Deliverable D03.03 Data Mining and Semantic Matching Engine

Providing for the Semantic Lifting and Integration of the distributed Personal Information Sphere

**keywords:** Semantic Lifting; Information Extraction; Data Crawling; Data Mining; Knowledge Integration; Knowledge Discovery;

Date: 31/08/2011
Authors: Simon Scerri (NUIG)
Keith Cortis (NUIG)
Ismael Rivera (NUIG)
Cristina Frà (TI)
Contributors: Mohamed Bourimi (USI)
Rafael Gimenez (BDIGITAL)

Status: Final
Dissemination level: Accompanying Report to Prototype
Nature: Public
Workpackage: WP 3 Semantic Representation, Lifting and Processing of Personal Data in Context
Related Tasks: Task 3.1 Ontology Framework
Task 6.5 Web-service Integration
## Version History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Editor</th>
<th>Change description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>29/07/2011</td>
<td>Simon Scerri (NUIG)</td>
<td>TOC</td>
</tr>
<tr>
<td>2</td>
<td>10/08/2011</td>
<td>Mohamed Bourimi (USI)</td>
<td>TOC Feedback (minor changes)</td>
</tr>
<tr>
<td>3</td>
<td>24/08/2011</td>
<td>Simon Scerri (NUIG)</td>
<td>First Internal Draft</td>
</tr>
<tr>
<td>4</td>
<td>18/04/2012</td>
<td>Simon Scerri (NUIG)</td>
<td>TOC Update</td>
</tr>
<tr>
<td>5</td>
<td>31/05/2012</td>
<td>Simon Scerri (NUIG)</td>
<td>Second Internal Draft</td>
</tr>
<tr>
<td>6</td>
<td>17/10/2012</td>
<td>Simon Scerri (NUIG)</td>
<td>Final Internal Draft</td>
</tr>
<tr>
<td>7</td>
<td>24/10/2012</td>
<td>Mohamed Bourimi (USI)</td>
<td>Final Internal Feedback</td>
</tr>
<tr>
<td>8</td>
<td>25/10/2012</td>
<td>Simon Scerri (NUIG)</td>
<td>Final</td>
</tr>
</tbody>
</table>
Abstract

The di.me project researches technologies that enable the use of personal data in a controlled, trustworthy, and intelligent way. The project develops a di.me platform that incorporates user-control deeply in design: a private service – the di.me userware -- offers a central node in a decentralized network, connecting with distinct identities to other users or external services, e.g. social networking platforms. Intelligent features will guide users e.g. by context-aware advice on trust and privacy, or on how to organise the personal information sphere.

This deliverables describes the Data Mining and Semantic Matching engines, two modules behind the Semantic Lifting and Integration objectives of the project. The various techniques employed for the extraction of heterogenous data from distributed sources – of a structured, semi-structured or unstructured natures – are described; together with the method for their representation against the available domain ontologies, and the techniques thereafter applied in order to determine semantic equivalence between two items that refer to the same object. The purpose of this deliverable is to show how these components have been integrated and how they are used within the digital.me userware.
# Table of Contents

Abstract ................................................................................................................................. 3

Table of Contents .................................................................................................................. 4

1 Introduction .......................................................................................................................... 6

1.1 Purpose of this document ............................................................................................... 6

1.2 Related Workpackages and tasks .................................................................................. 6

1.3 Document Structure ...................................................................................................... 6

2 Objective of the Semantic Lifting Component ................................................................... 7

2.1 Role in the digital.me system architecture .................................................................... 7

2.2 Technical Requirements and Decisions ......................................................................... 7

2.2.1 Overview of existing Data Mining/Crawling Techniques from personal devices .......... 8

2.2.2 Service Account APIs in consideration ...................................................................... 8

2.2.3 Approaches to Semantic Persistence ........................................................................ 9

2.2.4 Identifying Resources in a Distributed Environment ................................................ 10

2.2.5 Targeting non-conventional types of Personal Information ....................................... 10

2.3 Technological achievements .......................................................................................... 11

3 Re-use of Existing Standards ............................................................................................ 12

3.1 OpenSocial ..................................................................................................................... 12

3.1.1 Restrictions/Considerations ........................................................................................ 15

3.1.2 License ........................................................................................................................ 15

3.2 Keyhole Markup Language (KML) ................................................................................ 15

3.2.1 Restrictions/Considerations ........................................................................................ 16

3.2.2 License ........................................................................................................................ 16

3.3 Resource Description Framework in Attributes (RDFa) .................................................. 16

3.3.1 Restrictions/Considerations ........................................................................................ 16

3.3.2 License ........................................................................................................................ 16

3.4 Open Graph ..................................................................................................................... 17

3.4.1 Restrictions/Considerations ........................................................................................ 17

3.4.2 License ........................................................................................................................ 17

3.5 iCalendar (iCal) ............................................................................................................... 17

3.5.1 Restrictions/Considerations ........................................................................................ 17

3.5.2 License ........................................................................................................................ 17

4 Re-use of existing Technology .......................................................................................... 18

4.1 Aperture ......................................................................................................................... 18

4.1.1 Restrictions/Considerations ........................................................................................ 18

4.1.2 License ........................................................................................................................ 18

4.2 RDF2Go .......................................................................................................................... 18

4.2.1 Restrictions/Considerations ........................................................................................ 18

4.2.2 License ........................................................................................................................ 19

4.3 Apache Commons IO ...................................................................................................... 19

4.3.1 Restrictions/Considerations ........................................................................................ 19

4.3.2 License ........................................................................................................................ 19

4.4 Scribe ............................................................................................................................... 19

4.4.1 Restrictions/Considerations ........................................................................................ 19

4.4.2 License ........................................................................................................................ 19

4.5 XSPARQL ........................................................................................................................ 19

4.5.1 Restrictions/Considerations ........................................................................................ 20

4.5.2 License ........................................................................................................................ 20

4.6 JSON-lib .......................................................................................................................... 20

4.6.1 Restrictions/Considerations ........................................................................................ 20

4.6.2 License ........................................................................................................................ 20

4.7 The GATE Framework .................................................................................................... 20

4.7.1 Restrictions/Considerations ........................................................................................ 21

4.7.2 License ........................................................................................................................ 21
## 5 Functional Documentation

### 5.1 Distributed URI Naming Scheme

- Page 22

### 5.2 File Metadata Crawler

- Page 22

### 5.3 Service Integration

- Page 22
  - 5.3.1 OpenSocial Adapter
    - Page 23
  - 5.3.2 KML Adapter
    - Page 23
  - 5.3.3 iCal Adapter
    - Page 23

### 5.4 Data Mining

- Page 23
  - 5.4.1 Discovering PIM entities in unstructured data
    - Page 23
  - 5.4.2 Linking PIM knowledge to external knowledge
    - Page 24
  - 5.4.3 PIM Integration
    - Page 25
  - 5.4.4 Matching Persons behind multiple Profiles
    - Page 26

### 5.5 User Activity Context Extraction

- Page 27
  - 5.5.1 Context Extraction from Personal Applications and Services
    - Page 27
  - 5.5.2 Context Extraction from Device Sensors
    - Page 27
  - 5.5.3 Interpreting Mid-level Context
    - Page 28
  - 5.5.4 Context Extraction from Virtual Sensors
    - Page 28

## 6 References

- Page 30

## 7 Appendix

- Page 31

### 7.1 LinkedIn XSPARQL queries

- Page 31
  - 7.1.1 livepost/@all
    - Page 31
  - 7.1.2 livepost/@me/@all
    - Page 31
  - 7.1.3 person/@me/@all
    - Page 32
  - 7.1.4 profile/@me/@all
    - Page 34

### 7.2 Twitter XSPARQL queries

- Page 36
  - 7.2.1 livepost/@all
    - Page 36
  - 7.2.2 livepost/@me/@all
    - Page 37
  - 7.2.3 person/@me/@all
    - Page 38
  - 7.2.4 profile/@me/@all
    - Page 39

### 7.3 Facebook XSPARQL queries

- Page 40
  - 7.3.1 livepost/@all
    - Page 40
  - 7.3.2 livepost/@me/@all
    - Page 41
  - 7.3.3 person/@me/@all
    - Page 43
  - 7.3.4 profile/@me/@all
    - Page 43

### 7.4 KML XSPARQL query

- Page 46
  - 7.4.1 place/@all
    - Page 46

### 7.5 OpenSocial XSPARQL queries

- Page 46
  - 7.5.1 livepost/@me/@all
    - Page 46
  - 7.5.2 profile/@me/@all
    - Page 47
1 Introduction

1.1 Purpose of this document
This document details the work carried out in relation to Task 3.2 “Data Mining and Semantic Matching Engine”, spanning month 0 to month 24 inclusive. The contributions detail the progress achieved in the project over this period, with respect to the deliverable as outlined in the Description of Work (DOW). Minor variations from the foreseen contributions are also outlined where appropriate.

1.2 Related Workpackages and tasks
This deliverable is a result of the work done in WP3 – “Semantic Representation, Lifting and Processing of Personal Data in Context”, particularly the following tasks:

- Task 3.2 Data Mining and Semantic Mapping Engine
The following are the DOW WP objectives covered by this deliverable:

- Developing a data mining engine for the continuous analysis and semantic representation of personal information stored on the user devices (e.g. documents, calendar entries) and exchanged via electronic communication (e.g. IM, email). The mined data is semantically represented using the Ontology Framework.

- Developing a semantic mapping engine that targets the interpretation and integration of heterogeneous, distributed data as well as metadata in open knowledge resources e.g. community profiles, external enterprise services, external ontologies etc.

In addition, the progress reported in this deliverable is a prerequisite for the following tasks:

- WP3 - Task 3.3 Digital Face Generator
- WP3 - Task 3.4 Ontology-based Trust, Privacy, and Security Modelling
- WP3 - Task 3.5 Multi-Engine Recommender System
- WP5 - Task 5.4 Modelling and Semantic UI

1.3 Document Structure
The document is structured as follows. Section 3 starts by providing the motivation for the work covered by this deliverable, i.e. the objective of the required Semantic Lifting component, in the context of the project’s system architecture. Various technical requirements, based on the objectives, are then discussed together with a list of the technological advancements expected as a result of this deliverable. Section 4 details a list of existing technologies that have been re-used, applied and combined for the realisation of the semantic lifting component. This includes discussions of their original limitations, and our proposed adaptations. Finally Section 5 details the functional descriptions of the implemented or extended sub-components.
2 Objective of the Semantic Lifting Component

2.1 Role in the digital.me system architecture
The Semantic Lifting component embodies modules in two packages on the private service (PS): dime-datamining and dime-semantic (Figure 1). The former contains all the information extraction modules, including personal item and context crawlers, and text analytics techniques such as entity extraction and resolution against the available ontologies. The crawlers act upon data in two major sources: the user’s devices (acting as clients), and online service accounts. The extraction of this data is routed through the dime-communications package in the former and dime-servicegateway in the latter.

Once information is extracted and lifted onto the knowledge models, it is stored in the RDF repository within the dime-semantic package. Here, a semantic matching module then operates in order to determine whether the repository contains multiple reference to the real-world item being represented, in view of the possibility to integrate them at the Personal Information Model (PIM) level.

2.2 Technical Requirements and Decisions
This Section starts off by an overview of existing data mining technology for personal devices, and a discussion of their applicability in digital.me. A number of Service Account APIs are then considered for integration with digital.me, including the mining and matching of personal information available on the remote accounts to the local personal information model. Two other discussions covered in this section concern different approaches to metadata storage, in the context of the requirements of
the pursued architecture design, as well as a discussion in relation to the required distributed URI naming scheme.

### 2.2.1 Overview of existing Data Mining/Crawling Techniques from personal devices

A crawler, or commonly known as Web crawler, is a computer program that browses the World Wide Web in a methodical, automated manner or in an orderly fashion. However, a similar idea may also be applied to personal devices, as desktop search tools do, searching the contents of a user's own computer files, rather than searching the Internet. Beagle++\(^1\), Google Desktop\(^2\), Strigi\(^3\), Recol\(^4\) or Meta Tracker\(^5\) are just a few examples of these tools, designed to find information on the user's PC, including web browser histories, e-mail archives, text documents, sound files, images and video.

Aperture\(^6\) is another framework for extracting and querying full-text content and metadata from various information systems (e.g. file systems, mail boxes) and the file formats (e.g. documents, images, text files) occurring in these systems.

Strigi and Aperture were determined as the two best candidates for digital.me for several reasons. Both of them extract the metadata of the computer resources in RDF, using the Nepomuk vocabularies, such as the NIE (Nepomuk Information Element) ontology (Mylka et al., 2007). They are also open source and platform-independent.

### 2.2.2 Service Account APIs in consideration

Several services and service accounts are being considered for integration within digital.me, as listed in Table 1 below. This table is not meant to be exhaustive, and additional APIs or information provided by them might also be considered. Specific Data Categories being considered for each service account are also provided, together with a link to the API specifications (where available).

<table>
<thead>
<tr>
<th>Source</th>
<th>Data Categories</th>
<th>API Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email &amp; Email Folders (IMAP/POP)</td>
<td>Emails, Folders</td>
<td></td>
</tr>
<tr>
<td>Calendar (iCalendar)</td>
<td>Events, Tasks</td>
<td><a href="http://www.ietf.org/rfc/rfc2445.txt">http://www.ietf.org/rfc/rfc2445.txt</a></td>
</tr>
<tr>
<td>OpenSocial</td>
<td>Person, Group</td>
<td><a href="http://opensocial-resources.googlecode.com/svn/spec/trunk/Social-Data.xml">http://opensocial-resources.googlecode.com/svn/spec/trunk/Social-Data.xml</a></td>
</tr>
<tr>
<td>LinkedIn</td>
<td>People, Groups</td>
<td><a href="https://developer.linkedin.com/rest">https://developer.linkedin.com/rest</a></td>
</tr>
<tr>
<td>Facebook</td>
<td>User, Post, Checkin, Comment, Event, FriendList, Group, Link, Message, Note, Photo, Status, Message, Video</td>
<td><a href="https://developers.facebook.com/docs/reference/api/">https://developers.facebook.com/docs/reference/api/</a></td>
</tr>
<tr>
<td>Twitter</td>
<td>Timelines</td>
<td><a href="https://dev.twitter.com/docs/api">https://dev.twitter.com/docs/api</a></td>
</tr>
</tbody>
</table>

---

1. [http://beagle2.kbs.uni-hannover.de/](http://beagle2.kbs.uni-hannover.de/)
2. [http://googledesktop.blogspot.ie/](http://googledesktop.blogspot.ie/)
<table>
<thead>
<tr>
<th></th>
<th>Digital.me Functions</th>
<th>API Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google+</td>
<td>People, Activities, Comments</td>
<td><a href="https://developers.google.com/+/api/-">https://developers.google.com/+/api/-</a></td>
</tr>
<tr>
<td>Flickr</td>
<td>Activity, Contacts, Favorites, Galleries, People, Photos, Places, Tags</td>
<td><a href="http://www.flickr.com/services/api/">http://www.flickr.com/services/api/</a></td>
</tr>
<tr>
<td>Delicious</td>
<td>Update, Posts, Tags</td>
<td><a href="http://www.delicious.com/developers">http://www.delicious.com/developers</a></td>
</tr>
<tr>
<td>Dropbox</td>
<td>Dropbox accounts, Files and metadata</td>
<td><a href="https://www.dropbox.com/developers/reference/api">https://www.dropbox.com/developers/reference/api</a></td>
</tr>
<tr>
<td>Foursquare</td>
<td>Users, Checkins</td>
<td><a href="https://developer.foursquare.com/docs/">https://developer.foursquare.com/docs/</a></td>
</tr>
<tr>
<td>Doodle</td>
<td>Poll</td>
<td><a href="http://www.doodle.com/about/APIs.html">http://www.doodle.com/about/APIs.html</a></td>
</tr>
<tr>
<td>Keyhole Markup Language (KML)</td>
<td>Placemark</td>
<td><a href="https://developers.google.com/kml/documentation/">https://developers.google.com/kml/documentation/</a></td>
</tr>
<tr>
<td>FitBit</td>
<td>Activities, Sleep, Heart, Friends</td>
<td><a href="https://wiki.fitbit.com/display/API/Fitbit+API">https://wiki.fitbit.com/display/API/Fitbit+API</a></td>
</tr>
</tbody>
</table>

2.2.3 Approaches to Semantic Persistence

Due to the nature of the digital.me system, which relies heavily on RDF metadata for a lot of its functions, the question of where to store the required knowledge arose very early in the lifetime of the project. The storage options considered were two: a) store RDF data on each of the devices, integrate it on a personal service and b) store and integrate RDF data only on the remote PS. Due to the Social Semantic Desktop's nature (which considered only one device), the first option was sufficient. However, due to digital.me’s distributed nature, going for the first option would require integration in a central repository, which would in turn introduce a synchronisation issue. Therefore, it was decided that there shall be only one central repository. For scalability and performance reasons, it was further decided for this central RDF store be hosted on the PS, so as to avoid reduced performance and/or a slow userware on any of the devices (including smartphones).

