NoTube
Networks and Ontologies for the Transformation and Unification of Broadcasting and the Internet

FP7 – 231761

D3.1 User and Context model Specification

Coordinator: Davide Palmisano, Michele Minno
With contributions from: Vicky Buser, Dan Brickley, Libby Miller, Balthasar Schopman
Quality Assessor: Chris van Aart, Véronique Malaisé
Quality Controller: Libby Miller

<table>
<thead>
<tr>
<th>Document Identifier:</th>
<th>NoTube/2009/D3.1/1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class Deliverable:</td>
<td>NoTube EU-IST-2009-231761</td>
</tr>
<tr>
<td>Version:</td>
<td>1.1</td>
</tr>
<tr>
<td>Date:</td>
<td>February 22, 2010</td>
</tr>
<tr>
<td>State:</td>
<td>Final</td>
</tr>
<tr>
<td>Distribution:</td>
<td>Public</td>
</tr>
</tbody>
</table>
EXECUTIVE SUMMARY

This document is the second version of D3.1, the first version of which was delivered in month 3. This version expands on that document and specifies in more detail the user and context models. It includes RDF schemas weighted user preferences, and activities data, and describes our use of the Open Social APIs as a standard means of accessing this data. It describes in detail the current implementation of Beancounter, an application that harvests activity data from the Web and aggregates and processes it to produce the machine-processible RDF profile for the user, which can be used for recommending content to them. It discusses how we might use OAuth to facilitate data security and privacy in future implementations.
# Document Information

<table>
<thead>
<tr>
<th>IST Project Number</th>
<th>Acronym</th>
<th>Full Title</th>
<th>Project URL</th>
<th>Document URL</th>
<th>EU Project Officer</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP7 – 231761</td>
<td>NoTube</td>
<td>Networks and Ontologies for the Transformation and Unification of Broadcasting and the Internet</td>
<td><a href="http://www.notube.tv/">http://www.notube.tv/</a></td>
<td></td>
<td>Leonhard Maqua</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deliverable Number</th>
<th>Title</th>
<th>Work Package Number</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>User and Context model Specification</td>
<td>3</td>
<td>User Profiling and Recommendations</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date of Delivery</th>
<th>Contractual</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M13</td>
<td>28-02-2010</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th>Nature</th>
<th>Dissemination Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>prototype</td>
<td>public</td>
</tr>
<tr>
<td></td>
<td>report</td>
<td>consortium</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Authors (Partner)</th>
<th>Resp. Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>Davide Palmisano (Asemantics), Michele Minno (Asemantics), Vicky Buser (BBC), Libby Miller (BBC), Dan Brickley (VU), Balthasar Schopman (VU)</td>
<td>Davide Palmisano (Asemantics)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E-mail</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:davide@asemantics.com">davide@asemantics.com</a></td>
<td>+39 0645472503</td>
</tr>
</tbody>
</table>

**Abstract (for dissemination)**

This document describes the requirements for the User and Context model, the RDF schemas used, and an implementation of Bean-counter, an application that harvests activity data from the Web and aggregates and processes it to produce a machine-processible RDF profile for the user, which can be used for recommending content to them.

**Keywords**

FOAF, Semantic Web, profiler, recommendations, TV, activities, activity streams

**Version Log**

<table>
<thead>
<tr>
<th>Issue Date</th>
<th>Rev No.</th>
<th>Author</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>February 22, 2010</td>
<td>1</td>
<td>Libby Miller</td>
<td>2.1</td>
</tr>
</tbody>
</table>
# Project Consortium Information

<table>
<thead>
<tr>
<th>Participant's name</th>
<th>Partner</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vrije Universiteit Amsterdam</td>
<td></td>
<td>Guus Schreiber</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Phone: +31 20 598 7739/7718</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:schreiber@cs.vu.nl">schreiber@cs.vu.nl</a></td>
</tr>
<tr>
<td>British Broadcasting Corporation</td>
<td>Libby Miller</td>
<td>Phone: +44 787 65 65 561</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:Libby.Miller@bbc.co.uk">Libby.Miller@bbc.co.uk</a></td>
</tr>
<tr>
<td>Asemantics SRL Uninominale</td>
<td>Alberto Reggiori</td>
<td>Phone: +39 0639 7510 78</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:alberto@asemantics.com">alberto@asemantics.com</a></td>
</tr>
<tr>
<td>Engin Medya Hizmetleri A.S.</td>
<td></td>
<td>Ron van der Heiden</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Phone: +31 6 2003 2006</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:ron@engin.tv">ron@engin.tv</a></td>
</tr>
<tr>
<td>Institut fuer Rundfunktechnik GmbH</td>
<td>Christoph Dosch</td>
<td>Phone: +49 89 32399 349</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:dosch@irt.de">dosch@irt.de</a></td>
</tr>
<tr>
<td>Ontotext AD</td>
<td>Atanas Kiryakov</td>
<td>Phone: +35 928 091 565</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:naso@sirma.bg">naso@sirma.bg</a></td>
</tr>
<tr>
<td>Open University</td>
<td>John Domingue</td>
<td>Phone: +44 1908 655 014</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:j.b.domingue@open.ac.uk">j.b.domingue@open.ac.uk</a></td>
</tr>
<tr>
<td>RAI Radiotelevisione Italiana SPA</td>
<td>Alberto Morello</td>
<td>Phone: +39 011 810 31 07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:a.morello@rai.it">a.morello@rai.it</a></td>
</tr>
<tr>
<td>Semantic Technology Institute International</td>
<td>Lyndon Nixon</td>
<td>Phone: +43 1 23 64 002</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:lyndon.nixon@sti2.org">lyndon.nixon@sti2.org</a></td>
</tr>
<tr>
<td>Stoneroos B.V.</td>
<td>Annelies Kaptein</td>
<td>Phone: +31 35 628 47 22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: annelies.kaptein@stoneroos</td>
</tr>
<tr>
<td>Thomson Grass Valley France SA</td>
<td>Raoul Monnier</td>
<td>Phone: +33 2 99 27 30 57</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:raoul.monnier@thomson.net">raoul.monnier@thomson.net</a></td>
</tr>
<tr>
<td>TXT Polymedia SPA</td>
<td>Sergio Gusmeroli</td>
<td>Phone: +39 02 2577 1310</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:sergio.gusmeroli@txtgroup.com">sergio.gusmeroli@txtgroup.com</a></td>
</tr>
<tr>
<td>KT Corporation</td>
<td>Myoung-Wan Koo</td>
<td>Phone: +82 2 526 6347</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:mskim@kt.co.kr">mskim@kt.co.kr</a></td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS

LIST OF FIGURES ............................................. 7

LIST OF TABLES ............................................. 8

1 INTRODUCTION ............................................. 9
   1.1 Goals of this Workpackage ................................. 9
   1.2 Context: Privacy, Security and Usefulness of a Sea of Personal Data ... 10
   1.3 Linked Data and Recommendations ............................ 10

2 PROFILER REQUIREMENTS ................................. 12
   2.1 Requirements from WP7a .................................. 12
   2.2 Requirements from WP7b .................................. 12
   2.3 Requirements from WP7c .................................. 13
   2.4 Recommendations ...................................... 14
       2.4.1 Behaviour Based Recommenders ....................... 14
       2.4.2 Linked-data-based recommendations on Beancounter ...... 14
   2.5 Summary ............................................. 15

3 STATE OF THE ART IN USER MODELING .................. 16
   3.1 User Modeling Systems .................................. 16
       3.1.1 Collecting ....................................... 17
       3.1.2 Profiling ....................................... 18

4 THE BEANCOUNTER APPLICATION .......................... 21

5 IMPLEMENTATION .......................................... 23
   5.1 Architecture .......................................... 23
       5.1.1 Gathering raw data ................................ 24
       5.1.2 Enriching data ................................... 26
       5.1.3 Reasoning about data .............................. 27
       5.1.4 Modeling / exposing data ........................... 28
   5.2 The Open Social API ................................... 28
   5.3 Data flows ........................................... 28
       5.3.1 From different social networks to aggregated activities in RDF . 29
       5.3.2 Identity resolution as a pipe inside a pipeline ........... 30
       5.3.3 Rule-based simple entailment .......................... 30
       5.3.4 Reasonlet activation to produce user profiles ........... 31
       5.3.5 Accessing user interests by OpenSocial .................. 32
   5.4 Interfaces for administrators and users ................... 32
       5.4.1 Administrators .................................... 32
       5.4.2 Users ........................................... 32

6 USER AND CONTEXT MODELS ............................... 39
   6.1 The Layered Common Knowledge and User Model .................... 39
       6.1.1 The Layer Cake .................................... 39
       6.2 User Profile Model and Schemas .......................... 41
   6.3 Activities Model ....................................... 43
7 PRIVACY AND DATA SECURITY
7.1 Privacy .................................................. 46
  7.1.1 Privacy Issues Around Data Aggregation ............ 47
  7.1.2 Privacy Issues Around Linking and Data ‘Enhancement’ .. 48
  7.1.3 Privacy Issues Around Sharing ......................... 49
  7.1.4 Supporting User Control ............................ 50
  7.1.5 Privacy: Conclusions .............................. 51
7.2 Data Access Controls: OAuth and OpenSocial ........... 51

8 FUTURE WORK

REFERENCES

A RDF SCHEMAS
A.1 Atom Activities in RDF .................................. 56
A.2 Weighted Interests ................................. 62
LIST OF FIGURES

3.1 High-level information flow of a generic user modeling system 16

5.1 The highest level picture: the user has a profile which can be used to filter or recommend content to the user 23

5.2 At a glance Beancounter architecture 24

5.3 Pipelines - default and combinations 34

5.4 Last.fm response 35

5.5 Glue response 35

5.6 Last.fm triples 36

5.7 Last.fm graph 36

5.8 Glue triples 36

5.9 Glue graph 37

5.10 Integrated graph 37

5.11 OpenSocial results 38

6.1 A two-layer layer cake 39

6.2 A diagram of the weighted interests vocabulary 44

6.3 A diagram of the activities vocabulary 44

7.1 A picture of the Beancounter should NOT do in a public profile 49
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>User Profiling Requirements from WP7a</td>
<td>12</td>
</tr>
<tr>
<td>2.2</td>
<td>User Profiling Requirements from WP7b</td>
<td>12</td>
</tr>
<tr>
<td>2.3</td>
<td>User Profiling Context Requirements from WP7b</td>
<td>13</td>
</tr>
<tr>
<td>2.4</td>
<td>User Profiling Requirements from WP7c</td>
<td>13</td>
</tr>
<tr>
<td>2.5</td>
<td>Profiler UI and Privacy Requirements from WP7c</td>
<td>14</td>
</tr>
<tr>
<td>2.6</td>
<td>Behaviour-based Recommender Requirements on Beancounter</td>
<td>14</td>
</tr>
<tr>
<td>2.7</td>
<td>Profiler Recommender Requirements</td>
<td>15</td>
</tr>
</tbody>
</table>
1. Introduction

Workpackage 3 is primarily engineering workpackage, and like the other engineering workpackages, is based on the requirements outlined in the Use Cases of Workpackage 7. This deliverable is the User and Context Model Specification for NoTube. That is, it:

- Outlines the requirements for the User and Context Model from the user case workpackages WP7s a, b and c
- Describes the state of the art in user and context modeling
- Describes the current implementation and future work
- Describes user and context models to fulfill the WP7 requirements

Most of the material is new to this version of the deliverable, although we refer back to relevant sections of the previous version in each section.

