, S. (2014). CDPort: A Framework of Data Portability in Cloud Platforms. (pp. 126–133). Association for Computing Machinery.

Panah, A., Van Schyndel, R., Sellis, T., & Bertino, E. (2016). On the properties of non-media digital watermarking: a review of state of the art techniques. *IEEE Access 4*, (pp. 2670–2704).

Bianchi, T., & Piva, A. (2013). Secure watermarking for multimedia content protection: A review of its benefits and open issues. *IEEE signal processing magazine*, (pp. 87-96).

Kirovski, D., Malvar, H., & Yacobi, Y. (2002). Multimedia content screening using a dual watermarking and fingerprinting system. *Proceedings of the tenth ACM international conference on Multimedia.*

Kamran, M., & Farooq, M. (2018). *A comprehensive survey of watermarking relational databases research.* arXiv:1801.08271.

Agrawal, R., & Kiernan, J. (2002). Watermarking relational databases." VLDB'02: Proceedings of the 28th International Conference on Very Large Databases. *Morgan Kaufmann*.

Hamadou, A., Sun, X., Gao, L., & Shah, S. A. (2011). A fragile zero-watermarking technique for authentication of relational databases. *International Journal of Digital Content Technology and its Applications, vol. 5, no. 5*, 189-200.

Ergun, F., Kilian, J., & Kumar, R. (1999). A note on the limits of collusion-resistant watermarks. *International Conference on the Theory and Applications of Cryptographic Techniques.* Berlin: Springer.

Berchtold, W., Schäfer, M., & Steinebach, M. (2013). Leakage detection and tracing for databases. Proceedings of the first ACM workshop on Information hiding and multimedia security.

Gonzalez, R., Soriente, C., & Laoutaris, N. (2016). User profiling in the time of https. *Proceedings of the 2016 Internet Measurement Conference.*

Gonzalez, R., Manco, F., Garcia-Duran, A., Mendes, J., Huici, F., Niccolini, S., & Niepert, M. (2017). Net2Vec: Deep learning for the network. *Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, (pp. 13-18).

Siracusano, G., Trevisan, M., Gonzalez, R., & Bifulco, R. (2019). Poster: On the Application of NLP to Discover Relationships between Malicious Network Entities. *ACM SIGSAC Conference on Computer and Communications Security*, (pp. 2641-2643).

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). *Efficient estimation of word representations in vector space.* arXiv:1301.3781.

Lupton, D. (2016). *The quantified self.* John Wiley & Sons.

Razaghpanah, A., Nithyanand, R., Vallina-Rodriguez, N., Sundaresan, S., Allman, M., Kreibich, C., & Gill, P. (2018). *Apps, trackers, privacy, and regulators: A global study of the mobile tracking ecosystem.*

Vallina-Rodriguez, N. (2017). *Illuminating the third party mobile ecosystem with the lumen privacy monitor.*

Pielot, M. (2019). Telefonica Mobile Phone Use Dataset. CRAWDAD.

Lupton, D. (2016). The Quantified Self. John Wiley & Sons.