A second issue arose with the introduction of the multi-versioned data requirement in digital.me. For some of the planned features, including privacy preference monitoring, context recognition, context-driven rule previews, it is required to store different versions of the same information from different times. Thus, different versions of each privacy preference are retained for a specific period of time, in order to give the users more control and security over changes done to their privacy preferences. Similarly, to enable context recognition, earlier context snapshots need to be logged in the system. Thus, unlike the Social Semantic Desktop; it was no longer possible to store all of the user’s information in one named graph, as it would otherwise result in conflicting data. For this reason, the digital.me RDF repository refers to multiple named graphs, some of which contain data which no longer reflects the latest user activity context, privacy preferences, etc. To regulate these
timestamped graphs, the DUHO user history ontology\(^7\) was introduced in D03.01, Section 3.7. The use of multiple named graphs is adopted for other reasons, e.g. for practical issues a separate named graph (a new subclass of nrl:InstanceBase – nie:DataSourceGraph in NIE\(^8\)) contains information retrieved from each of the service accounts. Linking the named graph to the source (through nao:isDataGraphFor in NAO\(^9\)) thus makes it easier to create, update, or remove metadata for items in the separate sources. The PIM graph however still retains its special status, as the RDF graph which glues together information in different source graphs whenever it is integrated (e.g. integrates \(n\) different online account profiles from \(n\) online account graphs, by declaring them as pimo:occurrence/s of a unique pimo:Person instance in the PIM graph).

### 2.2.4 Identifying Resources in a Distributed Environment

Since digital.me targets the retrieval and integration of information from distributed sources, the naming scheme and conventions set out by projects like Nepomuk\(^10\) are not always sufficient. In fact, Nepomuk relied on URIs, which unlike URNs do not provide a globally unique way of identifying a resource, as it was sufficient to have unique resource identifiers on only one device. In contrast, digital.me requires a naming scheme which is more or less independent of the data source (account, device, etc.). For the purpose, the Nepomuk convention needs to be extended in order to support identifiers that are independent of the various source envirorments and can thus be unique. The resulting platform-independent naming scheme is presented in Section 5.1.

### 2.2.5 Targeting non-conventional types of Personal Information

Digital.me targets the representation of both conventional and non-conventional types of personal information. Conventional personal information in the form of files, folders, contacts lists, photo albums, etc., is normally structured or semi-structured, hence more straightforward to extract. On the other hand, non-conventional types of personal information such as a user’s context, histories and system preferences; are normally unstructured and not retrievable in a straightforward way.

One of the new personal information aspects being targeted is user activity context. Users of modern technology encounter recurring situations requiring repeated intervention, some of which are privacy-critical. Last-generation personal devices generate a great deal of context information that is retrievable or interpreted with relatively minor efforts. Social Web activities can also be mined for additional context. We exploit these sources to determine a user’s changing context and recognise recurring user situations. The representation of user activity context is enabled by the DCON Context and DPO Presence Ontologoes (refer to D03.01 Ontology Framework [internal deliverable], Sections 3.4, 3.6).

User system preferences will also be stored in the personal PIM cloud. The Privacy Preference Ontology (PPO) (Sacco & Passant, 2011) provides an access control framework for the Web of Data, regulating access to semantic resources on the Web. Adapted to digital.me, the PPO is used to model access rights between contacts and personal information items stored in the PIM (refer to D03.02 Ontology-based Trust, Privacy, and Security Modeling [internal deliverable], Section 3.1.2). Based on the gathered user activity context, the user can also define and manage context-driven rules for recommendations and automation, through the intelligent userware interface. Rules are represented as instances of the DRMO Rule Management Ontology (refer to D03.05 Multi-Engine Recommender [internal deliverable], Section 3.1). These rules can also regulate context-dependant sharing of personal information with trusted users. More information about the latter will be provided in D03.04 Digital Face Generator.

To store user histories, we utilise the named graph paradigm (Akthar et al., 2008) to store earlier versions of a user's privacy preferences, context logs, etc. as previous PIM subgraphs. Here, time-logged or activity-logged instances of the DUHO ontology are applied to persist personal information from earlier times. Refer to Section 3.7 in D03.01 Ontology Framework [internal deliverable] for further information.

---

10. [http://techbase.kde.org/Projects/Nepomuk/Uris](http://techbase.kde.org/Projects/Nepomuk/Uris)
2.3 Technological achievements

In this deliverable we have laid the groundwork for both the semantic lifting, and integration, of multi-source, heterogenous, personal data of a structured/semi-structured/unstructured nature. The achievements outlined in this deliverable can be summarised as follows:

1. We have enabled the elicitation and aggregation of personal information from multiple online accounts and services, and lifted it onto one unique representation format as provided by the digital.me ontologies. The use of existing technology used by the community (XSPARQL, Section 4.5) has been employed for the purpose of transforming the native heterogenous data into RDF. The sources for which the lifting has been enabled include: a) solutions for Customer Relationship Management (CRM) systems, b) popular social networks such as Facebook, LinkedIn, Twitter and Google+, c) and personal services such as Email services (any IMAP/Pop compliant service), Calendaring Tools (any iCal compliant service) and Health and Lifestyle Monitors (Fitbit\(^{11}\)).

2. We have enabled the elicitation and aggregation of personal information from multiple devices, consisting of legacy data adhering to different native structures and formats. In particular, we have extended existing solutions for personal information crawling (e.g. Aperture, Section 4.1) to cover additional types of information and convert it to RDF. A mobile client crawler supplements the desktop crawler in retrieving further information from mobile devices for the PIM aggregation task.

3. We have enabled the elicitation and aggregation of user activity context from different device and virtual sensors, onto a standard representation as provided by the DCON ontology (D03.01 Ontology Framework, Section 3.6). For the purpose, a number of sensor-data extraction techniques have been provided, each contributing information to a unified representation of user activity context. In addition, we have proposed plausible techniques for enriching context information from implicit sources, such as sharing activities in online social networks. We intend to have at least a partial of the latter virtual sensor extraction techniques implemented by the end of the project lifetime.

4. We have identified best techniques for the integration of duplicate (or more) occurrences of a unique real-life concept, based on text analytics techniques that attempt to identify the semantic equivalence of two separate extracted entities, regardless of the source. Their implementation will extend existing libraries and framework (GATE, Section 4.7), and is mostly targeted for the reconciliation of federated user and contact profiles across multiple local (application) and online (account services) profiles.

5. All the extended, newly-implemented code, modules and packages adhere to existing standards, and are open-source and open for re-use by interested parties.

\(^{11}\) www.fitbit.com/
3 Re-use of Existing Standards

3.1 OpenSocial
Since new social applications are constantly being released, the aim of OpenSocial\(^{12}\) is to eliminate the need of learning several specific social network APIs for developers, and thus use a set of common APIs for building social applications across various websites. Developers can create social applications that will be able to run on social websites that implement the OpenSocial APIs, with standard JavaScript and HTML. The OpenSocial API allows users of web services, mobile devices and desktop applications to interact with social data. Through the use of these APIs users will be able to access their data, information about their friends and also about other people (OpenSocial Specification 2.5.0\(^{13}\)). The major OpenSocial data categories that are currently being considered for digital.me are those of Person\(^{14}\) and Group\(^{15}\). All the fields of both these data categories have been investigated, so as to determine which properties are already defined in the digital.me Ontology Framework, and thus can be semantically mapped to. The property matching investigation of the Person data category is provided in Table 2, whilst the one for the Group data category is provided in Table 3. Each table contains the following details: OpenSocial field name, type and description, together with the corresponding matched digital.me ontology class, property name, data type and description (refer to Deliverable 3.1 [internal deliverable] for a detailed description of the referred ontologies).

<table>
<thead>
<tr>
<th>OpenSocial Field Name</th>
<th>OpenSocial Field Type</th>
<th>OpenSocial Field Description</th>
<th>Ontology Class</th>
<th>Ontology Property Name</th>
<th>Ontology Property Data type</th>
<th>Ontology Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aboutMe</td>
<td>String</td>
<td>Person’s general statement</td>
<td>nco:PersonContact</td>
<td>nco:descrption</td>
<td>rdfs:Literal</td>
<td>description for a Contact</td>
</tr>
<tr>
<td>alternateNames</td>
<td>Plural-Field &lt;Name&gt;</td>
<td>Person’s alternate names</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>formatted</td>
<td>String</td>
<td>full alternate name</td>
<td>nco:PersonaName</td>
<td>nco:nickname</td>
<td>xsd:string</td>
<td>nickname for Contact</td>
</tr>
<tr>
<td>addresses</td>
<td>Plural-Field &lt;Address&gt;</td>
<td>Person’s physical mailing address</td>
<td>nco:PersonContact</td>
<td>nco:hasPostalAddr</td>
<td>nco:PostalAddress</td>
<td>default address for a Contact</td>
</tr>
<tr>
<td>country</td>
<td>String</td>
<td>country name</td>
<td>nco:PostalAddress</td>
<td>nco:country</td>
<td>xsd:string</td>
<td>country name</td>
</tr>
<tr>
<td>formatted</td>
<td>String</td>
<td>full mailing address</td>
<td>nco:PostalAddress</td>
<td>nco:prefLabel</td>
<td>xsd:string</td>
<td>extended part of an address</td>
</tr>
<tr>
<td>latitude</td>
<td>String</td>
<td>latitude of location on a map</td>
<td>nco:PostalAddress</td>
<td>nco:address</td>
<td>geo:Point</td>
<td>geographica l location of a postal address</td>
</tr>
<tr>
<td>locality</td>
<td>String</td>
<td>city or locality</td>
<td>nco:PostalAddress</td>
<td>nco:locality</td>
<td>xsd:string</td>
<td>locality or city</td>
</tr>
<tr>
<td>longitude</td>
<td>String</td>
<td>longitude or location on a map</td>
<td>nco:PostalAddress</td>
<td>nco:address</td>
<td>geo:Point</td>
<td>geographica l location of a postal address</td>
</tr>
<tr>
<td>postalCode</td>
<td>String</td>
<td>Zip code or</td>
<td>nco: nco:postal</td>
<td>xsd:</td>
<td>postal code</td>
<td></td>
</tr>
</tbody>
</table>

\(^{12}\) http://www.opensocial.org
\(^{13}\) http://opensocial-resources.googlecode.com/svn/spec/2.5/OpenSocial-Specification.xml
\(^{14}\) http://opensocial-resources.googlecode.com/svn/spec/trunk/Social-Data.xml#Person
\(^{15}\) http://opensocial-resources.googlecode.com/svn/spec/trunk/Social-Data.xml#Group
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>region</td>
<td>String</td>
<td>state or region</td>
</tr>
<tr>
<td>street Address</td>
<td>String</td>
<td>full street address</td>
</tr>
<tr>
<td>displayName</td>
<td>String</td>
<td>name of Person, suitable for display</td>
</tr>
<tr>
<td>id</td>
<td>Object-Id</td>
<td>Person’s unique identifier</td>
</tr>
<tr>
<td>im</td>
<td>Plural-Field</td>
<td>IM address for Person</td>
</tr>
<tr>
<td>location</td>
<td>String</td>
<td>location of Person</td>
</tr>
<tr>
<td>name</td>
<td>Name</td>
<td>Person’s fully formatted real name</td>
</tr>
<tr>
<td>family Name</td>
<td>String</td>
<td>first name of Person</td>
</tr>
<tr>
<td>givenName</td>
<td>String</td>
<td>first name of Person</td>
</tr>
<tr>
<td>honorific Prefix</td>
<td>String</td>
<td>title of Person</td>
</tr>
<tr>
<td>honorific Suffix</td>
<td>String</td>
<td>suffix of Person</td>
</tr>
<tr>
<td>middle Name</td>
<td>String</td>
<td>middle name of Person</td>
</tr>
<tr>
<td>formatted</td>
<td>String</td>
<td>fullname of Person</td>
</tr>
<tr>
<td>organizations</td>
<td>Plural-Field</td>
<td>Person’s current or past organization affiliation</td>
</tr>
<tr>
<td>department</td>
<td>String</td>
<td>department within organization</td>
</tr>
<tr>
<td>description</td>
<td>String</td>
<td>Person’s role description within organization</td>
</tr>
<tr>
<td>date this</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>endDate</td>
<td>Date or String</td>
<td>Person left organization or role</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>name of organization</td>
</tr>
<tr>
<td>startDate</td>
<td>Date or String</td>
<td>date this Person joined organization or role</td>
</tr>
<tr>
<td>title</td>
<td>String</td>
<td>job title within organization</td>
</tr>
<tr>
<td>phone Numbers</td>
<td>Plural-Field &lt;String&gt;</td>
<td>Person’s phone number</td>
</tr>
<tr>
<td>value</td>
<td>Object</td>
<td>primary value of field</td>
</tr>
<tr>
<td>photos</td>
<td>Plural-Field &lt;String&gt;</td>
<td>URL of person’s photo</td>
</tr>
<tr>
<td>preferred Username</td>
<td>String</td>
<td>preferred username of Person</td>
</tr>
<tr>
<td>profileUrl</td>
<td>String</td>
<td>Person’s profile URL</td>
</tr>
<tr>
<td>status</td>
<td>String</td>
<td>Person’s status, headline or shoutout</td>
</tr>
<tr>
<td>thumbnailUrl</td>
<td>String</td>
<td>Person’s thumbnail URL</td>
</tr>
<tr>
<td>urls</td>
<td>Plural-Field &lt;string&gt;</td>
<td>URL of a web page relating to Person</td>
</tr>
<tr>
<td>value</td>
<td>Object</td>
<td>primary value of field</td>
</tr>
<tr>
<td>birthday</td>
<td>Date</td>
<td>birthday of Person</td>
</tr>
<tr>
<td>gender</td>
<td>String</td>
<td>gender of Person</td>
</tr>
<tr>
<td>interests</td>
<td>Plural-Field &lt;String&gt;</td>
<td>person’s interests, hobbies or passions</td>
</tr>
<tr>
<td>note</td>
<td>String</td>
<td>notes about</td>
</tr>
</tbody>
</table>
3.1.1 Restrictions/Considerations
OpenSocial supports several social networks but still doesn’t support some popular social networks such as Facebook and Twitter, amongst others. Therefore, other APIs will have to be considered in order to overcome this restriction. The API that we are currently using is Scribe\textsuperscript{16}, discussed in more detail in Section 4.5 of this deliverable. The OpenSocial fields that weren’t already defined in the digital.me ontologies will be further investigated within the next stage of the project, such that the ones which are deemed relevant and important for digital.me can be supported through further ontology extensions.