1.1 Goals of this Workpackage

From page 18 of the Technical Annex for the NoTube project:

This workpackage aims at providing novel techniques for the acquisition, the representation and the interpretation of user profiles and context models; as well as define appropriate ways for the presentation of user and context profiles for single and groups of users. The main goal is to bring the current-state-of-the-art of user modelling and personalized information presentation several steps further to meet the demands of interactive, distributed and multi-device environments (e.g. cross-context user profiles, sharing of user data, representation of user profiles on different devices). Further, the objective of this work package is to develop hybrid (symbolic and statistical) privacy-preserving algorithms for content filtering. Based on this we aim at providing services for content filtering and recommendation for both single users and communities.

This document explains our approach to these questions of novel techniques for the acquisition of user profiles, and describes how we represent user profiles as novel additions to the FOAF RDF vocabulary including representations of contexts. It also describes some of the general requirements from the NoTube usecase workpackages and specifically the impact of profile-based recommendations on what a user profile must contain.

The document then documents in detail our implementation of Beancounter, an application that harvests activity data from the Web, and aggregates and processes it to produce a machine-processible profile for the user, which can be used for recommending content to them. Finally, we discuss issues of privacy, authorisation and usability and describe future work.
1.2 Context: Privacy, Security and Usefulness of a Sea of Personal Data

The Web is awash with public and private user activity traces, which we term Activity Data. These can be automatically generated as a byproduct of user activity or manually created, and can include implicit or explicit expressions of user preferences. For example, what the user has watched, enjoyed, hated, took the time to review, or talked about online, and potentially what the user has bought. This activity data is generated and output by a variety of social media and content sites, such as Twitter, Facebook and YouTube.

There will probably never be a global TV network, because social, business and political concerns align to ensure there will always be multiple parties responsible for creating, aggregating and providing access to video and audio materials online. However, this sea of activity data both talks about media objects such as TV programmes, and also goes beyond the boundaries of individual companies or countries, because it exists on the Web. In many cases the activity data exists in a fashion that can be harvested and has some semantics, for example such data can be an Atom feed with links to online metadata representations of the TV content it refers to such as wikipedia or IMDB links. Part of the NoTube aim is to contribute to this virtual and decentralised network of users and their contributions, spanning all modern TV systems touched by the Web. In this section of the project we use existing parts of this network.

The novel technique we have used to acquire user profiles, then, is to data-mine these examples of Activity Data in order to generate a profile for the user. The approach we have taken is to put the user in the driving seat by allowing them to manage the input and combination of these datasources to allow them to create a profile that represents their preferences, based on their actual activity but sufficiently privacy-preserving and realistic to them to be useful for recommendations. We are also examining security aspects of gathering and aggregating such useful and potentially valuable information.

The separation of profile and recommendations in this way allows us to do several useful things:

- Swap in different or better profilers as they become available
- Swap in recommendation engines as they become available
- Keep the user in the driving seat, able to manage and control their online profile rather than personal data records solely kept privately by organisations for commercial use.

1.3 Linked Data and Recommendations

One of the outputs of this workpackage is research in applying complex user profiles to linked data content filters to provide recommendations to the user.

Linked Data is rather like putting multiple databases on the web and giving each item of interest a globally unique key that allows you to make links between different databases. In the Linked Data world, these keys are also dereferencable URLs, which
can give more machine-readable information about themselves when fetched, such as what they are connected to, what sort of a thing they are and what properties or attributes they have. The items of interest can be anything - people, documents, places, pictures, videos, anything that can be identified.

Using Linked Data enables interesting links to be made between items in a partially automated way. For example by combining MusicBrainz with DBpedia, which is derived from Wikipedia, we may be able to find out that a piece of music is related in an interesting way to another piece of music. Some of the links between the two pieces of music might be very uninteresting (‘both artists were American’), but we might find something interesting by linking up these datasets, (such as ‘both artists worked in Detroit in the 1960s’). This is a way to suggest content of interest to someone who has just listened to music by the first artist, or to provide more information of interest about the first artist (for example as part of the ‘Do you want to know more’ scenario). The Music Bore\(^1\) and dbrec\(^2\) are examples of using this technique without user profiles.

Our hypothesis is that a profile for a user stating that she is interested in particular things (such as a particular kind of music) will allow useful personalised content filters to be created, by narrowing the types of linked data connections that should be followed for this user. Further, if the user profile can state in a machine-processible fashion the preferences of the user over multiple interests in context then a smart linked data filter should be able to order the suggestions in priority of interestingness.

One of the nice features of this approach is that cross-domain recommendations can be produced, for example a TV programme could be used to recommend related music, or related documents or books. Another useful feature is that this technique can provide contentful explanations of why the user might be interested in a piece of content, as opposed to the ‘black box’ explanations of Amazon-style collaborative filtering techniques (e.g. ‘You might like Y because other people who bought X also bought Y’).

\(^1\)http://www.bbc.co.uk/blogs/radiolabs/2009/07/the_music_bore.shtml
\(^2\)http://dbrec.net/
2. Profiler Requirements

The structure of the NoTube project is that the engineering workpackages are driven by the practical usecases developed in workpackage 7. Here we outline these as they currently stand and draw out some conclusions for the kinds of properties that a user profile should contain.

2.1 Requirements from WP7a

D7.1a describes a series of scenarios involving personalized news for a family whose home is networked: their home knows how each person is moving inside the different rooms of the house and which devices people are using. The family also uses networked devices outside the home.

The usecases involve

- Bookmarking of half-watched news terms
- Automatically recommended news to specific individuals
- The creation of recommendations to specific people
- The creation of personalised news alerts
- Delivery of content to any platform

The scenarios from WP7a provide us with the following requirements on user profiles:

| R1.1 | Must be able to identify and authenticate the user |
| R1.2 | Must provide user interests as categories |
| R1.3 | User Profiling Context must deliver current device |

Table 2.1: User Profiling Requirements from WP7a

2.2 Requirements from WP7b

D7.1b describes various scenarios involving the delivery of appropriate advertising to a TV or device in specific contexts. The profiling aspect of the Beancounter is directly relevant, and WP7b generates the following requirements for user profiles:

| R2.1 | Must be able to authenticate the user |
| R2.2 | Must include age if available (age group) |
| R2.3 | Must include gender if available |
| R2.4 | Must include language if available |
| R2.5 | Must include favorite programme list if available |
| R2.6 | Must include interests list if available |

Table 2.2: User Profiling Requirements from WP7b

User Profiling Context should include
<table>
<thead>
<tr>
<th>R3.1</th>
<th>Time of day (dinnertime, evening)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3.2</td>
<td>Day of week (Saturday, Monday)</td>
</tr>
<tr>
<td>R3.3</td>
<td>Time of year (summer, winter)</td>
</tr>
<tr>
<td>R3.4</td>
<td>Device, multimodal capabilities</td>
</tr>
<tr>
<td>R3.5</td>
<td>Mood, feelings</td>
</tr>
</tbody>
</table>

Table 2.3: User Profiling Context Requirements from WP7b

2.3 Requirements from WP7c

D7c.1 describes three scenarios, the first of which is directly relevant to the user profile:

**Scenario 1: Recommendations for me on my TV based on my web behaviour**

Jana wants to see recommendations based on her social activity on her TV when she gets home at night. She and her friends talk a lot on Twitter and Facebook about what they watch on TV in the context of their online social lives, and Jana doesn’t see why she should have to explicitly tell any system what her preferences are. She wants to see recommendations clearly featured on the user interface of her set top box.

This scenario is the only one of the three in 7c to make explicit use of the Bean-counter profiler, and coupled with security and privacy considerations (see below) gives rise to the following requirements:

**General considerations:**

<table>
<thead>
<tr>
<th>R4.1</th>
<th>Must allow users to create accounts</th>
</tr>
</thead>
<tbody>
<tr>
<td>R4.2</td>
<td>Must allow users to add and remove social data accounts to their Bean-counter account</td>
</tr>
<tr>
<td>R4.3</td>
<td>Must output an RDF profile (in the format described in the Appendix below)</td>
</tr>
<tr>
<td>R4.4</td>
<td>Must provide access to the raw activities data</td>
</tr>
<tr>
<td>R4.5</td>
<td>Must provide access to the aggregated activities data</td>
</tr>
<tr>
<td>R4.6</td>
<td>Must provide secure access to any private data</td>
</tr>
</tbody>
</table>

Table 2.4: User Profiling Requirements from WP7c

**UI and Privacy Considerations:**
R5.1  By default all Profile data should be private until the user actively chooses to share it

R5.2  The user can only ‘share’ their Bean-counter Profile from the screen where they can see their Profile, To protect the user from sharing things inadvertently which they may prefer to keep private.

R5.3  Any updates or changes to the Profile are private by default

R5.4  Users must be able to edit their Profiles at any time (e.g. to delete / hide / strengthen / weaken specific topics)

Table 2.5: Profiler UI and Privacy Requirements from WP7c

2.4 Recommendations

2.4.1 Behaviour Based Recommenders

The separation of activity data and recommendations by the intervening user profile sets certain conditions on and requirements for the future recommendations engine. While there are advantages to this novel two-step approach (user privacy improvements, portability of the profile, ability to swap in different profile-based recommendations algorithms), this approach does exclude behaviour-based recommendations algorithms based directly on activities data. Therefore we have the following requirement:

R6.1  Retain and make available the raw activities data In order to be able to test content filtering recommenders against other types.