3.1.2 License
The code is released under the Apache License, Version 2.0\textsuperscript{17}. All OpenSocial specifications are licensed under the Creative Commons Attribution\textsuperscript{18}.

3.2 Keyhole Markup Language (KML)
KML\textsuperscript{19} is a file format used for storing and displaying geographic data within an Earth browser such as Google Earth and Google Maps. The KML 2.2 specification became an official international standard of the Open Geospatial Consortium in 2008. The tag-based structure used is in-line with the XML standard which contains nested elements and attributes. The easiest type of KML documents are the ones that can be authored directly in Google Earth/Maps such as Placemarks, ground overlays, paths, and polygons.

\textsuperscript{16} https://github.com/fernandezpablo85/scribe-java
\textsuperscript{17} http://www.apache.org/licenses/LICENSE-2.0
\textsuperscript{18} http://creativecommons.org/
\textsuperscript{19} https://developers.google.com/kml/documentation/
A Placemark is one of the most common features that are used in Google Earth/Maps, since it represents a position within the Earth’s surface through the use of a pushpin as the icon. The elements used for representing a Placemark are the name – used as the label, description – used as the description, and the Point – which specifies the position on the Earth’s surface in terms of longitude, latitude and optional altitude. The structure of a Placemark file contains an XML header, a KML namespace declaration (line 2 of every KML 2.2 file) and a Placemark object that contains all the elements which were previously mentioned.

3.2.1 Restrictions/Considerations

In digital.me we will be making use of the Placemarks KML documents that represent specific custom maps, as created from a user’s Google Maps account. Such KML documents can easily be downloaded from the user’s account. One restriction that the Placemarks KML documents have is that the type of a Placemark is not specified within each Placemark object. Therefore, it is difficult to distinguish between several types of placemarks such as airports, train stations, restaurants, hotels, etc., unless additional operations are applied such as advanced linguistic techniques.

The information extracted from KML will be converted to instances of new NFO ontology elements introduced by digital.me: namely nfo:PlacemarkContainer instances for the files themselves, and nfo:Placemark instances of the extracted placemarks (refer to Table 2 in D03.01 Ontology Framework [internal deliverable]).

3.2.2 License

There is no known license or patent concerns for the generation and use of the KML format.

3.3 Resource Description Framework in Attributes (RDFa)

RDFa is a data model that enables RDF to be expressed in HTML, XHTML and several XML-based document types. Therefore, rich metadata can be embedded within the mentioned documents through the use of RDFa, which is also a W3C Recommendation. Using RDFa users can easily convey semantic information in the form of structured data within web pages thus, giving the ability to machines and humans to read the exact same content. Additionally, the visual elements on the Web are given a meaning through a set of markup attributes which allow such semantic information to be expressed within e.g., a web page can contain a markup attribute for identifying the title of a blog post. Since RDFa is based on attributes, some HTML attributes such as ‘href’ and ‘src’, have been re-used, whilst some RDFa attributes are new.

RDFa makes use of URLs to identify everything e.g., instead of using a property such as ‘title’, to describe the title of a personal article, the title property –http://purl.org/dc/terms/title– of the Dublin Core Metadata Element Set vocabulary is used. This convention is more flexible since it eliminates the possibility for ambiguities within the terminology, given that a detailed description for each property is just a click away. Such a design also enables easier data portability, consistency and information sharing, whilst humans and machines will both understand what the URL actually refers to. RDFa can also be used to express Google Rich Snippets or schema.org within a web page.

3.3.1 Restrictions/Considerations

In digital.me we will be making use of RDFa for extracting additional meta-data from web pages. Even though at first RDFa was specified for XHTML only, RDFa 1.1 (latest version) is specified for both XHTML and HTML (including all HTML-family languages), together with any XML-based language.

3.3.2 License

There is no known license or patent concerns for the generation and use of RDFa.

---

20 http://www.w3.org/TR/xhtml-rdfa-primer/
21 http://dublincore.org/
22 http://rdfa.info/
3.4 Open Graph
Facebook’s Open Graph is an extension of the social graph (people and the connections that they are interested in), via the Open Graph protocol. The latest version includes third-party websites and pages that users ‘liked’ on the Web, and also actions and objects as created by third-party apps. The Open Graph is able to let apps model user activities based on actions and objects, e.g., a publication app may define the ability to “find” (action) a “paper” (object). Basically, the actions are the verbs that a user performs within the app, whilst the objects characterise the nouns applied to the actions. When an app makes use of the Open Graph, this becomes part of the user’s identity and social graph.

The Open Graph protocol allows a particular Web page to become a rich object within a social graph, e.g., a particular web page using this protocol will have the same functionality as any other object on Facebook. Therefore, basic metadata is required to be added to your Web page in order to convert this Web page into a graph object. The first version of this protocol is based on RDFa, thus the <meta> tags have to be places in the <head> of a web page. On the other hand, the four properties required for every page are: ‘og:title’, ‘og:type’, ‘og:image’ and ‘og:url’. For example, the Open Graph protocol markup for a particular television series would be as follows: title – “Suits”, type – “video.tv_show”, image – “http://ia.media-imdb.com/images/suits.jpg”, and url – “http://www.imdb.com/title/tt1632701/”. Any object can be enhanced with additional optional metadata. Therefore, given that Facebook’s Open Graph is based on RDFa, it can be used to express semantic information such as people, place, events, movies, etc., (as seen above) within a particular Web page.

3.4.1 Restrictions/Considerations
Similar to RDFa, Facebook’s Open Graph protocol will be used in digital.me for enhancing the semantics of every concept within the user’s PIM, as extracted from the meta-data of certain Web pages. One restriction within the Open Graph protocol is that this is not as expressive as RDFa, since you cannot define any concept or property from within a particular ontology (e.g. foaf:Person) but is limited to just the object types and properties as listed within the specifications.

3.4.2 License
The Open Graph specification is available under the Open Web Foundation Agreement, Version 0.9.

3.5 iCalendar (iCal)
iCalendar is a common format that enables users to exchange calendaring and scheduling information such as meeting requests and/or tasks to other users online, via email or through the sharing of files (having the .ics file format). This format helps in solving any interoperability problems that might arise between applications that do not use the same format when defining calendar-related information. Therefore, calendar applications from the same or different organisation can easily communicate with each other, though this format. iCalendar is used and supported in numerous products such as Google Calendar, Yahoo! Calendar, Apple Calendar, etc.

3.5.1 Restrictions/Considerations
The iCalendar standards defined in the RFC 2445 Specification Protocol will be used as is within digital.me.

3.5.2 License
There is no known license or patent concerns for the generation and use of the iCal standard.

23 https://developers.facebook.com/docs/opengraph/
24 http://ogp.me/
25 http://www.openwebfoundation.org/legal/the-0-9-agreements---necessary-claims
26 http://www.ietf.org/rfc/rfc2445.txt
4 Re-use of existing Technology

4.1 Aperture
Aperture is an open source library for crawling and indexing information sources such as file systems, websites and mail boxes. Aperture supports a number of common source types and document formats out-of-the-box and provides easy ways to extend it with custom implementations.

The Aperture framework consists of a number of related but independently usable parts:
- Crawling of information sources: file systems, websites, mail boxes.
- MIME type identification.
- Full-text and metadata extraction of various file formats.

Internally, it uses Apache Tika, a well-known and robust open source project for extracting metadata and structured text content from different resources. The metadata and structured content extracted is then transformed to RDF, based on the Social Semantic Desktop ontologies (which are a subset of the digital.me ontologies, as introduced in D03.03 Ontology Framework).

4.1.1 Restrictions/Considerations
Aperture was developed for desktop machines as part of the Nepomuk project. Since it is written in Java, it is possible to execute in a variety of operating systems, including Android, and theoretically it could run on mobile devices. However, due to the footprint of the library and its dependencies, and incompatibilities with Android, it is not possible to embed Aperture in a client for mobile devices. To work around these restrictions, light metadata extractors need to be developed to run in the mobile client when possible, and for CPU intensive or RAM demanding tasks, the metadata extraction may be done by Aperture in a server, although this increases the amount of data to be transferred back and forth.

4.1.2 License
The Aperture Framework itself is distributed under a BSD-style license27. The copyright holder for each particular source file is given within the file itself. It is free to use and to extend.

4.2 RDF2Go
RDF2Go is an abstraction over triple (and quad) stores, allowing developers to program against RDF2Go interfaces and choose or change the underlying implementation easily.

The three main features of RDF2Go are:
- Program now, decide on triple store later - triple-centric API.
- Easy to extend: only a few simple methods have to be implemented by a triple (or quad) store.
- Triple and quad support - All techniques are available for triple and quad models.

4.2.1 Restrictions/Considerations
The main advantage of RDF2Go, i.e., being able to use the same RDF API for any triple store implementation, is also its weakest point. This is because RDF2Go provides wrappers or bridges to implementations such as Sesame or Jena, but updates on these libraries require new wrappers. Therefore, to be able to use the latest version of Sesame or Jena, one also needs the appropriate wrapper for that version. At the moment, the latest version of Sesame supported by RDF2Go is 2.3.3, which was released on April 7th, 2011. There have been 15 new releases, the last one being 2.6.9, adding new functionality and fixing several bugs, but these are not supported by RDF2Go.

27 http://aperture.sourceforge.net/license.html
4.2.2 License
RDF2Go is released under the new BSD license, although an alternative licensing may be available upon request.

4.3 Apache Commons IO
Commons IO is a library of utilities to assist with developing IO functionality. It is frequently used in Java applications which need IO operations. It comes along with a File Monitor, a component for monitoring file system events (directory and file create, update and delete events).

4.3.1 Restrictions/Considerations
The File Monitor functionality is rather basic, and internally it acts as a runnable which spawns a monitoring thread triggering any registered file alteration at a specified interval, which is 10 seconds by default. If monitoring several directories at once, this can become quite costly.

Current operating systems (Windows, Linux, MacOS, etc.) have built-in functionality to monitor changes in the file system. Performance is better using the built-in notification support, but it makes an application platform dependent. For this reason, it will be considered to integrate OS built-in functions whenever is possible, and keep the Apache Commons IO File Alteration Monitor in other case.

4.3.2 License
Apache Commons IO is released under The Apache Software License, Version 2.0.²⁸

4.4 Scribe
Scribe supports major 1.0a and 2.0 OAuth APIs such as Google, Facebook, LinkedIn, Twitter, Foursquare, Dropbox and Flickr. Therefore, this OAuth Java library, allows a user to connect an application with several service accounts, as mentioned above. Users are also allowed to authenticate with a particular application through signing in with their provider accounts such as Twitter, Facebook and LinkedIn. Scribe takes care of the OAuth exchange between a service provider, a consumer and a user within a web application environment.

4.4.1 Restrictions/Considerations
Although Scribe connects with several service accounts, it doesn’t support OpenSocial (as discussed in Section 3.1 above), which is a very important standard for digital.me. Therefore, one consideration in such a case is to implement an adapter for any OpenSocial compliant service account within Scribe, given that it supports extensions. We will also be extending Scribe to support Fitbit, which uses OAuth Authentication (OAuth 1.0 Protocol).

4.4.2 License
Scribe is licensed under the terms of Apache License, Version 2.0.

4.5 XSPARQL
Given the available tools and languages, the translation between XML to RDF is quite a tedious and error-prone task. Several groups (W3C GRDDL and SAWSDL) have provided a XML-RDF transformation that relied on XSLT. The use of XSLT for handling RDF data is complicated due to the flexibility of the RDF/XML format in comparison to XML’s simple and known hierarchical structure. RDF is conceptually different than XML, and abstracts away from fixed, tree-like structures.

²⁸ http://commons.apache.org/io/license.html
²⁹ http://oauth.net/
³⁰ http://www.w3.org/XML/
³¹ http://www.w3.org/RDF/
³² http://www.w3.org/TR/grddl/
³³ http://www.w3.org/TR/sawsd1/
³⁴ http://www.w3.org/TR/xslt
As an alternative, XSPARQL\textsuperscript{35} (Akhtar et al., 2008) proposes a more natural approach based on merging XQuery\textsuperscript{36} and SPARQL\textsuperscript{37} (both W3C Recommendations) into a novel language for XML-RDF transformations. Given that SPARQL operates over RDF and XQuery over XML, this brings both representations closer together. XSPARQL provides solutions for mapping between XML and RDF in either direction, by addressing the GRDDL and SAWSDL use cases. XSPARQL is also endorsed by W3C as a member submission.

4.5.1 Restrictions/Considerations

In digital.me we are using XSPARQL for transforming returned XML into RDF, when mapping extracted social data from social networks (or any other service account) with the digital.me Ontology Framework. One restriction in XSPARQL is that it does not handle JSON\textsuperscript{38}. Therefore, if any social data is returned in this representation format, it will first require an initial transformation into XML, in order for XSPARQL to be used. To perform this operation, JSON-lib\textsuperscript{39} is used (refer to Section 4.6). Another restriction is that given that the name of any XML tag is the same as any reserved keyword in XSPARQL, this will halt the transformer. Therefore, such XML tags have to be renamed to a non-reserved XSPARQL keyword.

4.5.2 License

XSPARQL\textsuperscript{40} is licensed under the BSD License, GNU Library or Lesser General Public License, Version 2.0 (LGPLv2).

4.6 JSON-lib

JSON-lib has the functionality of converting beans, maps, collections, java arrays and XML to JSON and vice-versa. This java library is being used for converting social data that is retrieved in JSON into XML data, in preparation for the eventual XML-RDF transformation (refer to previous section).

4.6.1 Restrictions/Considerations

One restriction in JSON-lib is that it does not support the renaming of particular JSON name/value pair field names. This creates a problem, since JSON-lib does not support some 'non-colonised names' that are in the form of symbol characters, during the conversion of JSON data to XML data. This restriction is caused by JSON-lib’s inability to serialise JSON objects whose field names are not valid XML tag names. In such cases, the JSON name/value pairs need to be removed before they are converted to XML.

4.6.2 License

JSON-lib is licensed under the terms of Apache License, Version 2.0.