Table 2.6: Behaviour-based Recommender Requirements on Beancounter

2.4.2 Linked-data-based recommendations on Beancounter

Some requirements for the Beancounter and profile come from this planned approach:
### R7.1
Rationales have to be derivable from the information (i.e. the profile) made available to the recommender

<table>
<thead>
<tr>
<th>R7.1</th>
<th>From the recommender requirement: output a list of URIs with channels, dates of broadcast or availability details, and rationales (explanations of why something was recommended).</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>R7.2</th>
<th>The profile may provide preference orderings</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>For example, if there are multiple matches for recommended content, a linked data recommender may choose to provide an ordered list. It may get hints about preferences from the user profile.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R7.3</th>
<th>Generally, the profile must output enough information for recommendations to be made</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>i.e. the recommender is assumed to be a separate entity to the Beancounter.</td>
</tr>
</tbody>
</table>

**Table 2.7: Profiler Recommender Requirements**

### 2.5 Summary

Analysing these requirements we can see several themes concerning the **profile**:

- Traditional (age, gender, location)
- Interests (preferred programmes, series, brands, genres, contributors, other interests)
- Contextual information (device, time, current location)
- Preferences (orderings over interests in context)

and several more to do with the **Beancounter application**

- Authentication: security and privacy
- Identification: identifying the user
- Data access: profile in a linked data format, sufficient for recommendations to be made
- Data access: raw activity data available separately

Given these requirements for the profile and Beancounter, a literature survey was performed to find out what part of existing systems, resources and methods could be reused or applied in the NoTube context.
3. State of the Art in User Modeling

This brief chapter intends to provide an introduction on generic User Modeling Systems and a short overview about existing User Profiling systems that leverage Semantic Web technologies.

3.1 User Modeling Systems

Informally, a User Modeling System (UMS) is a software system responsible for producing, maintaining and making accessible information about knowledge, beliefs, interests and goals of the user. These pieces of information are then used to perform some content filtering operations on the resources accessed by the user. User modeling techniques are generally described with the term ‘Personalization’ (the term ‘Adaptive Web’ is also used), and they have found in the Web a huge battlefield [4]. Since personalization has been proved to be one of the most effective techniques to lower the information overload [11], the problem of building, managing and representing user characteristics has been growing into one of the main key issues.

Generally speaking, the main goal of a UMS is to gather information about subjects the user is interested in and about her behavior during the consumption of such content. These information will then grow into a formal user description snapshot, that can be used by other applications or services to enhance the user satisfaction when finding and consuming content.

Two UMSs can heavily differ in terms of technologies and methodologies they use to achieve the three depicted envisioned phases (represented with yellow arrows in Figure 3.1). Informally, we refer to those phases as:

- Collecting: the raw data about the user are collected. This phase strongly influences next phases in terms of what kind of user information could be represented.
- Profiling: the gathered raw data are represented as instances of a suitable user model. The whole user model life-cycle is addressed within this stage: building,
maintaining, updating the user model, as well as all that is needed to keep the profile quality and consistence, are performed within this step.

- Exporting: the generated user profile is made suitable for an external application. The application will make use of it in order to produce personalized content delivery.

In the following subsections the first two phases will be briefly described and some references to systems that better embody some interesting peculiarities will be given.

### 3.1.1 Collecting

The unique identification of the user is a crucial aspect of any system that aims to model the characteristics of individuals. Lots of work has been done in the past to find different solutions to this issue, like user login, cookies and http sessions. Even if such approaches could be considered stable and efficient, especially for those services which are exposed on the Web, such systems are related to a conception of the Web that is slightly out-of-date. Actually the emerging tendencies, informally described as the “Web of Applications”, lead to a ‘balkanized’ [19] scenario where a user interacts with several data repositories, keeping at the same time her identity and personal information unique and portable. For these reasons, and regarding the importance of the user identification within the user modeling, identity-based protocols as OpenID\(^1\) could play here an important role. An interesting scenario in this case could be the collection of a subset of a user profile using RDFa\(^2\) in her OpenID Identity Page [15].

Having found a proper way to identify the user, we now move to the user information collection itself. All the existing techniques can be roughly described as explicit or implicit:

**Explicit** - Some user active contribution is required in order to produce her profile. Users typically express their interests and set their general demographic condition filling some HTML text fields. Two major drawbacks of this approach are the user time consumption to produce these information and the tight relation between the quality of the provided information and the consistence of the resulting profile. In the social networking domain [6, 7], the business model is often based on the delivery of personalized advertisements, thus the quality and the coherence of user information is then the main issue. Moreover, if we consider that social networking web applications are basically repositories of user profiles, possibly communicating each other leveraging data portability [1], bad data inserted by the user in a repository of those will impact on the whole set of repositories this information is forwarded to.

Another important drawback of the explicit information gathering technique resides in the potential obsolescence of the provided data and in their required disambiguation. For sake of simplicity, let us imagine a user that expresses her music interest providing a list of rock bands she likes at the very moment she fills in the form. What happens if the user changes her tastes without updating her profile? Another problem is that user-provided information needs

\(^1\)http://openid.net

\(^2\)http://www.w3.org/TR/xhtml-rdfa-primer
a disambiguation step, especially if such data are expressed as a list of keywords: the same keyword could have different meanings.

**Implicit** - Data collection is performed without any active effort by the user. Several surveys could be found in literature covering a lot of techniques, ranging from logging the web searches of the user to using proxy servers that intercept the user browsing activity [12, 20]. Again, even if their feasibility has been formally proven, such attempts belong to a pre-social vision of the Web. Such methods do not take into account the hidden potential of extracting useful information about users from social networks, as Activity Streams. Informally, an activity stream is the flow of events that a user generates within the social application [1]. Likewise, these pre-social methods ignore the importance of social interactions that a user could have with other users on her friend list. It might be important to underline here that this phase, even if sometimes called “implicit user feedback collection”, does not foresee methods which collect the user feedback of a content recommendation. We are instead interested, for instance, in the Facebook “I like this” feature [21], because it is an activity itself rather than feedback, and thus adds value to the activity stream of the user. The importance of activity streams within our user model is justified in the following sections of this document.

Many studies aim to identify which type of data collection method (explicit vs implicit) gives the best results. Even if the main works on that [23, 24] show contrasting conclusions, an improved implicit approach has been recognized as the latest trend.

### 3.1.2 Profiling

The second phase of every UMS aims to construct and represent gathered user information according to a certain model. Traditionally, consolidated user modeling representations fall under these four main typologies:

**Stereotypical Profiles** - The user characteristics are modeled as a set of rules and facts (often following first order logic). Some earlier systems (roughly) follow the Collecting-Profiling-Exporting information flow pattern, but they are characterized by an extremely low level of modularity in their architecture: such systems are not re-usable and not general purpose [13]. The first attempt to make a general purpose UMS that could be used across different domains is represented by Tim Finin’s GUMS [9]. This work, although developed without the opportunities coming from the Social Web, has laid the groundwork for all the further developed systems, its design choices and principles still impacting on the most up-to-date solutions. Other proposed architectures are heavily influenced by GUMS, and share the same modeling principles about the user behavior and interests.

**Keyword based** - Each keyword can be directly provided by the user or automatically extracted from the content she consumes. The main idea is that each keyword aims to represent a topic of interest to the user. Basic construction methods are, mainly, derived from the classical information retrieval theory tf*idf weighting, LSI and LLIF [8, 2]: . As already discussed, keyword-based profiles have as major drawback their potential polysemy. From this point of view, all the
followed methodologies could be considered as an attempt to overcome and solve problems related to polysemy. But, on the other hand, these kind of approaches reveal all their potential in such scenarios where the content to be recommended is coupled with metadata structured as tags. Microblogging hash tags, tag clouds over some document corpus and other domain-specific tags (like geo tags) could be used to hook up the content to the keyword-based profile of a user. Such kinds of approaches are also adopted for other purposes, like query expansion [3] or for Web mashups to filter some news feeds, using for instance the tag cloud generated from the Twitter timeline [16].

Semantic Network based - A profile is represented by a semantic network, basically a graph, where the nodes represent keywords extracted from a document corpus and the weighted arcs represent their co-occurrences [14]. The strength of such an approach relies in the implicit relationships between words that the semantic network brings to light. An interesting way to build and maintain such profiles is found out by [10]: the author shares our vision that a user profile can vary according to the short-term and long-term interests of the user.

Concept based - Concept based profiles are not so different from the Semantic networks based ones. The main difference consists in the fact that a concept profile is to be intended as a graph where nodes are concepts and arcs are conceptual relationships between concepts. In order to enable the system to do generalizations over such profiles, some authors propose to adopt a hierarchy of concepts. Such systems could be divided into those that adopt a fixed hierarchy and those that adopt a dynamic hierarchy, updated according to user feedback. This class of representation deserves a particular attention due to their similarity to our ‘Layer Cake’ model described in the first version of this document. One of the most representative systems, the OBIWAN [18], uses the Open Directory Project\(^3\) as its internal concept hierarchy: from this point of view, our idea of using the Linked Data cloud represents an extension that makes our model more ontology-based than their system. Basically, every proposed technique aiming to build such kind of profiles is based on algorithms that rely on a base taxonomy: thus in this case the main challenge is how to link the raw user data to the existing taxonomy terms.

3.1.2.1 The Semantic Web approach to the user modeling

We believe that the approach described in this document is in some sense revolutionary in that it uses both Semantic Web technologies and Linked Data ontologies to build a user model that could solve some issues raised by the previous methods.

As said at the beginning of this chapter, some user modeling systems leverage Semantic Web technologies in order to perform some personalization activities. An interesting work done in an adaptive web-based system context [5] aims to represent the different dimensions of the user (her features, actions, context, and so on) as different planes of a multidimensional matrix and formalized by means of ontologies. This UMS represents the semantic dimensions of the matrix in OWL and uses SWRL\(^4\) as a rule

\(^3\)http://www.dmoz.org/

\(^4\)http://www.w3.org/Submission/SWRL/
language to infer a user profile. This approach is similar to the one adopted within NoTube, where the matrix is replaced by a layercake.
4. The Beancounter Application

The hypothesis behind the NoTube user profiling mechanism, as initially described in the project and then in the first version of this deliverable is based on the assumption that a coherent and suitable user profile can be built by extracting and representing user interests and behaviors from various activities she performs on the Web, primarily in social web applications.

According to a classical and widely accepted architecture, as described in the State of the Art chapter (see below), every user profiling system foresees a set of software components allowing the collection of user data, the user profiling by means of such collected raw data (keeping them coherent and updated) and a component responsible for the syndication of such profiles.

The NoTube specific scenarios and the novel approach to profiling taken result in constraints that heavily impact the software requirements, motivating the main architectural choices behind the proposed solution. Such constraints come out from the following considerations:

Heterogeneity of Data Sources - the main aspect of the user profiling mechanism adopted in NoTube is about the way raw data about users is collected. Collecting user data from different services all around the Social Web raises an important issue regarding the heterogeneity of the various data sources. Even if the so-called Web 2.0 approach has resulted in the availability of Web APIs to allow the access to user data by third-party applications, the different ways such APIs are exposed and delivered give rise to a serious heterogeneity problem. First of all, such systems often use different authentication mechanisms (from basic http authentication to more refined technologies like OAuth\(^1\)). Secondly, different data formats to represent user data are exploited (XML, JSON and the still not widely adopted RDF), even though with regard to semantics such information could be considered equivalent. This scenario related to the Social Web is often referred to, in literature, as the Walled Gardens problem\(^2\). Thirdly, the specificity of every Social Web application obviously impacts on the data model. For instance, a general purpose social network like Facebook and a microblogging service have two completely different data models. Therefore, aside from the different syndications and representations of the user data, which could be often reduced to an interoperability problem, from a user-profiling point of view is important to remark that a user profiling system as the one designed in NoTube must put into action a set of mechanisms to identify and extract information that could be useful to infer user interests and behaviors. Following a design principle that aspires to embrace such heterogeneity rather than fight it, the NoTube Beancounter has been devised to allow a complete customization in terms of information extraction capabilities.

Representation of User Data - the heterogeneity of the various data sources addressed above clearly demands the adoption of a uniform internal representation.

---

1[^oauth]: [http://oauth.net/](http://oauth.net/)
As described in the first version of this document, RDF has been identified as the best choice. RDF, as a Semantic Web technology stack pillar, has been chosen as a flexible, powerful and extremely versatile data model to represent in an integrated and uniform manner data pulled from different data sources. Moreover, keeping track of the Web resources that a user manages on various social networks by their URLs\(^3\) nicely matches the emerging Linked Data\(^4\) paradigm, where everything could be identified by an URI. In this sense, our internal representation model could be easily linked to various Linked Data Clouds, enriching such resources with incredible flavors of interesting data.

**Performance** - ingesting users activities, representing them in terms of RDF triples, linking them to a set or Linked Data URI to enrich them (enabling for instance their identity resolution, technically achieved in NoTube by a service built in the WP4\(^5\)\) are a set of activities exposed to the same threats as any ordinary ingestion pipeline, and thus ask for scalability, bottleneck avoidance and robustness. Aside from the URI linkage, the envisioned pipelining system is also useful to perform some trivial real-time RDF entailments. Without leveraging any DL\(^6\) reasoner, a simple pipe has been provided, that produces new triples according to the schema of a certain vocabulary. For instance, for every RDF statement with a symmetric property as predicate, a new triple representing the inverse relationship is added.

**Modularity** - it is essential to leverage different profiling algorithms to process the gathered raw user data. Once the data of a specific user has been stored and represented with RDF, they must be accessible by different pieces of software, for different purposes. Thus the implementation and the deployment of different components which access the stored data in a transparent manner must be made as easy and natural as possible. Ranging from statistics collection to more complex RDF entailment, the envisioned system must foresee a set of APIs to perform batch data processes. Moreover, the batch nature of such processes naturally leads to a parallel and clustered execution implementation, easily achievable following a Map Reduce\(^7\) approach.

These considerations led the development of the NoTube Beancounter (described in detail below), a general purpose plugin based system, and a set of plugins performing the extraction of user raw data, the representation with RDF according to a set of specific vocabularies (as detailed below), the linking to some Linked Data clouds in order to enrich the resources (using work from WP4) and the execution on them of a group of profiling algorithms.

The whole system has been implemented with Java, allowing the hot-deployment of plugins, but it also exposes a set of REST APIs to control it and a SPARQL endpoint to access the stored data.

---

\(^3\)Typically users directly share URLs of resources of interest or do other things that may reflect an interest on a topic covered by some Web pages.

\(^4\)http://linkeddata.org/

\(^5\)http://lupedia.ontotext.com/ - see the latest version of WP4 deliverable, on progress while writing

\(^6\)Description Logic

\(^7\)urlhttp://labs.google.com/papers/mapreduce.html
5. Implementation

With the following chapter we will describe the ideas, the architecture, and the data flows in the implemented user profiling system.

5.1 Architecture

The Beancounter is a system built to gather information about users from social networks they are registered in, to aggregate and enrich such data in various ways and finally to use them to infer new knowledge about the users that can be reused. The Beancounter has to be placed in a wider picture (similar to the figure 3.1 above) as the one-way manager of the flow from user Web activities to user profiles. These computed user profiles can be used for the generation of content recommendations to be delivered to the owners of those profiles, and this flow is symbolized in the figure 5.1 by the big red right-to-left arrow. Yellow and green flows will be further detailed in the following “Data flows” section.

![Image of the Beancounter system]

Figure 5.1: The highest level picture: the user has a profile which can be used to filter or recommend content to the user

Each user has an internal Beancounter ‘RDF representation’, made of all relevant triples, that describe her in all interesting aspects for media recommendation purposes. In figure 5.1, the physical user is depicted on the left, with all her activities (watching a movie, listening to a song, seeing a picture, being in a place) and interactions with other users on various social networks. This user has a representation on the right, with the information passing through the Beancounter ‘filter’ that makes the profile semantic (we say more about what this means below). This representation is made of all user activities and interactions and enriched by other relevant information gathered autonomously by the Beancounter throughout the Web. In addition to this, the Beancounter is able to reason itself to infer new triples about the user. All this huge
mass of data will then be shaped in order to constitute the user profile, leveraged by an external recommender system (still to be investigated) to perform the proper media content recommendations to the specific user.

The Beancounter as a complex system can be described by figure 5.2.

All modules inside the grey box are accessible through a RESTful API.

5.1.1 Gathering raw data

The module that gathers web user activities is the Tubelet Container. A tubelet is a component responsible for retrieving and interpreting data from a specific social network, calling the relative web service (typically a REST API). Each tubelet is deployed inside the Tubelet Container and from that moment is considered as active. An execution of a tubelet is therefore the call to an external API to access information relative to a single user inside a specific social network. For instance here follows a slice of a response of the Friendfeed service for the user “Michele Minno” (expressed in JSON):

```
{"entries": [
    {"updated":"2009-07-14T10:26:09Z",
     "service":{
         "profileUrl":"http://www.youtube.com/profile?user=micheleminno&view=favorites",
         "iconUrl":"http://friendfeed.com/static/images/icons/youtube.png?v=c6c140ea173a7cfe98e5128620164d31",
         "id":"youtube",
     },
```

1http://friendfeed.com/
"entryType":"favorite",
"name":"YouTube"
},
"title":"The prestige trailer",
"media": [
{"enclosures":null,
"title":"The prestige trailer",
"content": [
{
"url":"http://www.youtube.com/v/MgNVqUMV6ramM&app=youtube_gdata",
"type":"application/x-shockwave-flash"
},
{
"url":"rtsp://rtsp2.youtube.com/CnQLEny73DA==/0/0/0/video.3gp",
"type":"video/3gpp"
}
],
"player":"http://www.youtube.com/watch?v=MgNVC6Hv4KE",
"link":"http://www.youtube.com/watch?v=MgNVC6Hv4KE",
"thumbnails": [
{
"url":"http://i.ytimg.com/vi/MgNVC6Hv4KE/default.jpg",
"width":120,
"height":90
},
{
"url":"http://i.ytimg.com/vi/MgNVC6Hv4KE/2.jpg",
"width":120,
"height":90
}
]
},
"comments": [],
"link":"http://www.youtube.com/watch?v=MgNVC6Hv4KE",
"likes": [],
"anonymous":false,
"published":"2009-07-14T10:26:09Z",
"hidden":false,
"id":"04bbe121-0b85-cab9-ca8d-11388e28f506",
"user":
{
"profileUrl":"http://friendfeed.com/michelemirino",
"nickname":"michelemirino",
"id":"fcddd8a9-0f7c-4bbf-acde-50a47d44539f",
"name":"michele minno"
}
],
[
}
The activation of a tubelet is programmed according to a scheduling rule, specifically a CRON expression\(^2\). There is a specific module, the Scheduler, responsible for handling the execution of each schedulable component, namely tubelets and reasonlets (further described). Whenever the Scheduler triggers the activation of the tubelet, the tubelet container must know for which users it has to call the relative web service, and with which credentials. This information is managed by another module, named (obviously) User Manager. The User Manager handles the identity of the users. It maps the different user identities on different social networks into a unique BeanCounter internal user ID. Once it has got the information about users from the User Manager, the activated tubelet loops within all the registered users and for each user retrieves the XML or JSON document that describes the user activities on that specific social network. It will not get each time all the activities performed by that user on that social network, just any increment from last activation of the tubelet for such user. The Delta Manager module provides the tubelet container with the last fetched activity for each user, in order to compute the delta of data that represents the new information to be gathered. These retrieved lines of code are then parsed and some relative objects are built, according to a model shaped following the service response structure. These objects are now the programmatic representation of the information gathered.

Before being flushed into the system, a mapping phase is performed. This is a crucial point: in order to be aggregated with information coming from other social networks, each one having an own response with a different syntax, service-specific objects must be mapped into cross-service objects. These objects aim to model a generic user activity on the web and thus are independent from any specific activity the user can have on a particular social network. To achieve this objective, a set of well-known Linked Data ontologies has been used. A brand new ontology has also been devised, named “Atom Activity Streams RDF mapping” (AAIR), with the clear intent to map into RDF the activitystrea.ms\(^3\) specification. Besides objects, a possible alternative output for the tubelet is a quantity of RDF triples with the same informational content.

5.1.2 Enriching data

After being produced by a tubelet, user information (in the form of objects or triples) are flushed down along the chain. The intermediate step before reaching the storage is represented by the Pipeline Container. Like the tubelet container, the pipeline container hosts components (pipelines in this case) that can be hot-deployed (deployed without restarting the application.

A pipeline is essentially a tube, made up of different pipes boxed up together. Each pipe can perform a single enrichment action on the input that passes through

---

\(^2\)A CRON expression is a string comprising 6 or 7 fields separated by white space that represents a set of times, normally as a schedule to execute some routine. It is used primarily by the UNIX tool "cron". \(\text{http://en.wikipedia.org/wiki/CRON_expression}\)

\(^3\)\text{http://activitystrea.ms/spec/1.0/}
that pipe. It is generally made up by a consumer part and a producer part (see figure 5.3). The consumer part accepts what comes from above, while the producer part generates what goes out below. Thus the consumer can be an object or a triple consumer, as well as the producer, depending on which kind of data they handle. An example of enrichment can be on geographical information about the user, for instance coming from a Brightkite tubelet. For each user activity, such as “User X visited place Y”, the enriching pipe would add a number of triples describing Y, calling for that a specific external web service. These additional triples would then flush exactly like the other ones into the next pipe, if present, or directly into the triple storage. The binding between tubelet and pipeline is defined inside the tubelet itself, the same pipeline being able to be bound to different tubelets. The pipeline container comes with two default pipelines, one that consumes triples and the other that consumes objects (coming from a tubelet); both obviously produce triples, to be fed into the triple storage.

An open issue tied to the pipelining process, and to the enriching of triples in particular, is the absence of any contextual information around the triple to be enriched. The enricher actually simply adds some more triples for every input triple of a certain type. It cannot be fired just in the case that the input triple has other specific triples connected to it (that is, its context). For this reason, it could be wiser to consider as input of the enricher not just the single triple, but the whole subgraph constituted by the sum of the triple and its context.