4.7 The GATE Framework

The General Architecture for Text Engineering (GATE)\textsuperscript{41} is an open-source software tool for Language Engineering (LE). Its architecture is able to decompose complex linguistic processes into smaller tasks or modules, thus distributing the work to the appropriate components, whilst ensuring that each component interacts with each other as required. The GATE framework can be extended and customised according to a user’s specific needs, thus reducing the time that developers normally spend for building new LE systems or tweaking existing ones. Several components in GATE can be replaced by other implementations of the same module (e.g. different parsers), which can be fundamental for finding out which modules produce the best results. GATE also supports multilingual data processing (currently 28 languages), through Unicode for representing textual data. It has a built-in Information Extraction (IE) component set – ANNIE\textsuperscript{42}, which contains several main processing resources for common Natural Language Processing (NLP) tasks.

\[
\text{http://xsparql.deri.org/}
\]
\[
\text{http://www.w3.org/XML/Query/}
\]
\[
\text{http://www.w3.org/TR/rdf-sparql-query/}
\]
\[
\text{http://www.json.org/}
\]
\[
\text{http://json-lib.sourceforge.net/}
\]
\[
\text{http://sourceforge.net/projects/xsparql/}
\]
\[
\text{http://gate.ac.uk/}
\]
\[
\text{http://gate.ac.uk/ie/annie.html}
\]
such as a: tokeniser, sentence splitter, Part-of-speech (POS) tagger, gazetteer, finite state
transducer, orthomatcher and coreference resolver (Cunningham et al., 2002). Another module
worth noting in GATE is its Large KB Gazetteer\textsuperscript{43}, used for loading a particular ontology from
RDF. The imported gazetteer is then used for retrieving lookup annotations that have both the
instance and class URI which are being searched for.

4.7.1 Restrictions/Considerations
A limitation in ANNIE is that it is not able to handle all the different types of entities that we aim to
extract. Therefore, its Named Entity Recognition (NER) capability needs to be extended so that it is
able to extract the missing entities. The plan is to build a linguistic analysis pipeline based on the
reusable processing resources that are distributed as part of ANNIE.

The Large KB Gazetteer module will be used in digital.me for making use of the information that is
stored within the user’s PIM.

4.7.2 License
GATE is licenced under the GNU Lesser General Public Licence, Version 3.0.

4.8 DOBBS
DOBBS\textsuperscript{44} is a Firefox plugin that allows the monitoring of the user’s browsing behaviour, in order to
understand how they browse the Web (e.g., how much time they spent on the Web, how long they
stay on a website, how often they visit a website, how they use their browser, etc.). Its main features
are:

- Measuring idle/active time: detecting how long a user is idle (not using the PC), and how
  long a specific Firefox window is in the foreground/background.
- Sessions: determining the duration of the sessions, and how many pages are loaded during
  its session.
- Support for multi-tabbing: detecting which tab is active or in the foreground, and for how long
  pages remain opened in background tabs.

4.8.1 Restrictions/Considerations
The original idea behind was to anonymously study the behaviour of the users while browsing. For
that reason, DOBBS addresses privacy concerns by not collection personal data, and encrypting all
sensitive data (i.e. URLs of visited web pages) before being sent to the server. In digital.me the
approach is the contrary: the user and its personal data need to be gathered, and the specific plain
text URLs are important.

4.8.2 License
DOBBS is licensed under the New BSD License (also known as the Modified BSD License).

\textsuperscript{43} \url{http://gate.ac.uk/sale/tao/splitch13.html#x18-34500013.9}
\textsuperscript{44} \url{http://dobbs.deri.ie/}
5 Functional Documentation

5.1 Distributed URI Naming Scheme

As explained in Section 2.2.4, in digital.me we extend conventions set out by the Nepomuk project in order to enable an environment-independent naming scheme. As a result, URIs in the digital.me userware are UUID-based URNs. Each resource receives a unique automatically generated identifier while keeping a reference to the device/account dependent URL where the resource was found, and provenance metadata such as the device or account identifier. This strategy has the following advantages:

- **Avoiding collisions** when different resources share the same URL in different devices. For example, the file `/home/ismriv/photos/IMG_19.jpg` may exist in two devices, and be actually a different image. Since the digital.me userware integrates data from many data sources, the original URLs of the resources cannot be used as their URIs.

- **Identifiers never change**: the user may change the location of a resource (i.e. moving a file to a different directory), or a data provider may change its API URLs naming conventions. These events may be detected by the different crawlers or account adapters, and only the URL pointing to the location of the resource would need to be updated, while keeping all metadata untouched and the references from/to other resources in the digital.me userware.

Following the conventions laid out above, there does not necessarily be a link whatsoever to either the resource's location within a device/service account, and neither to a device itself. Instead, this information is all stored as metadata. In particular, the reference between a resource and the source wherein it was retrieved is enabled through the use of NIE vocabulary (Mylka et al., 2007); specifically the nie:dataSource property, which links a data object to a data source.

5.2 File Metadata Crawler

The filesystem crawler is a standalone Java application allowing digital.me users to integrate their files and folders into their personal information sphere. Based on the Aperture file crawler (Section 4.1), its functionality has been extended to cover the following features:

- Remote control from digital.me userware: the crawler status and configuration (directories to crawl, types of files, etc.) can be set from the unified digital.me user interface.

- Filtering specific file types: the user can select specific types of files such as “documents”, “pictures”, “videos”, etc., or even be more specific providing mime types, i.e. jpeg files.

- A copy of the files is sent to the user’s PS in order to make them accessible for other users when sharing them, even when the user’s devices are offline or switched off.

- Optimised file monitoring: a full crawl of the selected directories is only performed when the crawler is started. After the first crawl, a daemon detects when a file has been created, modified or deleted, notifying the crawler of such an action. Moreover, the state of the crawl is also persisted in order to be able to detect these events on iterative crawls, even if the file monitor is deactivated.

5.3 Service Integration

Retrieval and integration of data from third-party services is done by using adhoc service adapters. These adapters ensure all personal information, regardless of the service-specific data model or structure, is transformed into an RDF representation, based on the digital.me ontologies.

Data transformation, or semantic lifting, from the service data model to the di.me ontology concepts is leverage by XSPARQL. Each service adapter implementation must specific its own XSPARQL queries (refer to Section 4.5) specifying, in a declarative manner, the transformation rules for each entity or path. The service adapters also abstract the API of each third-party service, by wrapping it with the di.me API and mapping the calls from one to another. The return types of each path of the
di.me API are always concepts from the di.me ontologies. Paths targeted for data integration are: 
/livepost/*, /person/*, /profile/*, /group/*, /event/*, /resource/*, /place/* and /activity/* among others
(refer to D06.03 - Service Integration, Section 4.8). Specific XSPARQL queries for deserialising data
can be found in Appendices 7.1 (LinkedIn), 7.2 (Twitter), 7.3 (Facebook) and 7.6 (Fitbit).

Data formats supported are XML and JSON. XSPARQL can transform XML data into RDF, but not
JSON. To overcome this limitation, service responses using JSON are converted to XML using an
JSON parser (JSON-lib, Section 4.6). LinkedIn, Twitter and Fitbit are able to return data directly in
XML, therefore the XSPARQL transformations are performed straight-away. However, the
Facebook API can only return data in JSON.

Although, the majority of the services require a specific adapter and XSPARQL queries, there are
services which use standard data models such as OpenSocial, KML, or iCal (refer to Section 3).

5.3.1 OpenSocial Adapter
For OpenSocial, an abstract implementation is also provided. It is not aimed to be used out of the box,
since OpenSocial is not a service by itself, but a specification of a data model and API. This
adapter allows the integration of any OpenSocial-compliant service by just setting the specific
communication details such as API root path and authentication type.

The XSPARQL queries provided for deserialising data following the OpenSocial standard can be
found in Appendix 7.5.

5.3.2 KML Adapter
A KML to RDF adapter was implemented in order to be able to extract points-of-interest from both
local files (e.g. exporting Google Earth data) and online resources (e.g. Google Maps). The
XSPARQL queries provided for deserialising KML data can be found in Appendix 7.4. This adapter
is not restricted to a specific service, but instead accepts the URL of the KML location as its input.

5.3.3 iCal Adapter
Similar to the KML adapter, the iCal adapter accepts as input the URL or calendar location, such as
the one provided by Google Calendar, etc. This adapter does not require XSPARQL queries,
instead it takes advantage of the iCal support of Aperture.

5.4 Data Mining
In this section we discuss some of the most important processes that are performed within the data
mining module of digital.me, whose objective is to discover and match PIM entities in data coming
from numerous sources. The matching is both internal (personal) – linking multiple occurrences of
the same item in the user’s mental model in different sources; and external (social) – linking
personal occurrences of an entity that is publicly represented within the Linked Open Data (LOD)
cloud.

5.4.1 Discovering PIM entities in unstructured data
In order to elicit further knowledge from the user’s personal information sphere, we implement a text
analytics pipeline (based on GATE, Section 4.7) for extracting entities in unstructured data. The
latter consists of text that is available in information element names (e.g. file name), subjects (e.g.
email subject), descriptions (e.g. calendar event description) as well live posts (e.g. status
messages). Named Entity Recognition (NER) is performed to find entities within text, and represent
them as instances of ontology elements which match their determined type. Core concepts of the
digital.me ontologies are very similar to the generic entities typically extracted by NER algorithms
(e.g. people, organisations, locations), but they also include more personal (or group) entities (e.g.
projects, events, tasks). NER taggers based on gazetteers are a good fit for entity extraction where
a personal knowledge base (KB) may feedback the algorithm with new entities created either
directly by the user, or as the result of integrating data from an external KB (Scerri et al., 2012a).

The LargeKB Gazetteer module in GATE is being integrated in the implementation in order to make

use of the information that is already stored within the user’s PIM, in order to enrich the information that can be discovered in text. The LaregKB Gazetteer is dynamically populated by loading RDF data and ontologies from an existing Knowledge Base – which in digital.me’s case consists of the PIM. Thus, entities that are already known by the PIM (e.g. “digital.me” as an instance of pimo:Project) can be discovered in unstructured data as it is retrieved (e.g. in a status message referring to “digital.me”). This extended gazetteer will thus be be used to add additional lookup annotations from both instance and class URIs in the PIM KB.

Text documents will also be processed in order to find hidden knowledge within their text. The aim is the same as for other unstructured data sources: extract name entities, and link them to resources of the user’s PIM. Due to some characteristics of these documents (longer text, well-constructed sentences, etc), other NLP techniques are also applied in order to extract keywords and topics for automatic tagging. For each extracted keyword, a tag is automatically created and linked to the resource representing the document. Topics are abstract clusters of keywords/tags that tend to co-occur, and will be named (if not identified) by the user.

Hyperlink resolution will also be applied whenever a link is recognised within text, e.g. in live posts. In social network, hyperlinks are a way how users provide extra information in their posts without having to include the whole content, which in many cases is not possible (e.g. Twitter limitation of 140 characters). These resources are then analysed for:

- extracting relevant metadata to enrich the referencing resource’s metadata (the content of the hyperlinks is important for the resource referencing to it).
- decomposing the LivePost instance into more specific ones such as ImagePost, VideoPost, etc. (refer to D03.01 Ontology Framework [internal deliverable]).

For every web document fetched by following a hyperlink, these actions are taken:

- Remove the clutter around the relevant content, such as navigation bars, footers, ads, etc. Boilerplate provides different algorithms and strategies for this task, optimised for different types of Web documents such as news articles, blog posts, etc.
- Keyword extraction: GATE-based algorithm also used for text documents (see above).
- Extraction of RDFa metadata: due to the existence of standards such as RDFa (Section 3.3), web documents may contain embedded structured metadata, which can be more readily extracted without needing to be processed using the NER techniques, and directly be used in di.me.
- Analysis of meta tags: specifically looking for Open Graph (Section 3.4). This mechanism is based on RDFa, although the approach diverges from RDFa, and a special ad-hoc algorithm is required to extract this information.

Lastly, a live post can be decomposed into more specific posts depending on the type of the referenced resource (hyperlink). The first feature analysed is the mime-type of the resource, hence if a image, video, audio file are the resources linked, the corresponding ImagePost, VideoPost or AudioPost instances, having the resource as the ‘defining resource’ will be created. However, when the mime type is text/html, further analysis is carried out on the metadata extracted from the web document (RDFa, meta tags, etc.). Metadata described by vocabularies such as Open Graph or schema.org is then used to determine the type of live post (e.g. og:video).

### 5.4.2 Linking PIM knowledge to external knowledge

By the end of the project lifetime, we intend to enable the automatic enrichment of the PIM with external public (social) knowledge. Recognised PIM entities will be linked to corresponding entities that are discovered in public RDF datasets in the LOD cloud, or other external sources. For example, the city concept – ‘Galway’, which in the user’s PIM is an instance of ‘pimo:City’, can be linked (using properties such as nao:isRelated) to several corresponding LOD resources such as a DBPedia resource i.e. [http://dbpedia.org/resource/Galway](http://dbpedia.org/resource/Galway).

---

5.4.3 PIM Integration

Following the semantic lifting (elicitation and matching to PIM elements) of extracted personal information, the data mining module attempts to integrate the newly discovered information with existing PIM knowledge. For the purpose, the discovered entity (e.g. a 'city') is matched against all PIM items of the same type (all pimo:City instances in the PIM). This matching procedure consists of the following stages:

1) Syntactic Matching: generic attributes of the new entity are compared to the corresponding attributes of candidate entities having the same type. These attributes are typically derived from the NAO ontology (Scerri et al., 2007), which by definition contains ‘generic metadata for resources’, and can thus apply across the board for any type of resource. Other attributes apply for the most common types of resources, including information elements and files. Examples of generic resource/information element/file attributes are:

a) nao:prefLabel. In digital.me will always carry the label of an item as can be seen by the user. Therefore this attribute will contain the text corresponding to the extracted Named Entity.
b) nie:title. The name of extracted information elements (files, folders, etc.) will be stored using this property.
c) nie:url. Information Elements retrieved from the Web will have the same url.
d) nfo:hasHash. Identical files will have the same hash generated regardless of the source from which they were retrieved.
e) nao:externalIdentifier. Items retrieved from public sources (including social networks) will have an identical external identifier (e.g. reference to a tag on Twitter or a group on Facebook).

Further attributes are considered for the matching of specific entity types, such as person names (nco:hasPersonName) for contacts/persons (refer to Section 5.4.4) and ncal:location for calendar events.

2) Semantic Search Extension: Although syntactically two items might not be considered a match, due to variations in names etc., a new item might still be semantically equivalent to existing items in the PIM. In this part of the matching procedure we try to identify common relationships between the new entity and existing PIM items, e.g. although an event is known by different names, it might still be located in the same city, take place at the same date/time, and be related to the same topics. To perform the semantic search, the user’s PIM is used as the main KB, since it contains knowledge which is most likely to be relevant to the user. Remote KBs such as DBPedia47 are also used to support detection of a semantic relationship between two profile attributes, when no link can be found within the PIM. As with syntactic matching, there are a number of attributes which are given more weight when attempting to quantify the semantic similarity of two entities based on their common relationships:

a) nao:isRelated. A high number of common ‘related’ resources, defined through this property, may indicate a match.
b) pimo/nao:hasTag. A high number of common tags, may indicate a match.
c) pimo/nao:hasTopic. A high number of common ‘topics’, may indicate a match.
d) pimo:hasLocation/locatedWithin. The location of an entity (person, organisation, building, event, etc.) may not be exactly the same. However, due to the transitive nature of the pimo:locatedWithin property, it might result that there is a match between an event that takes place in Galway and an event that takes place in Ireland.

Following the syntactic and semantic matching, the implemented algorithm will determine whether the discovered relationships constitute a match. For obvious reasons, the syntactic matching results are generally given a higher weighting in the decision. When the match score is considered as definite, the two entities are integrated at the PIMO (refer to Deliverable 3.1 Ontology Framework [internal deliverable], Section 3.2.1.2) level automatically. In some cases, such as person matching,

47 http://dbpedia.org/
high scores are regardless presented as ‘possible matches’ to the user, who can then approve or disapprove their integration. In either case, the PIM integration will result in one unique PIM entity (e.g. pimo:Event) point to one or more occurrences (ncal:Event) using the special pimo:occurrence property.