5.1.3 Reasoning about data

Up to now we have analyzed how raw data from social networks (and generic web services in general) can be manipulated and stored into the triple storage as RDF triples. All this tubelet plus pipeline data flow is purely source-dependent, being bound to a particular web source response. Also triples added in the enriching phase are generated synchronously with the event of tubelet activation but produce only its resulting local triples. To make more complex and general triples generation, another specific component (and relative container) has been devised: the Reasonlet. This name has been chosen in analogy with the tubelet, because a reasonlet can be seen as a particular type of tubelet. The only big difference is that a reasonlet takes as input the whole content of the triple storage, and not new raw data from a specific social network. Thus the input of each reasonlet is basically a snapshot of the ‘knowledge’ stored in the Beancounter, taken an instant before its activation. In this case we can talk of ‘reasoning’, because we have as input the entire set of triples and so a lot of complex algorithms on the RDF graph can be put in action. A reasonlet is activated by the scheduler, just as tubelets are. When its reasoning is over, the additional generated triples are fed into a pipeline and then into the storage, as we saw before. The introduction of the reasonlets became necessary to expand the knowledge of the Beancounter in a cross-source manner, in order to make some aggregations / consolidations / quality enhancements on semantic data contained inside, or simply some general enrichments, not tied to a single data source.

\footnote{The only constraint is that the last producer must be a triples producer, because it goes directly into the triple storage.}

\footnote{http://brightkite.com/}
5.1.4 Modeling / exposing data

The Beancounter must also be able to accept data in a synchronous way, by exposing specific REST services API. A first attempt to develop such functionality is the store_activity() REST service. By means of store_activity, a single new user activity can be pushed into the system. It must therefore be a self-contained atomic activity performed by a specific user, i.e.: ‘User X watched movie Y’. The model used for the service is the the AAIR ontology (see the end of ‘Gathering raw data’ section above). This information, packed as triples or as objects, can be afterwards flushed into a pipeline as usual. This is a service that writes onto the triple storage.

Another implemented service that reads from it is the OpenSocial SPARQL API interface (see below). These two services just outlined are only two implemented instances of a more generic component, still to be developed, named Modelet. A modelet (that lives inside its own container, the Modelet Container) wraps all is needed by the Beancounter to expose a new web service. It could be RESTful or not, in a write or read mode, and could follow any desired model or specification. Both in the read and in the write case, data could pass through a pipeline as it happens for the tubelets and the reasonlets.

5.2 The Open Social API

As we have sketched before, the modelet container will enable any developer to write their own web service to read and/or write data shaped in any desired structure (∼ ontology). In addition, the Beancounter comes with some APIs based on what appears to become the main standard for applications to be interoperable with social networks: OpenSocial\(^6\). OpenSocial defines a set of APIs about user social interactions and activities, along with general purpose and persistence APIs. The Beancounter exposes a subset of these APIs, namely that relative to the user interests. Basically what happens is that specific SPARQL queries are internally run on the triple storage endpoint and results are mapped into OpenSocial API results. The system uses Apache Shindig\(^7\), the reference implementation of the OpenSocial API specification. Using this endpoint a specific user profile could be obtained in a format totally compliant to the OpenSocial specification.

5.3 Data flows

This section is mainly intended to be a technical description and shows in detail how the data flows are transformed by the various deployed NoTube Beancounter plugins to achieve Linked-Data-based user profile. Taking a set of real-world data, the whole ingestion and enrichment process is described showing how the data are transformed in order to produce a user profile, syndicated according the OpenSocial specification.

The described scenario foresees a generic user who holds two accounts on two different social web applications: Last.fm\(^8\) and Glue.com\(^9\). Last.fm simply allows the

---

\(^6\)http://www.opensocial.org/
\(^7\)http://incubator.apache.org/shindig/
\(^8\)http://www.last.fm
\(^9\)http://getglue.com
user to listen to her favorites bands and artists, sharing songs with her friends and keeping track of live events of the bands. Glue.com acts like a real user log, technically achieved by means of a browser plugin. Conceptually similar to those user profiling systems that make use of external applications to track the users behaviors, Glue.com makes available through a set of Web APIs an exhaustive set of Web resources the user visited, shared or liked, enriching them with an internal categorization. The next sections show how the NoTube Beancounter aggregates data from these sources, links the information to several Linked Data clouds and reasons using the data in order to make explicit the user interests and behaviors. The information aggregation is achieved through the identity resolution made against different ontologies. It will be described underlining its importance for the final user profiling objective.

Summarizing, the following example shows how it is possible to infer a set of interests, uniquely identified by some Linked Data identifiers, starting from the tracks and the Web resources accessed by a user across two different social networks.

### 5.3.1 From different social networks to aggregated activities in RDF

Once a running instance of the Beancounter has been set up and once the user has provided the NoTube Beancounter with the required credentials to let it access the Last.fm and Glue.com Web APIs, two deployed tubelets periodically start to call two specific services obtaining the responses illustrated in 5.4 and 5.5:

The Last.fm response is basically a list of tracks the user recently listened to, providing for each of them the artist and the album UUIDs\(^{10}\), a name and other data that is out of the scope here.

The Glue.com response is a list of interactions that the user had with several Web resources, typically html pages or anything else could be uniquely identified with an URL. Such interactions are described in term of the HTTP link, the title of the resource, the action the user performed on it (liked, shared, commented, bookmarked), various other information and, most important, the category of the resource according to Glue.com’s internal hierarchy of concepts.

The first important thing to remark here is the double heterogeneity that immediately occurs:

- **Representation Heterogeneity**: two different XML schemas are employed. This issue is straightforwardly addressed by developing two different tubelets with embedded the suitable code to parse these two different serializations. Moreover, as detailed below, the two tubelets natively contain the code to represent such data with RDF, according the Atom Activity Streams RDF Vocabulary.

- **Data Model Mismatch**: aside from differences in syntax, the main issue is related to different kinds of implicit information these two services are bringing within their own data models. The main challenge here is trying to leverage such flat descriptions to integrate them on a semantic level. If we assume that the tracks showed by Last.fm are from artists in which the user has some interest and such interests also reflect on the way she browses on the Web, then it is possible to leverage such implicit information, making them part of her profile.

\(^{10}\)http://en.wikipedia.org/wiki/UUID
So, as a first step, the NoTube Beancounter translates such responses in the following RDF statements, listed in figures 5.6, 5.7, 5.8 and 5.9 for the sake of clarity both as triples (subject, predicate, object) and as nodes and arcs.

As the example shows, the two tubelets concretely realize a first data integration representing the two responses in a uniform way according the Atom Activity In RDF Vocabulary.

The next step is to perform the identity resolution on the extracted resources, wherever it is possible.

5.3.2 Identity resolution as a pipe inside a pipeline

To achieve this task, the NoTube Beancounter is equipped with a pipeline able to trigger an external service (more specifically, LUpedia\(^{11}\) provided as part of the WP4 work) that provides a set of Linked Data URIs that are basically identifiers of the concepts embedded or represented by resources passed to it as input. For example, with reference to the Figure 5.10.

Two different RDF graphs, derived from two different actions on two different social applications, are here being connected by

http://dbpedia.org/resource/Nick_Drake

Linking the two Web resources with owl:sameAs statement claims in a uniform way that they are referring to the same concept.

This is possible because the NoTube Beancounter, making use of its powerful pipelining mechanism, asks to LUpedia to identify such resources and produces new connections between them among the RDF graphs built by the tubelets.

5.3.3 Rule-based simple entailment

The identity resolution is just an example of all possible data enrichments that the NoTube Beancounter pipeline mechanism could offer. It is also used to perform a simple, but meaningful, entailment over the extracted RDF statements. The entailment is extremely important to automatically make explicit some implicit claims potentially contained in the extracted RDF statements. Such entailment is straightforwardly achieved by translating a Vocabulary RDF Schema into a set of rules, or just manually providing a set of them. Each rule firing, roughly speaking, consist in adding some defined statements when a characteristic of a certain property is matched.

Such mechanism, even for the cases where it is not as powerful as a OWL-DL reasoner, could be employed to perform a simple kind of data consolidation that sets the stage for the next steps while building the users profiles.

For instance, let us suppose that two different tubelets, after the identity resolution pipelining, produced these two statements:

http://com.asemantics.notube/instance#13affb  http://www.w3.org/2002/07/owl#sameAs
http://dbpedia.org/resource/The_Science_of_Sleep

http://com.asemantics.notube/instance#9bfdab  http://www.w3.org/2002/07/owl#sameAs
http://dbpedia.org/resource/The_Science_of_Sleep

\(^\text{11}\)http://lupedia.ontotext.com/
then simple rules based on the transitivity of the \textit{owl:sameAs} property triggers the generation of the following statement:

\begin{verbatim}
http://com.asemantics.notube/instance#9bfdaa http://www.w3.org/2002/07/owl#sameAs
http://com.asemantics.notube/instance#13affb
\end{verbatim}

5.3.4 Reasonlet activation to produce user profiles

The extraction and the pipelining-based data enrichment produces, for each user registered to the Beancounter, a huge, uniform and coherent RDF graph containing all the user activities. Within this graph several Web resources have links to concepts belonging to other less mutable Linked Data clouds. In this sense the Beancounter provides all the machinery needed to link the Web of Data with the so-called Real-Time Web. Linking these two different paradigms, it naturally leads to develop an innovative way to approach the user profiling. Since every RDF user graph contains uniquely identified resources, it is possible to gather from them all the necessary information to infer interests and behaviors of the users.

As the above discussions and examples show, in the RDF graph of our initial user we found several (and not only) DBpedia URIs like the following ones:

\begin{verbatim}
http://dbpedia.org/resource/Eternal_Sunshine_of_the_Spotless_Mind
http://dbpedia.org/resource/Neverending_Sunshine_of_the_Spotless_Mind
http://dbpedia.org/resource/Nick_Drake
\end{verbatim}

Thus the main challenge here was how to extract music genres or other kind of concepts that are somehow interesting for the user. Our first approach (to be still properly evaluated and assessed) is based on \textit{SKOS}\footnote{http://www.w3.org/2004/02/skos/}.

The main idea is to use the \textit{skos:subject} property values as identifiers of the user interests: if a user listens to a certain number of bands or musical artists sharing the same \textit{skos:subject}, then it could be reasonable to infer that such subject represents an interest for that user (making a reasoning similar to a logical induction). Moreover, the extremely rich and complex \textit{SKOS} hierarchy of DBpedia allows to extract a lot of other interesting information. For example, if a user is particularly interested in movies where a particular actor or actress played, more information will be available about this in the system, since it is highly probable that DBpedia contains some \textit{SKOS} subjects describing this.