5.4.4 Matching Persons behind multiple Profiles

The last section describes the techniques for determining semantic equivalence between PIM entities. One of the most important use cases for this technique, is the identification and subsequent resolution of federated person profiles, given that this may impact privacy-sensitive data sharing decisions (Bourimi et al., 2012). In this section, we elaborate on how the matching techniques presented in the last section are extended for this particular use case, i.e. to detect semantically equivalent online user profiles (i.e. they belong/represent the same person) as retrieved from multiple devices (and applications) CRM systems and other online service accounts.

The approach involves four different steps, as shown in Figure 2. The first stage consists of the extraction of the user’s semi/unstructured personal information, as is made available by multiple service account APIs (stage A, refer to Section 5.3). As the entity type is already known, this information is thus lifted onto the digital.me ontologies (stage B) as a new instance of nco:Contact.

In stage C, the new entity (contact) is matched against existing contact instances in the PIM. As shown in Figure 2, stage C consists of four distinctive steps. First, linguistic analysis is performed on the profile attributes that contain complex or unstructured information such as a postal address or person description. Step 2 performs syntactic matching as explained in the previous section. Attributes of datatype string are matched directly (e.g. hasPersonName, hasPostalAddress, hasEmailAddress) whereas attributes of other datatype values (e.g. birthDate, age) are matched depending on their expected value. In addition, as attributes containing text (e.g. hasPostalAddress, Affiliation) might contain entities whose type could not be resolved, profile matching also performs what is termed ‘Indirect String Matching’. In this case, the entity text is compared against the nao:prefLabel of any PIM entity, in order to try and discover its type.

After the semantic search extension explained in Section 5.4.3 is performed, a score function, which includes weights for different profile attributes, will then factor in the results of all the matching phases in stage C, in order to determine whether there exists a semantic equivalence between a retrieved profile and one or more existing profiles in the PIM. Thus, multiple profiles for a single person are reconciliated and integrated in the user’s PIM (Cortis et al., 2012). This has various privacy-related implications, as covered in D4.03 Infrastructure Mobile Security, Section 5.2.
5.5 User Activity Context Extraction

One of the new personal information aspects being targeted by digital.me are the user’s own perceivable activities, as gathered through a variety of physical and virtual ‘sensors’ available on personal devices, which act as a proxy. For the purpose, a context crawler is available for each type of personal device.

The Context information extraction components described in this section will be used as input for the context processing component, described in Deliverable 3.4. The list is non-binding and not exclusive, meaning that all listed sources are planned to be integrated by the project lifetime, but some may not be feasible, whereas others that are not listed might be integrated in the future. The information extracted is sent to the LiveContext graph, which contains a dynamically updated instance of the DCON Ontology that reflects the current context of a user (refer to D03.01 Ontology Framework [internal deliverable], Section 3.6).

5.5.1 Context Extraction from Personal Applications and Services

Some kinds of user activity context information can be obtained through the device and service crawlers. The context context crawler currently performs on these applications and services. Beside each item in brackets, is the relevant DCON context aspect (Scerri et al., 2012b), to which each piece of context information will be attached:

- **Events (Schedule Aspect):** A user’s personal current and planned events will be crawled from calendaring tools and other external services.
- **Tasks (Schedule Aspect):** Tasks in a task manager can be read to retrieve a user’s current and planned tasks.
- **Online Presence (State Aspect):** Detection of a user’s online presence from various applications (e.g. Skype, Whatsapp) and services (e.g. Gtalk status).
- **Files and Applications (Attention Aspect):** Detection of a user’s attention through monitoring their device activities to determine the applications running/in the foreground, the files being open and/or edited.
- **Web browsing (Attention Aspect):** Given the vast amount of activities that are able to be performed while browsing the internet, internet browsers are applications which are given special treatment. Information extracted includes visited web pages, open tabs both in the foreground and background.

5.5.2 Context Extraction from Device Sensors

Exploiting the capabilities of the mobile devices, different types of raw context data can be retrieved from many device sensors. The following list of sensors is being integrated in digital.me:

- **Wi-Fi (Connectivity Aspect):** this sensor retrieves the list of MAC addresses of the nearby Wi-Fi access points and the corresponding signal strength expressed in dBm. To avoid possible incorrect scans, provided signals strengths are an average of last detected values. Connectivity Aspect is also improved detecting if the device is connected to one of the nearby APs.
- **Bluetooth (Connectivity Aspect):** this sensor retrieves the list of the Bluetooth addresses visible nearby. The id of the local Bluetooth interface is also provided.
- **Cell (Connectivity Aspect):** this sensor retrieves the identifier of the mobile cell the device is connected to. If the device is connected to the data network, an indication of which type of connectivity (e.g. UMTS, HSDPA, etc.) is available is also provided.
- **Position (SpaTem Aspect):** this sensor retrieves current geographic location of the device in terms of latitude and longitude. An accuracy indication is also provided together with altitude measurement.
- **Compass (SpaTem Aspect)**: the compass is used to retrieve user’s movements.
- **Accelerometer (SpaTem Aspect)**: the accelerometer is used to retrieve the user’s movement speed.
- **Microphone (SpaTem Aspect)**: the microphone can be used to detect environmental noise nearby.
- **Device Mode**: Mostly relevant for smartphones, which can have various modes enabled (e.g. silent, loud, vibrate).

### 5.5.3 Interpreting Mid-level Context

Raw context data extracted from services, sensors and applications are sent to the PS, where it can be processed to infer - using also external services support if necessary – mid-level context information to further enrich the user’s live context:

- **Places (SpaTem Aspect)**: the system uses raw position detected by the device to search for the current and nearby places. An external service is used to get a list of Points of Interest nearby a given couple of geographic coordinates. Using position and indoor Wi-Fi-based location (retrieved by an external service given the WiFi information coming from the device) it is possible to infer that a user is actually in a specific place.
- **Current Time Period (SpaTem Aspect)**: by comparing the known time of day/week/month/year (from the system clock) to the value ranges for existing DPO Presence Ontology instances (refer to D03.01 Ontology Framework [internal deliverable], Section 3.4), it can be deduced which time period the user is in (e.g. morning, holiday season, summertime).
- **User's Proximity (Peers Aspect)**: Bluetooth and Wi-Fi information coming from the device are given as input to an external service able to compute them and to return an estimation of the proximity to other devices. This result is translated inside the PS as corresponding nearby peers.
- **Direction (SpaTem Aspect)**: the compass is used to retrieve user’s movements. This information can overtime determine the user’s travelling direction.
- **Speed (SpaTem Aspect)**: the accelerometer raw data can be used to infer more complex user’s motion situation as defined in the DPO ontology like running, walking, etc.
- **Weather (Environment Aspect)**: the user’s position can be given as input to an external service able to return weather forecast in terms of weather conditions, temperature, humidity, etc. In addition, this information can also be compared to existing DPO instances in order to detect which weather category currently surrounds the user.
- **Activity (State)**: The combination of various types of raw context information can hint towards the activity of the user, as provided by DPO instances (e.g. working, sleeping, travelling). In particular, Attention information (files, applications, Web browser), together with the Online Presence and Location can hint towards the user’s activity. As this mid-level context information is the most challenging to determine, the userware should also allow the user to manually select it.

### 5.5.4 Context Extraction from Virtual Sensors

A user’s online activities constitute a rich source for extracting user context information, since they are usually the only means by which a typical user consciously puts effort into sharing their activities. In view of this opportunity, digital.me targets the extraction/discovery of additional context information from multiple streams of heterogeneous online posts, through advanced linguistic techniques. The context crawler includes a component that extracts implicit context knowledge embedded in multiple streams of heterogeneous online posts.

Online posts are retrieved through service account adapters (refer to Section 5.3), and transformed from semi-structured posts to an instance of the DLPO Ontology (refer to D03.01 – Ontology Framework [internal deliverable], Section 3.5) through XSPARQL (refer to Section 4.5). The textual
content of a post is analysed by a text analytics pipeline (refer to Section 5.4.1) which starts with the following operations: emoticons removal, abbreviations substitution/removal (using noslang.com as an abbreviations dictionary), POS tagging, stop words removal and stemming. NER is then performed in order to lift the recognised entities to the user’s PIM. The Large KB Gazetteer GATE module is also applied to query the PIM in order to create a gazetteer for entity lookup.

Information retrieved from online posts using the above-described techniques enriches the overall known user activity context. For example, in an online post saying “In the MSM workshop in Lyon. Simon is giving a great presentation, amazing work!”, ‘Simon’ is recognised as a named-entity (Person). This entity is matched to a specific `pimo:Person` instance by combining context information from the desktop and mobile crawlers, e.g. a workshop event (instance of `ncal:Event`) that is known to be taking place at the same time, and in which the same person is also marked as an attendee (Scerri et al., 2012a). The following is a list of entities targeted for extraction from online posts, which act as ‘virtual’ sensors:

- **Activity (State Aspect)** – current activity of the user as deduced from a user’s online activities such as a status message.
- **Availability (State Aspect)** – current availability of the user as deduced from a user’s online availabilities.
- **Person (Peers Aspect)** – persons (peers) that are known to be nearby as identified by their network connections.
- **Event (Schedule Aspect)** – current/upcoming event that the user will be participating in or soon-to-be in, as retrieved from a calendar entry.
- **Place/Location (SpaTem Aspect)** – nearby/current place as represented from the user’s spatial and temporal information.
6 References


OpenSocial Specification 2.5.0 DRAFT. Available at: http://opensocial-resources.googlecode.com/svn/spec/trunk/OpenSocial-Specification.xml.


7 Appendix

7.1 LinkedIn XSPARQL queries
The LinkedIn XSPARQL queries in the sub-sections below target the retrieval of: i) all the LivePosts within the user’s timeline, ii) all the user’s personal LivePosts, iii) all the contacts of the user, and iv) the user’s profile personal information.

7.1.1 /livepost/@all

```
declare namespace nco = "http://www.semanticdesktop.org/ontologies/2007/03/22/nco#";
declare namespace dlpo = "http://www.semanticdesktop.org/ontologies/2011/10/05/dlpo#";
declare namespace xsd = "http://www.w3.org/2001/XMLSchema#";

let $doc := doc("%doc%")
let $statuses := $doc/network/updates/update
return
  for $status in $statuses
    let $statusId := $status/update-key
    let $stId := fn:replace($statusId,'-','')
    let $timestamp := $status/timestamp
    let $statusMessage := $status/update-content/person/current-status
    let $counter := count($statusId/preceding::*) + count($statusId/ancestor::*)
    #Construct for the status message
    construct
      {_:stm{data($counter)} a dlpo:Status;
       nao:externalIdentifier {data($stId)};
       dlpo:timestamp {data($timestamp)}^^xsd:dateTime;
       dlpo:textualContent {data($statusMessage)};
       nao:creator _c{data($counter)}.
      }
    #Construct for the user's information
    construct
      {
      _c{data($counter)} a nco:PersonContact;
      nao:externalIdentifier {data($userId)};
      nco:photo {data($photoUrl)};
      nao:description {data($summary)};
      nco:hasPersonName _pn{data($counter)}.
      _pn{data($counter)} a nco:PersonName;
      nco:nameGiven {data($name)};
      nco:nameFamily {data($surname)};
      nco:fullname {fn:concat(data($name)," ",data($surname))}.
      }
    }
```