Or, again, if a user listens to a certain number of different bands, all of them originally born in a specific geographical region (quite often the most popular music genres are related to a specific region), then this could be represented with a \textit{skos:subject} property. Assuming that a user is particularly interested in bands from a certain region, this could be a statement useful to perform recommendations of other bands and artists.

The developed algorithm has two different aspects:

1. The linkage of the resources across the Web of Data. More specifically, it is simply achieved fetching the various \textit{SKOS} subjects of the resources to be categorized,
2. Then such set of skos:subject is collected and simply counted (and here comes the name Beancounter), revealing the statistical side of the algorithm. Only such subjects that have been counted more than a certain threshold value will be considered as meaningful for a suitable user profile.

Such values will represent the weights of the interests according to the Weighted Interests representation detailed below.

Finally, the modularity and completely pluggable architecture of the Beancounter allows us to implement such algorithm as a Reasonlet.

This is just a first quite trivial implemented attempt to model and compute user profiles. Different algorithms or mechanism could be similarly implemented and deployed, the various intermediate results being integrated together or not in the final user profiles.

5.3.5 Accessing user interests by OpenSocial

Finally, an OpenSocial compliant endpoint is made available, to allow third-party applications to retrieve user interests (and profile in general). The following SPARQL query is internally executed on the triple storage and mapped to the OpenSocial data model in order to retrieve the JSON code with the desired response.

```
SELECT ?uri FROM <user_graph>
WHERE {<http://notube.tv/profile/user-graph>
  <http://notube.tv/profiles/hasInterest> ?interest.
  ?interest <http://www.w3.org/2002/07/owl#sameAs> ?uri }
```

5.4 Interfaces for administrators and users

There are two main interfaces for humans: an interface for administrators that allows them to manage all aspects of the system, and an interface addressed to the final Web user, the owner of the activities gathered and used by the system.

5.4.1 Administrators

The administrator user interface enables simple operations relating internal components and Web users. It enables the administrator to add or remove any component belonging to any internal container (tubelet, pipeline, reasonlet, modelet). The administrator has also the possibility to force the execution of such components, not waiting for the ordinary scheduling rule to take effect. New Web users can be added onto the system, and associated to a specific tubelet (inserting the user’s own credentials on the relating social network).

5.4.2 Users

Web end users have their own user interface to see what the Beancounter did for them. In particular, they can inspect their profile computed from their activities, edit it and
decide not to show some profile branches. In this interface the user will also be able to see the recommendations about media content, which are the final result of the entire system and which will be delivered to the proper user media devices in order to be enjoyed by the user (the recommendations will be available via an API, and this is one area where the user will be able to view them - other places might be on their media centre or set top box, or in another video viewing system on the Web).

\[13\text{All aspects related to user privacy will be addressed as the next step of the development.}\]
Figure 5.3: Pipelines - default and combinations
Figure 5.4: Last.fm response

Figure 5.5: Glue response
Figure 5.9: Glue graph

Figure 5.10: Integrated graph
Figure 5.11: OpenSocial results
6. User and Context models

As described in the previous version of this document, we use layered model for a user profile: activities, preferences, and interests. We summarise here a part of section 5 of the first version of this deliverable as a reminder of this approach.

6.1 The Layered Common Knowledge and User Model

We need to be able to effectively model the user and to embed her activity, her tastes, her habits, her consumption contexts in a common knowledge background. Due to the high complexity of the different knowledge domains involved, the best model architecture results in being a sort of ‘layer cake’. Each layer will represent a different knowledge domain, i.e. temporal, spatial, geographic, music-specific, movie-specific, etc.

6.1.1 The Layer Cake

The layer cake is a 3D geometrical figure, depicted below, here intended to represent the stratification of knowledge that a human recommender would implicitly or explicitly leverage to effectively perform a good recommendation to someone else. Each layer hosts a graph, with nodes standing for concepts and arches standing for semantic connections between them. Instead of representing this knowledge as a whole flat huge graph, a third z dimension has been introduced, in order to achieve a differentiation of domains. Nodes with the same z coordinate will share the same domain, i.e. will be the instrument and player concepts inside the same music layer. Since layers are strongly coupled, due to many semantic connections among nodes belonging to different layers, vertical/oblique links are also possible. While horizontal links represent semantic connections between two nodes pertaining to the same domain, a vertical link connects two nodes of different domains that have some kind of relation between them. For example, a music player can performs in a certain town, modeled in the geographical layer.

Figure 6.1: A two-layer layer cake
6.1.1.1 The hardly changing part: World, things

There is some further difference among layers. Some of them are intended to be fixed over time and users. A geographical layer will remain almost the same over time, unless little changes, shared among all the users. These almost fixed layers represent the background static part on which the dynamic user profile and context evolves. In the same way, connections between these generic layers will hardly change: a link between a town in the geographical layer and a year in the temporal layer indicating its foundation will remain always the same.

6.1.1.2 The slowly changing part: social graph

A semi-dynamic layer is that representing social relationships among people, the so-called social graph. This is considered as a slowly changing layer, since people change slowly the nature of their interactions with others. For example, a person can add a new friend in her social graph, or can change her relationship from friend to lover, but this happens quite rarely in the passing of time. Surely it will happen more frequently that a change in some layer of the first kind, like the geographical one. An example of link between these mentioned two layers could be a link between a town and a person born in that town.

6.1.1.3 The quickly changing part: user activity, context and profile

Until now we have seen the part of the layer cake that is shared by the users, the world description, and the social layer that could be partially shared among some people. What is specific to each user is her activity, the context she is in and her profile. All this information comes from the traceable actions user perform on the web.

- **Activity**: the straightforward way to trace user web activity is by activity streams. An activity stream builds a life-stream for the user by aggregating her social activities all in one place. Then this life stream should be gathered and formalized in a layer of the layer cake. Another more subtle and effective way to achieve a description of the very actual real-time activity of the user is to listen to her relevant actions, like open a music web player or look at some pictures. All this will result in useful hints to make the right content recommendation to the user.

- **Context**: another crucial aspect of the user on the web is the modality she interacts with it. So another layer will be specifically dedicated to context. Context in all its possible forms and flavors. The first important context information is about spatial/geographical and temporal context. If the user is in Rome or in Norway, it could make a big difference in terms of content and/or language she wants to be open to. So it does for temporal ranges: the morning could lead to different preferred music genres than the evening, for example. A spatial information more linked to user functional aspects is ‘home/work’ or something similar, cause here the space tells more than just a location.

Another orthogonal dimension is that related to the device the user has got to experience the web. This is very important, since the very content may change de-
pending on which kind of media quality, dimensions and functional requirements the user device supports.

- **Profile:** the last layer is the most important one, since it aims to encode all possible statements about user in general. Therefore it turns out to be the formal representation of the user profile. Each statement that is going to populate this layer will come from a reasoner that will deduce it from the other layers’ information and from this layer too, in a loop fashion that will be further described in details. So these statements do not directly derive from user, but filtered by a reasoning engine that will compute every fact entailed by the whole layer cake and will make the profile ‘living’.

The user profile and context are described in **User Profile Model and Schemas** below. The underlying activities vocabulary is sketched in the section below that. Work on additional RDF vocabularies used in various places in the project occurs in WP1; the vocabularies below will also be placed at SPARQL endpoints within that workpackage.

### 6.2 User Profile Model and Schemas

Within the project we have designed a vocabulary for describing user interests in some detail. The aim of this experimental vocabulary is to provide filtering and recommendation services with a summary of a user’s preferences when they are in different environments (contexts). It can be combined or used instead of straightforward FOAF interest profiles, and be combined with or used instead of information traditionally used to make recommendations to users, in particular age, gender, and location.

The model is inspired by the economic notion of preferences, specifically an ordering over bundles of goods and services, for example ‘The genre sport experienced while I am at work in the afternoon’. In economic theory the rational agent is able to compare any bundle of services in this way and choose between them. This model is controversial and may not be generally applicable, but nevertheless intuitively in some situations the user can make such a choice; as we gather more information about the user we will be able to apply algorithms to represent this choice; and such an ordering may be useful to certain types of recommendations systems.

Nevertheless there may be cases where the user cannot or does not want to express a preference, and here we simply represent the user as being interested in something. Likewise, a user may be specifically not interested in something, and we represent that too. Finally, guided by requirements, we must be able to represent age, sex and location, the traditional advertising user profile components; and we must also be able to represent programmes, brands, series, contributors, genres and potentially other items of interest.

RDF, and specifically the well-used FOAF vocabulary as described in RDF, gives us age, sex, gender and basic interests expressed as URIs, so that fulfills many of the requirements. Here is an example of a FOAF file with concepts pointing to interests, an the addition of some basic typing of the URIs to help a recommender application:

```xml
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix progs: <http://purl.org/ontology/po/> .
```
Here’s an example of the weighted interests vocabulary, which extends FOAF to allow people to describe their preferences as orderings within a certain context.

In words:

I prefer radio 4 over radio 5 when I am working at home (which is every weekday between 8am and 7pm).

and the same example using the vocabulary:
Another example in words:

I hate the X-Factor

In the vocabulary:

```n3
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix wi: <http://xmlns.com/wi#> .
<http://swordfish.rdfweb.org/people/libby/rdfweb/webwho.xrdf#me> a foaf:Person;
foaf:name "Libby Miller";
```

The model is illustrated in figure 6.2. The RDF schema is available in the Appendix and online.

### 6.3 Activities Model

The activities model is a representation of Atom Activities In RDF (AAIR). The general model is that an activity has an actor, an object and a verb type ('consumed', 'bookmarked' and so on) that represents the interaction that the actor has with the object. We have kept the RDF translation as close to the Atom work as possible and we will continue to follow it. This is the internal representation of users' activities, and this RDF version therefore enables us to:

- Connect different vocabularies and items to the entities described using URIs
- Use RDF tools to store and query the data

Figure 6.3 shows the vocabulary diagramatically.

An example showing mixing with other vocabularies is below as N3:

```n3
@prefix aair: <http://xmlns.notu.be/aair/> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix terms: <http://purl.org/dc/terms/> .
<http://com.asemantics.notube/instance#e3df54> rdf:type aair:Activity .
<http://com.asemantics.notube/instance#cfda09> rdf:type aair:Verb ;
aair:verbType "Looked" .
<http://com.asemantics.notube/instance#e3b250> rdf:type ns:Object ;
owl:sameAs <http://getglue.com/recording_artists/nick_drake?source=search> ;
aair:type "recording_artists/nick_drake" ;
terms:description "none" ;
dc:title "Nick Drake" .
```

1http://xmlns.notu.be/wiki#
Figure 6.2: A diagram of the weighted interests vocabulary

Figure 6.3: A diagram of the activities vocabulary
The RDF schema is in the Appendix and online².