7.1.2 /livepost/@me/@all

```
declare namespace nco = "http://www.semanticdesktop.org/ontologies/2007/03/22/nco#";
declare namespace dlpo = "http://www.semanticdesktop.org/ontologies/2011/10/05/dlpo#";
declare namespace xsd = "http://www.w3.org/2001/XMLSchema#";

let $doc := doc("%doc%")
let $status := $doc/person/current-share
let $statusId := fn:replace($status/id,'-','')
return
  #Construct for the status message
  if (fn:not(($statusId)="")) then {
    let $timestamp := $status/timestamp
    return
      {_:stm{data($counter)} a dlpo:Status;
       nao:externalIdentifier {data($statusId)};
       dlpo:timestamp {data($timestamp)}^^xsd:dateTime;
       dlpo:textualContent {data($statusMessage)};
       nao:creator _c{data($counter)}.
      }
`
let $statusMessage := $status/message  #changed xml 'comment' element to 'message'

construct
{
  _stm{data($statusId)} a dlpo:Status;
  nao:externalIdentifier {data($statusId)};
  dlpo:timestamp {data($timestamp)}^^xsd:dateTime;
  dlpo:textualContent {data($statusMessage)};
  nao:creator _c{data($statusId)}.
}

let $userId := fn:replace($status/author/id,'-','')
let $name := $status/author/first-name
let $surname := $status/author/last-name

#Construct for the user's information
construct
{
  _c{data($statusId)} a nco:PersonContact ;
  nao:externalIdentifier {data($userId)} ;
  nco:hasPersonName _pn{data($statusId)}.
  _pn{data($statusId)} a nco:PersonName;
  nco:nameGiven {data($name)};
  nco:nameFamily {data($surname)};
  nco:fullname {fn:concat(data($name)," ",data($surname))}.
}
}

7.1.3 /person/@me/@all

let $doc := doc("%doc%")
let $connections :=$doc/connections/person

return
for $connection in $connections
let $connectionId := fn:replace($connection/id, '-', '')
let $name := $connection/first-name
let $surname := $connection/last-name
let $locations := $connection/location
let $photoUrl := $connection/picture-url
let $summary := $connection/summary
let $positions := $connection/positions/position
let $phones := $connection/phone-numbers/phone-number
let $ims := $connection/im-accounts/im-account
let $dob := $connection/date-of-birth
let $dobdd := if ((fn:string-length($dob/day))=1)
  then (fn:concat("0",$dob/day))
  else ($dob/day)
let $dobmm := if ((fn:string-length($dob/month))=1)
  then (fn:concat("0",$dob/month))
  else ($dob/month)
let $dobyyyy := if ((fn:string-length($dob/year))=4)
  then ($dob/year)
  else ("")
let $birthdate := fn:concat(data($dobyyyy),"-",data($dobmm),"-",data($dobdd))
let $address := if (fn:exists($connection/main-address))
  then ($connection/main-address)
  else ("")
let $interests := fn:tokenize($connection/interests,", ")

#Construct for the connection
construct
{
  _c{data($connectionId)} a nco:PersonContact;
  nao:externalIdentifier {data($connectionId)};
  nco:photo {data($photoUrl)};
  nao:description {data($summary)};  #'summary' field for each connection is not retrieved from LinkedIn API - probably an API bug
  nao:prefLabel {fn:concat(data($name)," ",data($surname))};
  nao:prefSymbol {data($photoUrl)};
  nco:hasPersonName _pn{data($connectionId)}.
_:pn{data($connectionId)} a nco:PersonName;
nco:nameGiven {data($name)};
nco:nameFamily {data($surname)};
nco:fullname {fn:concat(data($name)," ",data($surname))}.

#Construct for the interests of the user
for $interest in $interests
  construct
  [  
    _:c{data($connectionId)} nco:hobby {data($interest)};
  ]
).

#Construct for the birth date of the user
if ((fn:string-length($birthdate))=10) then ({
  let $counter := count($dob/preceding::*) + count($dob/ancestor::*)
  construct
    [     
      _:bd{data($counter)} a nco:BirthDate;
      nco:birthDate {data($birthdate)}^^xsd:date.
      _:c{data($connectionId)} nco:hasBirthDate _:bd{data($counter)};
    ]
  })
else ()
).

#Construct for the positions of the user
for $position in $positions
  let $posId := fn:string($position/id)
  let $title := $position/title
  let $posStartYr := $position/start-date/year
  let $posStartMo := if ((fn:string-length($position/start-date/month))=1)
    then (fn:concat("0",$position/start-date/month))
    else if ((fn:string-length($position/start-date/month))=0)
      then ("01")
    else $position/start-date/month
  let $posStart := if (fn:exists($position/start-date))
    then (fn:concat(data($posStartYr),"-",data($posStartMo),",",-
      ,data("01")","00:00:002"))
    else ()
  let $posEndYr := $position/end-date/year
  let $posEndMo := if ((fn:string-length($position/end-date/month))=1)
    then (fn:concat("0",$position/end-date/month))
    else if ((fn:string-length($position/end-date/month))=0)
      then ("01")
    else $position/end-date/month
  let $posEnd := if (fn:exists($position/end-date))
    then (fn:concat(data($posEndYr),"-",data($posEndMo),",",-
      ,data("01")","00:00:002"))
    else ()
  let $counter := count($position/preceding::*)) + count($position/ancestor::*))
  let $orgID := count($position/company/preceding::*)) + count($position/company/ancestor::*))
  construct
    [     
      _:pos{data($counter)} a nco:Affiliation;
      nao:externalIdentifier {data($posId)}^^xsd:string;
      nco:role {data($title)};
      nco:start {data($posStart)}^^xsd:dateTime;
      nco:end {data($posEnd)}^^xsd:dateTime;
      nco:org _:org{data($orgID)}.
      _:c{data($connectionId)} nco:hasAffiliation _:pos{data($counter)}.
    ]
  }
else ()
let $company := $position/company/name
  construct
    [     
      _:org{data($orgID)} a nco:OrganizationContact;
      nie:title {data($company)}.
    ]
  }
}.

#Construct for the phone numbers of the connection
for $phone in $phones
  let $phoneNumber := $phone/phone-number
  let $counter := count($phone/preceding::*)) + count($phone/ancestor::*))
  construct
    [     
      _:c{data($connectionId)} nco:PhoneNumber {data($phoneNumber)};
    ]
  ]
}.

#Construct for the email addresses of the connection
for $email in $emails
  construct
    [     
      _:c{data($connectionId)} nco:EmailAddress {data($email)};
    ]
  ]
}.

#Construct for the addresses of the connection
for $address in $addresses
  construct
    [     
      _:c{data($connectionId)} nco:Address {data($address)};
    ]
  ]
}.

#Construct for the mailing addresses of the connection
for $address in $mailingAddresses
  construct
    [     
      _:c{data($connectionId)} nco:MailingAddress {data($address)};
    ]
  ]
}.

#Construct for the other contact details of the connection
for $otherContact in $otherContacts
  construct
    [     
      _:c{data($connectionId)} nco:OtherContact {data($otherContact)};
    ]
  ]
}.
construct
{
  _:ph{data($counter)} a nco:PhoneNumber;
  nco:phoneNumber {data($phoneNumber)}.
  _:c{data($connectionId)} nco:hasPhoneNumber _:ph{data($counter)};
}
).

#Construct for the instant messenger accounts of the connection
{
  for $im in $ims
    let $imName := $im/im-account-name
    let $imType := $im/im-account-type
    let $counter := count($im/preceding::* ) + count($im/ancestor::* )
  construct
  {
    _:im{data($counter)} a nco:IMAccount;
    nco:imAccountType {data($imType)};
    nco:imID {data($imName)}.
    _:c{data($connectionId)} nco:hasIMAccount _:im{data($counter)};
  }
}.

#Construct for the address of the user
{
  if (fn:not((data($address)=""))
  then {
    let $counter := count($address/preceding::* ) + count($address/ancestor::* )
  construct
  {
    _:addr{data($counter)} a nco:PostalAddress;
    nao:prefLabel {data($address)}.
    _:c{data($connectionId)} nco:hasPostalAddress _:addr{data($counter)};
  }
  } else ()
  }
}

7.1.4 /profile/@me/@all

declare namespace nco = "http://www.semanticdesktop.org/ontologies/2007/03/22/nco#";
declare namespace nie = "http://www.semanticdesktop.org/ontologies/2007/01/19/nie#";
declare namespace xsd = "http://www.w3.org/2001/XMLSchema#";

let $doc := doc("%doc%")
let $persons :=$doc/person
return
for $person in $persons
let $pid := "1"
let $personId := fn:replace($person/id, '-', '')
let $name := $person/first-name
let $surname := $person/last-name
let $locations := $person/location
let $photoUrl := $person/picture-url
let $summary := $person/summary
let $positions := $person/positions/position
let $ims := $person/im-accounts/im-account
let $dob := $person/date-of-birth
let $dobdd := if ((fn:string-length($dob/day))=1)
  then (fn:concat("0",$dob/day))
  else ($dob/day)
let $dobmm := if ((fn:string-length($dob/month))=1)
  then (fn:concat("0",$dob/month))
  else ($dob/month)
let $dobyyyy := if ((fn:string-length($dob/year))=4)
  then ($dob/year)
  else ("")
let $birthdate := fn:concat(data($dobyyyy),"-",data($dobmm),"-",data($dobdd))
let $address := if (fn:exists($person/main-address))
  then ($person/main-address)
  else ("")
let $interests := fn:tokenize($person/interests," ", "")

#Construct for the user
construct {
  _:c{data($pid)} a nco:PersonContact;
  nao:externalIdentifier [data($personId)];
  nco:photo [data($photoUrl)];
  nao:description [data($summary)];
  nao:prefLabel {fn:concat(data($name),", ",data($surname))};
  nco:hasPersonName _:pn{data($pid)}.
  _:pn{data($pid)} a nco:PersonName;
  nco:nameGiven [data($name)];
  nco:nameFamily [data($surname)];
  nco:fullname {fn:concat(data($name),", ",data($surname))}.
}

#Construct for the interests of the user
for $interest in $interests
  construct {
    _:c{data($pid)} nco:hobby [data($interest)];
  }.

#Construct for the birth date of the user
if ((fn:string-length($birthdate))=10) then {
  let $counter := count($dob/preceding::*) + count($dob/ancestor::*)
  construct {
    _:bd{data($counter)} a nco:BirthDate;
    nco:birthDate [data($birthdate)]^^xsd:date.
    _:c{data($pid)} nco:hasBirthDate _:bd{data($counter)};
  } else ().
}

#Construct for the positions of the user
for $position in $positions
  let $posId:= $position/id
  let $title := $position/title
  let $posStartYr := $position/start-date/year
  let $posStartMo := if ((fn:string-length($position/start-date/month))=1)
    then (fn:concat("0",$position/start-date/month))
    else if ((fn:string-length($position/start-date/month))=0)
      then ("01")
    else (fn:position/start-date/month)
  let $posEndYr := $position/end-date/year
  let $posEndMo := if ((fn:string-length($position/end-date/month))=1)
    then (fn:concat("0",$position/end-date/month))
    else if ((fn:string-length($position/end-date/month))=0)
      then ("01")
    else (fn:position/end-date/month)
  let $posStart := if (fn:exists($position/start-date))
    then (fn:concat(data($posStartYr),"-",data($posStartMo),"-
",data("01"),"T","00:00:02z"))
  else ()
  let $posEnd := if (fn:exists($position/end-date))
    then (fn:concat(data($posEndYr),"-",data($posEndMo),
"-
",data("01"),"T","00:00:02z"))
  else ()
  let $counter := count($position/preceding::* + count($position/ancestor::*))
  let $orgId := count($position/company/preceding::* + count($position/company/ancestor::*))
  construct {
    _:pos{data($counter)} a nco:Affiliation;
    nco:externalIdentifier [data($posId)]^^xsd:string;
    nco:role [data($title)];
    nco:start [data($posStart)]^^xsd:dateTime;
    nco:end [data($posEnd)]^^xsd:dateTime;
    nco:org _:org{data($orgId)}.
    _:c{data($pid)} nco:hasAffiliation _:pos{data($counter)}.
    [let $company := $position/company/name
    construct {
      _:org{data($orgId)} a nco:OrganizationContact;
    }.
  }.
nie:title {data($company)}. 
}

#Construct for the phone numbers of the user
{
  for $phone in $phones
    let $phoneNumber := $phone/phone-number
    let $counter := count($phone/preceding::*) + count($phone/ancestor::*
    construct
      {_iph(data($counter)) a nco:PhoneNumber;
       nco:phoneNumber {data($phoneNumber)}.
      }
    }
}

#Construct for the instant messenger accounts of the user
{
  for $im in $ims
    let $imName := $im/im-account-name
    let $imType := $im/im-account-type
    let $counter := count($im/preceding::*) + count($im/ancestor::*
    construct
      {_iim(data($counter)) a nco:IMAccount;
       nco:imAccountType {data($imType)};
       nco:imID {data($imName)}.
      }
    }
}

#Construct for the address of the user
{
  if (fn:not((data($address)="")))
    then ({
      let $counter := count($address/preceding::*) + count($address/ancestor::*
      construct
        { _iaddr(data($counter)) a nco:PostalAddress;
          nao:prefLabel {data($address)}.
        }
    })
  else ()
}

7.2 Twitter XSPARQL queries
The Twitter XSPARQL queries in the sub-sections below target the retrieval of: i) all the LivePosts within the user’s timeline, ii) all the user’s personal LivePosts, iii) all the contacts of the user, and iv) the user’s profile personal information.

7.2.1 /livepost/@all

declare namespace nco = "http://www.semanticdesktop.org/ontologies/2007/03/22/nco#";
declare namespace dlpo = "http://www.semanticdesktop.org/ontologies/2011/10/05/dlpo#";
declare namespace geo = "http://www.w3.org/2003/01/geo/wgs84_pos#";
declare namespace xsd = "http://www.w3.org/2001/XMLSchema#";

let $doc := doc("%doc%")
let $liveposts := $doc/statuses/status
return
for $livepost in $liveposts
let $timestamp := $livepost/created_at
let $livepostMessage := $livepost/text
let $livepostId := $livepost/id
let $user := $livepost/user
let $userId := $user/id
# Construct for the livepost

```
construct 
{ 
  _:lvp{data($livepostId)} a dlpo:Status;  
  nao:externalIdentifier {data($livepostId)};  
  dlpo:timestamp {data($timestamp)}^^xsd:dateTime;  
  dlpo:textualContent {data($livepostMessage)};  
  nao:creator _:c{data($userId)}.
}

let $userFullname := $user/name
let $userPhotoUrl := $user/profile_image_url
let $userDescription := if (fn:not(($user/description)=""))
  then ($user/description)
  else ()
let $userLocation := $user/location
let $username := $user/screen_name
let $userUrl := if (fn:not(($user/url)=""))
  then ($user/url)
  else ()

# Construct for the user's information

construct 
{ 
  _:c{data($userId)} a nco:PersonContact; 
  nco:contactUID {data($userId)}; 
  nco:photo {data($userPhotoUrl)}; 
  nao:description {data($userDescription)}; 
  nco:websiteUrl {data($userUrl)}; 
  nco:hasName :name{data($userId)} . 
  :name{data($userId)} a nco:Name; 
  nco:fullname {data($userFullname)}; 
  nco:nickname {data($username)}. 
}

# Construct for the location of the user

{ 
  if (fn:not(($userLocation)=""))
    then {
      let $counter := count($userLocation/preceding::*) +
        count($userLocation/ancestor::*):
      