²http://xmlns.notu.be/aair#
7. Privacy and Data Security

NoTube’s mission is to ‘put the user back in the driving seat’, through making their needs and interests central to a modernised experience of television. This is attempted by building systems based around rich and expressive representation of user interests, but also by adopting an architecture that embraces the entire Web as an environment in which users can explore and share content. In particular, NoTube expects ‘television’ to melt into the wider Web, rather than remain a separate medium. Many of the characteristics of the Web experience such as linking, annotation, bookmarking and universal access will eventually merge with television. Given these background assumptions, it is critical that NoTube specifies mechanisms that keep users in control of their data, especially when they are being linked, shared and integrated between diverse sites and services. NoTube’s approach to user profiles is to ground them in users’ online behaviour, and with their permission to derive higher level summaries of user interests from analysis of activity streams and other data collected from around the Web.

7.1 Privacy

Several aspects of the Beancounter are of particular significance with respect to privacy. These are:

- The aggregation and analysis of users’ previously disparate and unconnected data
- The ‘enhancement’ of users’ data by linking it to other data, thereby creating relationships where none may have existed before
- The collective aggregation of users’ data for statistical analysis

While we believe there will be significant benefits to the user of collecting their data in this way (by delivering more personalised recommendations, a portable interests file, and previously unavailable functionality such as ‘show me which TV series I watched the most last year’), the Beancounter project faces several challenges in finding ways of effectively communicating the associated privacy risks to users without scaring them and in enabling users to have full control over their data whilst still being willing to share it appropriately.

This is a challenge because:

- People find it very difficult to think about privacy in an abstract way: for a start it’s hard to define what we mean by ‘privacy’ (ID theft? credit card fraud?). In this context we are talking about intimate data: information users might prefer not to share.
- Perceptions of privacy vary across nations and cultures. For example, people in India are much more comfortable about giving out personal details on social networks than people in America\(^1\).

\(^1\)from SYNOVATE 2008: Social Network Users
Managing privacy in everyday life is an intuitive process: the front you present is not only tailored to the pertinent audience but also to the context (e.g. at work/at home) and it determines the amount of information you are willing to disclose to the audience. It is subtle and complex, but it comes automatically and you do not have to think about it very much.

People systematically underestimate privacy risks online because the urge to sociality is such a highly motivating psychological force.

The reaction of users to hypothetical or artificial situations is not a good predictor of actual behaviour and feelings experienced in real life.

Reassuring people about online privacy tends to make them more, not less, concerned. The results of a series of experiments conducted by Carnegie Mellon University showed that people who were reminded about privacy were less likely to reveal personal information than those who were not. As The Guardian said: “Users care about privacy, but don’t really think about it day to day. The social networking sites don’t want to remind users about privacy, even if they talk about it positively, because any reminder will result in users remembering their privacy fears and becoming more cautious about sharing personal data.”

It becomes even more complicated if anyone can aggregate anyone else’s data.

7.1.1 Privacy Issues Around Data Aggregation

The attention data is already out there, so does aggregating it actually make a difference? Yes, because data aggregation turns individual pieces of information into a more complete picture which is greater than the sum of its parts and from which interesting patterns of behaviour can be derived (as with a Tesco clubcard).

A set of aggregated data for an individual therefore has a greater value attached to it than the discrete items of data within it. There is value not only to the individual (data aggregation has been used to improve recommendation systems of websites like Netflix and Amazon), but also to commercial organisations who can make use of such data to deliver more highly targeted advertising for example (attention data is a commodity to be bought, sold and traded in “the attention economy). Ownership of this valuable data is therefore a key issue.

Beancounter is based on the principle that the user owns their own data, can securely move their data wherever and whenever they want to and can dictate how it can be used. However, this is quite a new concept for people: within most media applications, individuals do not have access to their activity data and can not use or share it how they choose.

It is important to be able to safeguard this Beancounter data, and to be transparent and accountable over how it is used. It is also important to have clear policies about deletion of data: Dopplr is a good example of this - if a user deletes their Dopplr account all their data is wiped from the Dopplr service and emailed to the user for

---

3http://www.heinz.cmu.edu/~acquisti/papers/acquisti_grossklags_eisRefs.pdf
4http://www.guardian.co.uk/technology/2009/jul/15/privacy-internet-facebookcommented
5http://hellomatty.com/wod/s5.html
On deletion of a Flickr account, all the user’s photos are deleted from the system. Whilst you can delete your complete location information from Fire Eagle this will not delete location data collected over time by authorised sites by your Fire Eagle account. If users opt out of Fire Eagle, previously collected information can be kept by the developers offering the service through their applications. By contrast, a user who leaves Facebook can’t take their data with them.

Collective aggregation of Beancounter data is also potentially very valuable for statistical analysis by programme makers and other interested third parties - for example, to see how many people watched a particular episode of a TV programme and to determine what else these people watched. This use of an individual’s data would require ‘opt-in’, which leads to the challenge of finding ways to elicit real ‘informed consent’ and providing privacy policies that are clear and accessible to the user. Is ticking a checkbox enough to obtain genuinely informed consent? Geest et al. argue that providing a checkbox is inappropriate and insufficient in the context of users providing information for personal profiles in online environments.

7.1.2 Privacy Issues Around Linking and Data ‘Enhancement’

Besides issues of ownership and data portability, combining/linking attention data and then performing statistical correlation can reveal surprising things that people might not want to know about themselves, or not want others to know. Information accrual combined with semantic modelling makes it possible to discover information and relationships that were previously impossible and the unwitting disclosure of personal information - even if that data is publically available.

For example, a team of researchers developed a Semantic Web application for detecting conflicts of interest relationships among potential reviewers and authors of scientific papers. Whilst the focus of the project was building applications that leverage semantics, it had the unexpected outcome of discovering that many authors flout conflict of interest rules by using people they know to review their papers. The researchers used information publicly available about authors of scientific papers from two different sources and found that authors who had published joint papers together in the same research institution later acted as reviewers for each other’s papers when one of the authors had moved to a different institution.

In the past, finding out such information would have been very difficult, but with social networks combined with semantic web techniques it is much easier for a software program to detect such patterns.

MIT’s Gaydar project is another example of how information can be shared inadvertently. The project used seemingly irrelevant Web profile details (people’s lists of friends in Facebook) to make statistically reliable guesses about private matters - in this case, people’s sexuality.

“Using data from Facebook, two students in a MIT class on ethics and law on the electronic frontier made a striking discovery: just by looking at a person’s online friends, they could predict whether the person was gay”

http://www.dopplr.com/account/closeout
We do not yet know what Beancounter ‘enhancement’ and analysis might reveal about people. We have used the very obvious ‘porn’ example to date (see figure 7.1), but we should probably assume that there will be other potentially sensitive examples which users may wish to remain private, for example health issues or political opinions. Further, while users can choose which sources of attention data to add to the Beancounter, if those sources then link to other sources to which the user has not given permission to be ‘counted’ and the data from this source becomes available to Beancounter via this route (e.g. Facebook becomes integrated with LinkedIn), then the user’s apparent control is compromised.

**Bob’s Bean Counter Profile**

Bean Counter has analysed your watching and listening activity to generate a profile of your interests and friends:

**Your interests**
- cars
- cookery
- food
- jazz
- malaysia
- ornithology
- porn
- tennis

Figure 7.1: A picture of the Beancounter should NOT do in a public profile

### 7.1.3 Privacy Issues Around Sharing

While the success of Beancounter-driven recommendations may rely, in part, on users sharing their Beancounter profiles with others (so that recommendations can be made on the basis of friends’ tastes as well as the user’s interests), it is important to give users maximum control over how their data is shared and with whom so as to avoid any humiliation or embarrassments. However, this is easier said than done.

People may be quite guarded about sharing their TV viewing behaviours; they may worry about how much time they can be seen to be wasting by watching trashy TV (their ‘guilty pleasures’) which might not align with the public view of themselves they have presented online.

This may not be such an issue for Beancounter because users will be able to control exactly which sources of attention data are ‘counted’ (so presumably if it is their public Twitter stream they will only have tweeted about programmes which reinforce their online persona as a cultivated and witty person rather than a couch potato).
but we might still assume that many people would probably only want to share their Beancounter profile within a small network of trusted friends. We also have to allow users the ability to edit their profile before sharing it, and to start and stop sharing data at any time.

It is very difficult to provide granular control of disclosure of personal information without imposing a heavy cognitive burden on the user. Facebook now has very extensive and precise privacy settings, but general consensus seems to be that they are very confusing and difficult to use.

This goes back to the previous point about how difficult it is for us to abstract privacy issues because we deal with them so effortlessly in everyday life, which might explain why many users never customise their privacy settings at all but just stick with the defaults (which, for social networking sites, are going to promote the sharing of data rather than keeping it private - without sharing there is no social network). This can be problematic when discretion is required: for example, earlier this year, the appointment of the new head of MI6 was exposed by his wife’s Facebook page\footnote{http://www.guardian.co.uk/politics/2009/jul/05/mi6-facebook-sawers-wife-miliband}. She had left all her Facebook privacy settings on their default settings, so all the information was available to any of the 200 million users in the open-access London network, as well as being searchable on Google. It seems that for Beancounter, the most appropriate default setting is ‘keep everything private until I choose to share it’.

7.1.4 Supporting User Control

When it comes to deciding what to share and with whom, it may be easier for users to focus on the ‘who’ i.e. the individuals they want to share with as the starting point, rather than focusing on the ‘what’ i.e. the type of information to be shared (the latter seems to be the current user experience paradigm). A ‘who’ approach may align more closely with the way we think offline. For example, Lederer et al. [2003]\footnote{http://portal.acm.org/citation.cfm?id=765952} found that privacy preferences varied by ‘inquirer’ more than by situation - i.e. individuals were more likely to apply the same privacy preferences to the same person in different situations than to apply the same privacy preferences to different people in the same situation.

“For me, ‘who’ is all that matters. If I don’t trust the person with personal information, I wouldn’t want to give them information at any time. If I do trust the person, I’m willing to give out information freely”

However, there are exceptional circumstances where the situation may be more important: for example, you don’t want your partner to know your location because you’re organising a surprise for them. In the context of Beancounter, it’s also possible that you might not want your partner, for example, to see what you’ve been watching, which is why granular sharing of the ‘what’ also has to be supported.

In another interesting project, Lipford et al. (2008) found that presenting an audience-oriented view of profile information significantly improved the understanding of privacy settings on Facebook, which may be why there is now an option to ‘See how a friend sees your profile’ (you then type in the person’s name).
The Locaccino project, a location-based friend-finding service for Facebook based on research at Carnegie Mellon University, found that:

“Most of [the researcher’s] test subjects started out reluctant to share their every move, even with friends. But users generally warmed to the system after they found the ‘hide my location’ button for when they wanted to drop off the map”.