        construct 
        { 
          _:loc{data($counter)} a nco:PostalAddress;  
          nao:prefLabel {data($userLocation)}.
          _:c{data($userId)} nco:hasPostalAddress _:loc{data($counter)};
        }
      }
    else ()
}
}
```

7.2.2 /livepost/@me/@all

declare namespace nco = "http://www.semanticdesktop.org/ontologies/2007/03/22/nco#";
declare namespace dlpo = "http://www.semanticdesktop.org/ontologies/2011/10/05/dlpo#";
declare namespace geo = "http://www.w3.org/2003/01/geo/wgs84_pos#";
declare namespace xsd = "http://www.w3.org/2001/XMLSchema#";

let $doc := doc("%doc%")
let $liveposts := $doc/statuses/status
return

for $livepost in $liveposts
let $timestamp := $livepost/created_at
let $livepostMessage := $livepost/text
let $livepostId := $livepost/id
let $type := if ($livepost/in_reply_to_status_id=""
  then "Status"
  else "Comment"
) let $replyOfId := if (fn:not(($livepost/in_reply_to_status_id)=""))
  then ($livepost/in_reply_to_status_id)
  else ()
let $replyOfUserId := if (fn:not({$livepost/in_reply_to_user_id}=""))
then {$livepost/in_reply_to_user_id}
else ()
let $replyOfUsername := $livepost/in_reply_to_screen_name
let $user := $livepost/user
let $userId := $user/id

#Construct for the livepost
construct
{
    _1vp{data($livepostId)} a dlpo:{data($type)};
    nao:externalIdentifier {data($livepostId)};
    dlpo:timestamp {data($timestamp)}^^xsd:dateTime;
    dlpo:textualContent {data($livepostMessage)};
    nao:creator _:c{data($userId)}.
    
    let $userFullname := $user/name
    let $userPhotoUrl := $user/profile_image_url
    let $userDescription := if (fn:not({$user/description}=""))
then ($user/description)
else ()
    
    let $userLocation := $user/location
    let $username := $user/screen_name
    let $userUrl := if (fn:not({$user/url}=""))
then ($user/url)
else ()

    #Construct for the user's information
    construct
    {
        _c{data($userId)} a nco:PersonContact;
        nco:contactUID {data($userId)};
        nco:photo {data($userPhotoUrl)};
        nao:description {data($userDescription)};
        nco:websiteUrl {data($userUrl)};
        nco:hasName _name{data($userId)} .
        _name{data($userId)} a nco:Name .
        nco:fullname {data($userFullname)};
        nco:nickname {data($username)}.
    }

    #Construct for the location of the user
    {
        if (fn:not({$userLocation}=""))
        then {
            let $counter := count($userLocation/preceding::*) +
            count($userLocation/ancestor::*);
            
            construct
            {
                _loc{data($counter)} a nco:PostalAddress;
                nao:prefLabel {data($userLocation)}.
                _c{data($userId)} nco:hasPostalAddress _:loc{data($counter)};
            }
        }
        else {}
    }
    }

    _1vp{data($livepostId)} dlpo:relatedResource _:c{data($replyOfUserId)}.
    _c{data($replyOfUserId)} a nco:PersonContact;
    nco:contactUID {data($replyOfUserId)};
    nco:hasName _name{data($replyOfUserId)} .
    _name{data($replyOfUserId)} a nco:Name .
    nco:nickname {data($replyOfUsername)}.
    _1vp{data($livepostId)} dlpo:replyOf _1vp{data($replyOfId)}.
    _1vp{data($replyOfId)} a dlpo:Status;
    nao:externalIdentifier {data($replyOfId)};
    nao:creator _:c{data($replyOfUserId)}.
}

7.2.3 /person/@me/@all

declare namespace nco = "http://www.semanticdesktop.org/ontologies/2007/03/22/nco#";
declare namespace nie = "http://www.semanticdesktop.org/ontologies/2007/01/19/nie#";
declare namespace geo = "http://www.w3.org/2003/01/geo/wgs84_pos#";
declare namespace xsd = "http://www.w3.org/2001/XMLSchema#";
let $doc := doc("%doc%")
let $connections := $doc/users_list/users/user

return

for $connection in $connections
let $personId := fn:replace($connection/id, '-', '')
let $fullname := $connection/name
let $nickname := $connection/screen_name
let $location := $connection/location
let $photoUrl := $connection/profile_image_url
let $websiteUrl := if (fn:not(($connection/url)=""))
    then ($connection/url)
    else ()
let $summary := if (fn:not(($connection/description)=""))
    then ($connection/description)
    else ()

#Construct for the user
construct
{
_:c{data($personId)} a nco:PersonContact;
    nco:contactUID {data($personId)};
    nco:photo <{$photoUrl}>;
    nao:description {data($summary)};
    nco:websiteUrl {data($websiteUrl)};
    nco:hasPersonName _pn{data($personId)}.
    _pn{data($personId)} a nco:PersonName ;
    nao:fullname {data($fullname)};
    nco:nickname {data($nickname)}.
}

#Construct for the location of the user
{
    if (fn:not(($location)=""))
        then {
            let $counter := count($location/preceding::*) + count($location/ancestor::*

            construct
            {
                _loc{data($counter)} a nco:PostalAddress;
                nao:prefLabel {data($location)}.
                _c{data($personId)} nco:hasPostalAddress _loc{data($counter)};
            }
        }
        else ()
    }
}

7.2.4 /profile/@me/@all

declare namespace nco = "http://www.semanticdesktop.org/ontologies/2007/03/22/nco#";
declare namespace nie = "http://www.semanticdesktop.org/ontologies/2007/01/19/nie#";
declare namespace xsd = "http://www.w3.org/2001/XMLSchema#";
declare namespace geo = "http://www.w3.org/2003/01/geo/wgs84_pos#";

let $doc := doc("%doc%")

let $persons := $doc/user

return

for $person in $persons
let $personId := fn:replace($person/id, '-', '')
let $fullname := $person/name
let $nickname := $person/screen_name
let $location := $person/location
let $photoUrl := $person/profile_image_url
let $websiteUrl := if (fn:not(($person/url)=""))
    then ($person/url)
    else ()
let $summary := if (fn:not(($person/description)=""))
    then ($person/description)
    else ()
let $interests := $person/suggestions/category

#Construct for the user
construct
{
    _:c{data($personId)} a nco:PersonContact;
nco:contactUID {data($personId)};
nco:photo <{$photoUrl}>;
nao:description {data($summary)};
nco:websiteUrl {data($websiteUrl)};
nco:hasPersonName _pn[data($personId)] .
:pn[data($personId)] a nco:PersonName ;
nco:fullname {data($fullname)};
nco:nickname {data($nickname)}.

#Construct for the interests (suggestions) of the user
{for $interest in $interests
   let $interestName := $interest/name
   construct
      { _:c{data($personId)} nco:hobby {data($interestName)};
   }).

#Construct for the location of the user
{if (fn:not(($location)=""))
   then {
      let $counter := count($location/preceding::*) + count($location/ancestor::*
      construct
         { _:loc{data($counter)} a nco:PostalAddress;
            nao:prefLabel {data($location)}.
            _:c{data($personId)} nco:hasPostalAddress _:loc{data($counter)};
         }
      }
   }
   else ()
}

7.3 Facebook XSPARQL queries
The Facebook XSPARQL queries in the sub-sections below target the retrieval of: i) all the LivePosts within the user's timeline, ii) all the user's personal LivePosts, iii) all the contacts of the user, and iv) the user's profile personal information.

7.3.1 /livepost/@all
declare namespace nco = "http://www.semanticdesktop.org/ontologies/2007/03/22/nco#";
declare namespace dlpo = "http://www.semanticdesktop.org/ontologies/2011/10/05/dlpo#";
declare namespace geo = "http://www.w3.org/2003/01/geo/wgs84_pos#";
declare namespace xsd = "http://www.w3.org/2001/XMLSchema#";
let $doc := doc("%doc%")
let $liveposts := $doc/person/data/e
return
for $livepost in $liveposts
   let $livepostId := fn:replace($livepost/id, '-', '')
   let $timestamp := $livepost/updated_time
   let $message := $livepost/message
   let $type := if ($livepost/type="photo") then "ImagePost"
              else if ($livepost/type="video") then "VideoPost"
              else if ($livepost/type="status") then "Status"
              else if ($livepost/type="link") then "WebDocumentPost"
              else if ($livepost/type="checkin") then "Checkin"
              else if ($livepost/type="event") then "EventPost"
              else ( "LivePost" )
   let $link := $livepost/link
   let $icon := $livepost/icon
   let $label := $livepost/name  #json 'name' element
   let $description := $livepost/description
   let $userId := $livepost/user/id  #changed xml 'from' element to 'user'
   let $userFullname := $livepost/user/name  #changed xml 'from' element to 'user'
#Construct for a LivePost
construct
   { _lvp{data($livepostId)} a dlpo:{data($type)};
nao:externalIdentifier {data($livepostId)};
nao:prefSymbol {data($icon)};
nao:prefLabel {data($label)};
nao:description {data($description)};
dlpo:timestamp {data($timestamp)}^^xsd:dateTime;
dlpo:textualContent {data($message)};
nao:creator _:c{data($userId)}.
_:c{data($userId)} a nco:PersonContact;
nco:contactUID {data($userId)};
nco:hasName _:name{data($userId)}.
_:name{data($userId)} a nco:Name;
_:name{data($userId)} a nco:fullName {data($userFullname)}.

#Construct for the likes of a LivePost
{
let $likes := $livepost/likes/data/e
for $like in $likes
let $userId := $like/id
let $userFullname := $like/name

#Construct for the user's information
construct
{
_:c{data($userId)} a nco:PersonContact;
nco:contactUID {data($userId)};
nco:hasName _:name{data($userId)}.
_:name{data($userId)} a nco:Name;
_:name{data($userId)} a nco:fullName {data($userFullname)}.
_:lvp{data($livepostId)} nao:favouritedBy _:c{data($userId)}.
}
}.

#Construct for the replies (comments) of a LivePost
{
let $comments := $livepost/comments/data/e
for $comment in $comments
let $commentId := $comment/id
let $commentTimestamp := $comment/created_time
let $commentMessage := $comment/message
let $userId := $comment/user/id  #changed xml 'from' element to 'user'
let $userFullname := $comment/user/name  #changed xml 'from' element to 'user'

construct
{
_:com{data($commentId)} a dlpo:Comment;
nao:externalIdentifier {data($commentId)};
dlpo:timestamp {data($commentTimestamp)}^^xsd:dateTime;
dlpo:textualContent {data($commentMessage)};
nao:creator _:c{data($userId)}.
_:c{data($userId)} a nco:PersonContact;
nco:contactUID {data($userId)};
nco:hasName _:name{data($userId)}.
_:name{data($userId)} a nco:Name;
_:name{data($userId)} a nco:fullName {data($userFullname)}.
_:lvp{data($livepostId)} dlpo:hasReply _:com{data($commentId)}.
}
}

7.3.2 /livepost/@me/@all

declare namespace nco = "http://www.semanticdesktop.org/ontologies/2007/03/22/nco#";
declare namespace dlpo = "http://www.semanticdesktop.org/ontologies/2011/10/05/dlpo#";
declare namespace geo = "http://www.w3.org/2003/01/geo/wgs84_pos#";
declare namespace xsd = "http://www.w3.org/2001/XMLSchema#";

let $doc := doc("$doc%")

let $statuses := $doc/person/data/e

return

for $status in $statuses
let $statusId := fn:replace($status/id, '-', '')
let $timestamp := $status/updated_time
let $statusMessage := $status/message

#Construct for the Status Message
construct
{
_:stm{data($statusId)} a dlpo:Status;
  nao:externalIdentifier {data($statusId)};
  dlpo:timestamp {data($timestamp)}^^xsd:dateTime;
  dlpo:textualContent {data($statusMessage)}.
}

#Construct for the user of a Status Message
{
let $userId := $status/user/id   #changed xml 'from' element to 'user'
let $userFullname := $status/user/name   #changed xml 'from' element to 'user'

#Construct for the user's information
construct
{
  _c{data($userId)} a nco:PersonContact;
  nco:contactUID {data($userId)};
  nco:hasName _:name{data($userId)} .
  _:name{data($userId)} a nco:Name ;
  nco:fullname {data($userFullname)}.
  _:stm{data($statusId)} nao:creator _:c{data($userId)}.
}

#Construct for the likes of a Status Message
{
let $likes := $status/likes/data/e
for $like in $likes
  let $userId := $like/id
  let $userFullname := $like/name

  #Construct for the user's information
  construct
  {
    _c{data($userId)} a nco:PersonContact;
    nco:contactUID {data($userId)};
    nco:hasName _:name{data($userId)} .
    _:name{data($userId)} a nco:Name ;
    nco:fullname {data($userFullname)}.
    _:stm{data($statusId)} nao:favouritedBy _:c{data($userId)}.
  }
}

#Construct for the replies (comments) of a Status Message
{
let $comments := $status/comments/data/e
for $comment in $comments
  let $commentId := $comment/id
  let $commentTimestamp := $comment/created_time
  let $commentMessage := $comment/message

  #Construct for the comments of the status message
  construct
  {
    _:com{data($commentId)} a dlpo:Comment;
    nao:externalIdentifier {data($commentId)};
    dlpo:timestamp {data($commentTimestamp)}^^xsd:dateTime;
    dlpo:textualContent {data($commentMessage)}.
    _:stm{data($statusId)} dlpo:hasReply _:com{data($commentId)}.
  }
}

#Construct for the user of a Comment
{
let $userId := $comment/user/id   #changed xml 'from' element to 'user'
let $userFullname := $comment/user/name   #changed xml 'from' element to 'user'

#Construct for the user's information
construct
{
  _c{data($userId)} a nco:PersonContact;
  nco:contactUID {data($userId)};
}
nco:hasName _:name{data($userId)} .
_:name{data($userId)} a nco:Name ;
  nco:fullname {data($userFullname)}.
_:com{data($commentId)} nao:creator _:c{data($userId)}.

} 
} 
}

7.3.3 /person/@me/@all 

let $doc := doc("%doc%")
let $connections :=$doc/connections/friends/data/e
return 

for $connection in $connections
let $connectionId := fn:replace($connection/id, '-', '')
let $fullname := $connection/name
#Construct for the connection 
construct 
{
  _:c{data($connectionId)} a nco:PersonContact;
    nco:contactUID {data($connectionId)};
  nco:hasName _:name{data($connectionId)} .
    _:name{data($connectionId)} a nco:Name ;
      nco:fullname {data($fullname)}.
  }

7.3.4 /profile/@me/@all 

let $doc := doc("%doc%")
let $persons :=$doc/person
return 

for $person in $persons
let $personId := fn:replace($person/id, '-', '')
let $name := $person/first_name
let $surname := $person/last_name
let $fullname := $person/name
let $nickname := $person/username
let $locations := $person/location/name
let $photoUrl := $person/picture
let $summary := $person/bio
let $positions := $person/work/e
let $email := if (fn:exists($person/email))
then ($person/email)
else ("")
let $dob := $person/birthday
let $dobdd := fn:substring($dob,4,2)
let $dobmm := fn:substring($dob,1,2)
let $dobyyyy := fn:substring($dob,$dobyyyy)
let $birthdate := fn:concat(data($dobyyyy),"-",data($dobmm),"-",data($dobdd))
let $gender := if ((fn:contains($person/gender,"female")))
then "Female"
else if ((fn:contains($person/gender,"male")))
then "male"
else ()
let $profileUrl := $person/link
let $location := if (fn:exists($person/location/name))
then ($person/location/name)
else ("")
let $hometown := if (fn:exists($person/hometown/name))
then ($person/hometown/name) 
else ("
let $interests := $person/interests/data/e

#Construct for the user  
construct  
{  
_:c{data($personId)} a nco:PersonContact;  
nco:contactUID {data($personId)};  
nco:photo {data($photoUrl)};  
nco:summary {data($summary)};  
nco:url {data($profileUrl)};  
nco:gender nco:{data($gender)}.

#Construct for the personal details of the user  
{  
let $counter := count($name/preceding::*) + count($name/ancestor::*):

construct  
{  
_:pn{data($counter)} a nco:PersonName;  
nco:nameGiven {data($name)};  
nco:nameFamily {data($surname)};  
nco:fullname {data($fullname)};  
nco:nickname {data($nickname)}.

_:c{data($personId)} nco:hasPersonName _:pn{data($counter)}.
}  

#Construct for the interests of the user  
{  
for $interest in $interests  
let $interestName := $interest/name

construct  
{  
_:c{data($personId)} nco:hobby {data($interestName)};
}

#Construct for the birth date of the user  
{  
if ((fn:string-length($birthdate))=10)
then {
let $counter := count($dob/preceding::*) + count($dob/ancestor::*):

construct  
{  
_:bd{data($counter)} a nco:BirthDate;  
nco:birthDate {data($birthdate)}^^xsd:date.

_:c{data($personId)} nco:hasBirthDate _:bd{data($counter)}.
}  
}  
else ()
}

#Construct for the email of the user  
{  
if (fn:not((Email)=""))
then {
let $counter := count($email/preceding::*) + count($email/ancestor::*):

construct  
{  
_:e{data($counter)} a nco:EmailAddress;  
nco:emailAddress {data($email)}.

_:c{data($personId)} nco:hasEmailAddress _:e{data($counter)};
}  
}  
else ()
}

#Construct for the positions of the user  
{  
for $position in $positions  
let $posId:= $position/position/id  
let $title := $position/position/name  
let $posStartYr := fn:substring($position/start_date,1,4)  
let $posStMo := fn:substring($position/start_date,6,2)  
let $posStartMo := if ((fn:string-length($posStMo))=1)
then (fn:concat("0",$posStMo))

else if ((fn:string-length($posStMo))=0) then ("01") else $posStMo
let $posStart := if (fn:exists($position/start_date)) then (fn:concat(data($posStartYr),"-",data($posStartMo),"-",data("01"),"T","00:00:00Z")) else ()
let $posEndYr := fn:substring($position/end_date,1,4)
let $posEMo := fn:substring($position/end_date,6,2)
let $posEndMo := if ((fn:string-length($posEMo))=1) then (fn:concat("0",data($posEMo))) else if ((fn:string-length($posEMo))=0) then ("01") else $posEMo
let $posEnd := if (fn:exists($position/end_date)) then (fn:concat(data($posEndYr),"-",data($posEndMo),"-",data("01"),"T","00:00:00Z")) else ()
let $counter := count($position/preceding::* ) + count($position/ancestor::* )
let $orgId := $position/employer/id
count
construct
"_,pos{data($counter)} a nco:Affiliation;
nao:externalIdentifier {data($posId)};
nco:role {data($title)};
nco:start {data($posStart)}^^xsd:dateTime;
nco:end {data($posEnd)}^^xsd:dateTime;
nco:org _:org{data($orgId)}.
_:c{data($personId)} nco:hasAffiliation _:pos{data($counter)}.
}
lid
)
}
else ()
else if (fn:not(($location)="")) then ({
let $locationCity := fn:substring-before($location,",")
let $locationCountry := fn:substring-after($location,", ")
let $counter := count($location/preceding::* ) + count($location/ancestor::* )
count
construct
"_,loc{data($counter)} a nco:PostalAddress;
nao:prefLabel {data($location)};
nco:locality {data($locationCity)};
nco:country {data($locationCountry)};
_:c{data($personId)} nco:hasPostalAddress _:loc{data($counter)};
}
lid
)
else ()
else if (fn:not(($hometown)="")) then ({
let $hometownCity := fn:substring-before($hometown,",")
let $hometownCountry := fn:substring-after($hometown,", ")
let $counter := count($hometown/preceding::* ) + count($hometown/ancestor::* )
count
construct
"_,hom{data($counter)} a nco:PostalAddress;
nao:prefLabel {data($hometown)};
nco:locality {data($hometownCity)};
nco:country {data($hometownCountry)};
_:c{data($personId)} nco:hasPostalAddress _:hom{data($counter)};
}
lid
)
else ()
}
7.4 KML XSPARQL query
The KML XSPARQL query in the sub-section below targets the retrieval of placemarks information of a specific custom map as obtained from the user’s Google Maps account.

7.4.1 /place/@all

```
declare namespace nco = "http://www.semanticdesktop.org/ontologies/2007/03/22/nco#";
declare namespace nfo = "http://www.semanticdesktop.org/ontologies/2007/03/22/nfo#";
declare namespace xsd = "http://www.w3.org/2001/XMLSchema#";
declare namespace geo = "http://www.w3.org/2003/01/geo/wgs84_pos#";
declare namespace kml = "http://earth.google.com/kml/2.2";

let $doc := doc("%doc%")
let $placemarks := $doc//@kml:Placemark

return
for $placemark in $placemarks
let $name := $placemark//kml:name
let $description := fn:replace($placemark//kml:description,"<().*?>",""")
let $placemarkId := count($name//preceding::*) + count($name//ancestor::*)

#Construct for the placemarks
construct
{
_:plm{data($placemarkId)} a nfo:Placemark;
nao:externalIdentifier {data($placemarkId)};
nao:prefLabel {data($name)};
nao:description {data($description)}.
}

#Construct for the geo:Point of a placemark
{
let $geoPoints := $placemark//kml:Point//kml:coordinates
let $points := fn:tokenize($geoPoints,"","")
construct
{
_:plm{data($placemarkId)} a geo:Point;
geo:lat {data($points[2])};
geo:long {data($points[1])}.
}
}
```

7.5 OpenSocial XSPARQL queries
The OpenSocial XSPARQL queries in the sub-sections below target the retrieval of: i) all the user’s personal LivePosts, and ii) the user’s profile personal information.

7.5.1 /livepost/@me/@all

```
declare namespace nco = "http://www.semanticdesktop.org/ontologies/2007/03/22/nco#";
declare namespace dlpo = "http://www.semanticdesktop.org/ontologies/2011/10/05/dlpo#";
declare namespace xsd = "http://www.w3.org/2001/XMLSchema#";
declare namespace person = "http://ns.opensocial.org/2008/opensocial";

let $doc := doc("%doc%")
let $status := $doc//@person:status
let $statusId := count($status//preceding::*) + count($status//ancestor::*)

return
#Construct for the status message
if (fn:not(($statusId)=0))
then {
let $statusMessage := $status

construct
{
_:stm{data($statusId)} a dlpo:Status;
nao:externalIdentifier {data($statusId)};
dlpo:textualContent {data($statusMessage)};
nao:creator _:c{data($statusId)}.
}
}
[let $personId := $doc//person:id
let $displayName := $doc//person:displayName

#Construct for the user's information
construct
{
  _c{data($statusId)} a nco:PersonContact;
  nco:contactUID {data($personId)};
  nco:hasPersonName _pn{data($statusId)}.
  _pn{data($statusId)} a nco:PersonName;
  nao:prefLabel {data($displayName)}.
}
}
)

7.5.2 /profile/@me/@all

declare namespace nco = "http://www.semanticdesktop.org/ontologies/2007/03/22/nco#";
declare namespace nie = "http://www.semanticdesktop.org/ontologies/2007/01/19/nie#";
declare namespace geo = "http://www.w3.org/2003/01/geo/wgs84_pos#";
declare namespace person = "http://ns.opensocial.org/2008/opensocial";

#let $doc := doc("$doc")
return
for $person in $doc
let $personId := $person//person:id
let $name := $person//person:name
let $firstname := $name/person:givenName
let $surname := $name/person:familyName
let $fullname := $name/person:formatted
let $honPrefix := $name/person:honorificPrefix
let $honSuffix := $name/person:honorificSuffix
let $middlename := $name/person:middleName
let $nickname := $person//person:alternateNames/person:formatted
let $displayName := $person//person:displayName
let $summary := $person//person:aboutMe
let $addresses := $person//person:addresses
let $emails := if (fn:exists($person//person:emails))
  then ($person//person:emails)
  else ("")
let $ims := $person//person:ims
let $positions := $person//person:organizations
let $phones := $person//person:phoneNumbers
let $photoUrl := $person//person:photos/person:value
let $profileUrl := $person//person:profileUrl
let $thumbnailUrl := $person//person:thumbnailUrl
let $birthdate := $person//person:birthday
let $gender := if ((fn:contains($person//person:gender,"female")))
  then "female"
  else if ((fn:contains($person//person:gender,"male")))
  then "male"
  else ()
let $interests := $person//person:interests
let $note := $person//person:note
let $location := if (fn:exists($person//person:location))
  then ($person//person:location)
  else ("")

#Construct for the user
construct
{
  _c{data($personId)} a nco:PersonContact;
  nco:contactUID {data($personId)};
  nco:photo <{$photoUrl}>;
  nao:description {data($summary)};
  nao:prefSymbol {data($thumbnailUrl)};
  nco:url {data($profileUrl)};
  nco:gender nco:{data($gender)};
  nco:note {data($note)};
  nco:hasPersonName _pn{data($personId)}.
  _pn{data($personId)} a nco:PersonName;
  nao:prefLabel {data($displayName)};
  nco:nameGiven {data($firstname)};
nco:nameFamily {data($surname)};
nco:fullname {data($fullname)};
nco:nameHonorificSuffix {data($honSuffix)};
nco:nameHonorificPrefix {data($honPrefix)};
nco:nameAdditional {data($middlename)};
nco:nickname {data($nickname)}.

#Construct for the interests of the user
{for $interest in $interests
    let $interestName := $interest//person:value
    construct
        {_:c[data($personId)] nco:hobby {data($interestName)};}
}.

#Construct for the birth date of the user
{if ((fn:string-length($birthdate))>0)
    then ({
        let $counter := count($birthdate/preceding::*) + count($birthdate/ancestor::*)
        construct
            {_:bd[data($counter)] a nco:BirthDate;
                nco:birthDate {data($birthdate)}^^xsd:date.
            _:c[data($personId)] nco:hasBirthDate _:bd[data($counter)].
    })
    else ()
}.

#Construct for the email of the user
{for $email in $emails
    let $emailAddress := $email//person:value
    let $counter := count($email/preceding::*) + count($email/ancestor::*)
    construct
        {_:e[data($counter)] a nco:EmailAddress;
            nco:emailAddress {data($emailAddress)}.
        _:c[data($personId)] nco:hasEmailAddress _:e[data($counter)].
}.

#Construct for the phone numbers of the user
{for $phone in $phones
    let $phoneNumber := $phone//person:value
    let $counter := count($phone/preceding::*) + count($phone/ancestor::*)
    construct
        {_:ph[data($counter)] a nco:PhoneNumber;
            nco:phoneNumber {data($phoneNumber)}.
        _:c[data($personId)] nco:hasPhoneNumber _:ph[data($counter)].
}.

#Construct for the instant messenger accounts of the user
{for $im in $ims
    let $imName := $im//person:value
    let $imType := $im//person:type
    let $counter := count($im/preceding::*) + count($im/ancestor::*)
    construct
        {_:im[data($counter)] a nco:IMAccount;
            nco:imAccountType {data($imType)};
            nco:imID {data($imName)}.
        _:c[data($personId)] nco:hasIMAccount _:im[data($counter)].
}.

#Construct for the positions of the user
{for $position in $positions
    let $title := $position//person:title

let $posStart := if (fn:exists($position//person:startDate))
then ($position//person:startDate)
else ()
let $posEnd := if (fn:exists($position//person:endDate))
then ($position//person:endDate)
else ()
let $department := $position//person:department
let $description := $position//person:description
let $counter := count($position/preceding::*) + count($position/ancestor::*

construct
{ 
_:pos{data($counter)} a nco:Affiliation;
nao:externalIdentifier {data($counter)};
nao:description {data($description)};
nco:role {data($title)};
nco:start {data($posStart)}^^xsd:dateTime;
nco:end {data($posEnd)}^^xsd:dateTime;
nco:department {data($department)};
   nco:org _:org {data($counter)}.
_:c{data($personId)} nco:hasAffiliation _:pos{data($counter)}.
} 

let $company := $position//person:name
construct
{ 
_:org{data($counter)} a nco:OrganizationContact;
nie:title {data($company)}. 
}
}

#Construct for the address of the user
{
for $address in $addresses
let $street := $address//person:streetAddress
let $country := $address//person:country
let $locality := $address//person:locality
let $postcode := $address//person:postalCode
let $region := $address//person:region
let $counter := count($address/preceding::*) + count($address/ancestor::*

construct
{ 
_:addr{data($counter)} a nco:PostalAddress;
nao:prefLabel {data($fullAddress)};
nco:locality {data($locality)};
nco:country {data($country)};
nco:postalcode {data($postcode)};
nco:region {data($region)};
nco:streetAddress {data($street)};
nco:addressLocation _:plm{data($counter)}.
_:c{data($personId)} nco:hasPostalAddress _:addr{data($counter)}.
} 

#Construct for the geo:Point of a placemark
{
let $longitude := $address//person:longitude
let $latitude := $address//person:latitude

construct
{ 
_:plm{data($counter)} a geo:Point;
geo:lat {data($latitude)};
geo:long {data($longitude)}.
}
}
.
}

#Construct for the location of the user
{
if (fn:not({$location}=""))
then {
  let $counter := count($location/preceding::*) + count($location/ancestor::*

  construct
  { 
  _:loc{data($counter)} a nco:PostalAddress;
  nao:prefLabel {data($location)}.
  _:c{data($personId)} nco:hasPostalAddress _:loc{data($counter)};
  }
}

7.6 Fitbit XSPARQL queries

The Fitbit XSPARQL queries in the sub-sections below target the retrieval of: i) all the contacts of the user, ii) the user’s profile personal information and iii) the user’s recent activities.

7.6.1 /person/@me/@all

declare namespace nco = "http://www.semanticdesktop.org/ontologies/2007/03/22/nco#";
declare namespace nie = "http://www.semanticdesktop.org/ontologies/2007/01/19/nie#";
declare namespace xsd = "http://www.w3.org/2001/XMLSchema#";
declare namespace geo = "http://www.w3.org/2003/01/geo/wgs84_pos#";

let $doc := doc("%doc%")
let $friends := $doc/result/friends/friend/user
return
for $friend in $friends
let $friendId := count($friend/preceding::*) + count($friend/ancestor::*
let $summary := $friend/aboutMe
let $thumbnailUrl := $friend/avatar
let $birthdate := $friend/dateOfBirth
let $displayName := $friend/displayName
let $fullname := $friend/fullName
let $gender := if ((fn:contains($friend/gender,"FEMALE")) then "female"
else if ((fn:contains($friend/gender,"MALE")) then "male"
else if ((fn:contains($friend/gender,"NA")) then "NA"
else ()
let $nickname := $friend/nickname

#Construct for the user
construct
{ 
  _:c{data($friendId)} a nco:PersonContact;
  nco:contactUID {data($friendId)};
  nao:description {data($summary)};
  nao:prefSymbol {data($thumbnailUrl)};
  nco:hasPersonName _:pn{data($friendId)}.
  _:pn{data($friendId)} a nco:PersonName;
  nao:prefLabel {data($displayName)};
  nco:fullname {data($fullname)};
  nco:nickname {data($nickname)}.
}

#Construct for the gender of the user
{ 
  if (fn:not((?gender)="NA"))
    then ( 
      let $counter := $friendId
      construct
      { 
        _:c{data($friendId)} nco:gender nco:{data($gender)}.
      } )
    else ()
  }.

#Construct for the birth date of the user
{ 
  if ((fn:string-length($birthdate))>0)
    then ( 
      let $counter := count($birthdate/preceding::*
      let $dob := fn:concat(data($birthdate),'T','00:00:00Z')
      construct
      { 
        _:bd{data($counter)} a nco:BirthDate;
        nco:birthDate {data($dob)}^^xsd:date.
        _:c{data($friendId)} nco:hasBirthDate _:bd{data($counter)}.
      } )
    else ()
  }.
nco:birthDate {data($dob)}^^xsd:date.
    _:bd{data($counter)} nco:hasBirthDate _:_bd{data($counter)}.
  }]]}} else ()
}

7.6.2 /profile/@me/@all

declare namespace nco = "http://www.semanticdesktop.org/ontologies/2007/03/22/nco#";
declare namespace nie = "http://www.semanticdesktop.org/ontologies/2007/01/19/nie#";
declare namespace xsd = "http://www.w3.org/2001/XMLSchema#";
declare namespace geo = "http://www.w3.org/2003/01/geo/wgs84_pos#";

let $doc := doc("%doc%")
let $persons := $doc/result/user
return
for $person in $persons
  let $personId := count($person/preceding::*) + count($person/ancestor::*)
  let $summary := $person/aboutMe
  let $thumbnailUrl := $person/avatar
  let $birthdate := $person/dateOfBirth
  let $displayName := $person/displayName
  let $fullName := $person/fullName
  let $gender := if ((fn:contains($person/gender,"FEMALE"))) then "female"
    else if ((fn:contains($person/gender,"MALE"))) then "male"
    else if ((fn:contains($person/gender,"NA"))) then "NA"
    else ()
  let $nickname := $person/nickname

#Construct for the user
construct

let $doc := doc("%doc%")
let $activities := $doc/recentActivities/loggedActivityReference

return
for $activity in $activities
    let $activityId := $activity/activityId
    let $calories := $activity/calories
    let $description := $activity/description
let $distance := $activity/distance
let $duration := $activity/duration
let $name := $activity/name

let $activityName := if ($activity/name="Shopping") then "Shopping"
else if ($activity/name="Sleeping") then "Sleeping"
else if ($activity/name="Bicycling") then "Bicycling"
else if ($activity/name="Running") then "Running"
else if ($activity/name="Swimming") then "Swimming"
else if ($activity/name="Walking") then "Walking"
else if ($activity/name="Squash") then "SportActivity"
else if ($activity/name="Soccer, casual, general") then "SportActivity"
else if ($activity/name="Riding in a bus") then "RidingInABus"
else (fn:replace($activity/name, ",;" , "," ))

"SportActivity"

let $activity := if ($name="Shopping") then "OtherActivity"
else if ($name="Shopping") then "OtherActivity"
else if ($name="Sleeping") then "OtherActivity"
else if ($name="Bicycling") then "OtherActivity"
else if ($name="Running") then "OtherActivity"
else if ($name="Swimming") then "OtherActivity"
else if ($name="Walking") then "OtherActivity"
else if ($name="SportActivity") then "OtherActivity"
else if ($name="RidingInABus") then "Travelling"
else ("Activity")

let $uri := if ($activity = "Activity") then "fitbit"
else ("dpo")

#Construct for the activity
construct
{
{data($uri)}:{data($activityName)} a dpo:{data($activity)};
nao:externalIdentifier {data($activityId)};
nao:prefLabel {data($name)};
nao:description {data($description)};
dcon:caloriesExpended {data($calories)}^^xsd:nonNegativeInteger;
dcon:distanceCovered {data($distance)}^^xsd:nonNegativeInteger;
dcon:duration {data($duration)}^^xsd:nonNegativeInteger.
}