Likewise users of Fire Eagle can temporarily stop people from seeing their location using a ‘Hide me’ option.

Experimenting with the best ways to support granular control in the Beancounter UI is a major area of future investigation for NoTube. It would be good if we too, like Locaccino, could design a very simple feature which is easy to understand and use, and which gives users immediate control without requiring them to enter into a complex decision-making process.

7.1.5 Privacy: Conclusions

Several areas of further NoTube user experience research have been identified so far in this document, including presentation of privacy policies for informed consent and design of simple and effective tools for complex granular control of data sharing.

In addition, we will also need proper legal advice, although the law has yet to catch-up with ever-evolving needs for privacy.

In the meantime we need to focus on ways of balancing our desire for people to share their Beancounter data in order to influence the recommendations that their contacts receive (because it seems that people respond better to recommendations made by people they know), with our obligations to ensure that people actually understand what the potential costs to privacy might be.

7.2 Data Access Controls: OAuth and OpenSocial

The Beancounter accesses various data sources with the user’s permission and currently using the username and password of the user directly. This works because this pattern is common across all secure sites, but in general is not a good approach to keeping user data private, for the following reasons:

• It encourages users to give away their password to sites that could be ‘phishing’, i.e. misusing the user’s account for social engineering or other fraudulent purposes. Twitter users, for example, are frequently on the receiving end of such attacks.

• It lays open the opportunity for passwords to be stored in a way (even if hashed) that developers may accidentally make them available to attackers via attacks on their software, social hacking, or security breaches.
In the future we expect to resolve some of the difficulties with use and storage of passwords in Beancounter with OAuth, a technology that makes it possible for a Web site to request data held at a second site, through a Web-based interface in which the user gets to grant (or revoke) data access tokens.

For example, a NoTube-based site might want to negotiate access to information (book purchases, microblog messages, bookmarks, favourite videos) which are not themselves publically available. OAuth provides a framework for this. We expect to use OAuth 1.0a and monitor the ongoing IETF standards work around OAuth and the newly proposed (SSL-centric) OAuth-Wrap work. A second scenario is the use of OAuth for controlling access to data held by NoTube systems; for example, a recommender component might want to negotiate access to a summary of some users’ YouTube viewing history. Although that summary might have been produced by NoTube-inspired software and methods, this doesn’t mean that the data can be handed over freely to any other piece of software that wants it. So, even within the NoTube ecosystem, we need a permissioning framework that puts users firmly in control of data flow.

The OAuth setup process works best in a full (desktop) Web browser. It can be achieved on smart-phones too (eg. iPhone safari) but the user experience is suboptimal. However once a software component (eg. a recommender tool embedded in media-centre software such as MythTV, Boxee, MediaPortal or Freevo) has - probably via WWW - acquired an access token to restricted data, that access token can be used directly and invisibly for inter-component communication. This is typically but not necessarily achieved with HTTP; however an OAuth binding to XMPP has been defined\(^\text{12}\); we anticipate exploring this for remote controls. For example, a smartphone-based remote in possession of such an access token could refresh its list of user profile fields by consulting a database of profiles which was otherwise inaccessible.

OAuth is also a key component of the wider OpenSocial framework for extensible, interoperable Social Network systems. High level user profiles in NoTube are being accessible also via OpenSocial APIs, so there is likelihood we can exploit OAuth for mediated access to these too.

\(^{12}\)XEP-0235: OAuth Over XMPP - \texttt{http://xmpp.org/extensions/xep-0235.html}
8. Future Work

The NoTube Beancounter concretely represents the most important deliverable of the WP3 during the first project year. It has been developed with the main aim of providing a solid infrastructure for deploying and experimenting different user profiling algorithms and techniques. The SKOS, and more generally the Linked Data, approach is still in an embryonic stage and the presence of a modular architecture could play a key technology role.

Even if WP3 is, mainly, a research work package, a lot of effort has been employed to deliver a solid code base. In this sense the Beancounter, acting as an innovative and extensible user profiling component, plays a key role in the whole NoTube architecture\(^1\). Its services, exposed as a set of REST APIs, foster the integration with the use cases work packages. Even if those work packages are not all at the same level with regards to the integration, one of them, the 7a, already integrated its services fully delegating the user identity and management to a running and publicly available instance of the Beancounter.

Two papers covering two different aspects of the key research behind the Beancounter have been accepted and discussed. The first one\(^2\) covered the main ideas behind the Beancounter: data aggregation and consumption to achieve a personalized TV content fruition. The second one focused on a SKOS-based autocompletion system\(^3\) which contained the first prototypal ideas behind the usage of SKOS to describe and make use of such information.

The remaining time of the project dedicated to user profiling will be spent on consolidating the software. Substantial effort on integration with the other use case partners is expected, as well expanding the impact of the Beancounter on the Open Source community. The whole Beancounter code has been released with a MIT license\(^4\) and is attracting various third-party developers interested in maintaining the code. In addition, the user interfaces will be improved in order to make the utilization of such services more user friendly to boost the impact on the Web.

We hope that the Open Source community can help to maintain and improve the code since the remaining part of the project will be dedicated to develop and implementing a TV content recommendation engine running on the data collected by the Beancounter. Architectural aspects, algorithms, access policies and other technological issues will emerge and therefore addressed and described in the next expected deliverables.

\(^1\) see the last WP6 deliverable, in progress while writing.
\(^2\) http://www.opensource.org/licenses/mit-license.php

53 of 64
REFERENCES


[17] Davide Palmisano, Michele Minno, Dan Brickley, and Michele Mostarda. A semantic web based solution for an autocompletion system.


[22] Chris van Aart, Lora Aroyo, Yves Raimond, Dan Brickley, Guus Schreiber, Michele Minno, Libby Miller, Davide Palmisano, Michele Mostarda, Ronald Siebes, and Vicky Buser. The notube beancounter: Aggregating user data for television programme recommendation.


A. RDF Schemas

Note that the namespace URLs may yet change: we are investigating getting permanent purls for them to ensure they have a life after the end of the project.

The schemas are also available online in RDF and human-readable form: AAIR: http://xmlns.notu.be/aair# and Weighted Interests: http://xmlns.notu.be/wi#

A.1 Atom Activities in RDF

```xml
<?xml version="1.0"?>
<rdf:RDF xmlns:event="http://purl.org/NET/c4dn/event.owl#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:status="http://www.w3.org/2003/08/u-vocab-status/ns#"
  xmlns:xsl="http://www.w3.org/2002/07/xsl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:rdf="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:vann="http://purl.org/vocab/vann/"
  xmlns:owl="http://www.w3.org/1994/02/oil#"
  xmlns:skos="http://www.w3.org/2004/02/skos/core#"
  xmlns:aair="http://xmlns.notu.be/aair#">
  <owl:Ontology rdf:about="http://xmlns.notu.be/aair#">
    <vann:preferredNamespaceUri>http://xmlns.notu.be/aair#</vann:preferredNamespaceUri>
    <vann:preferredNamespacePrefix>aair</vann:preferredNamespacePrefix>
    <dc:title>Atom Activity Streams in RDF</dc:title>
    <rdfs:comment>draft vocabulary for mapping the Atom Activity streams work to RDF.</rdfs:comment>
    <foaf:maker rdf:resource="http://it.linkedin.com/in/micheleminno"/>
    <foaf:maker rdf:resource="http://identi.ca/user/30610"/>
  </owl:Ontology>

  <rdfs:Class rdf:about="#Activity">
    <status:term_status>unstable</status:term_status>
    <rdfs:label>Activity</rdfs:label>
    <rdfs:comment>A generic activity an agent performs on the web.</rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:about="#Actor">
    <status:term_status>unstable</status:term_status>
    <rdfs:label>Actor</rdfs:label>
    <rdfs:comment>The agent who performs the activity, modelled as atom:author</rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:about="#Annotation">
    <status:term_status>unstable</status:term_status>
    <rdfs:label>Annotation</rdfs:label>
    <rdfs:comment>an extra text-based note added to an activity by the user.</rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:about="#Application">
    <status:term_status>unstable</status:term_status>
    <rdfs:label>Application</rdfs:label>
    <rdfs:comment>An actor that is also an application.</rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:about="#Article">
    <status:term_status>unstable</status:term_status>
    <rdfs:label>Article</rdfs:label>
    <rdfs:comment>Articles generally consist of paragraphs of text, in some cases incorporating embedded media such as photos and inline hyperlinks to other resources.</rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:about="#Audio">
    <status:term_status>unstable</status:term_status>
    <rdfs:label>Audio</rdfs:label>
    <rdfs:comment>audio content.</rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:about="#Bookmark">
    <status:term_status>unstable</status:term_status>
    <rdfs:label>Bookmark</rdfs:label>
    <rdfs:comment>pointer to some URL -- typically a web page.</rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:about="#Comment">
    <status:term_status>unstable</status:term_status>
    <rdfs:label>Comment</rdfs:label>
    <rdfs:comment>A textual response to another object. The comment object type MUST NOT be used for other kinds of replies, such as video replies or reviews. If an object has no explicit type but the object element has a thr:in-reply-to element a consumer SHOULD consider that object to be a comment.</rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:about="#Contact">
    <status:term_status>unstable</status:term_status>
    <rdfs:label>Contact</rdfs:label>
    <rdfs:comment></rdfs:comment>
  </rdfs:Class>
</rdf:RDF>
```
<rdfs:Class rdf:about="#File"/>
<brdfs:term_status>unstable</brdfs:term_status>
<brdfs:subClassOf rdf:resource="#Object"/>
<brdfs:label>File</brdfs:label>
<brdfs:comment>a document or other file with no additional machine-readable semantics.</brdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="#Group"/>
<brdfs:term_status>unstable</brdfs:term_status>
<brdfs:subClassOf rdf:resource="#Object"/>
<brdfs:label>Group</brdfs:label>
<brdfs:comment>a collection of people which people can join and leave.</brdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="#GroupOfUsers"/>
<brdfs:term_status>unstable</brdfs:term_status>
<brdfs:subClassOf rdf:resource="#Actor"/>
<brdfs:label>GroupOfUsers</brdfs:label>
<brdfs:comment>An actor that is also a group of users.</brdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="#Join"/>
<brdfs:term_status>unstable</brdfs:term_status>
<brdfs:subClassOf rdf:resource="#Verb"/>
<brdfs:label>Join</brdfs:label>
<brdfs:comment>the Actor has become a member of the Object. This specification only defines the meaning of this Verb when its Object is a group.</brdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="#Located"/>
<brdfs:term_status>unstable</brdfs:term_status>
<brdfs:subClassOf rdf:resource="#Verb"/>
<brdfs:label>Located</brdfs:label>
<brdfs:comment>the Actor is located in the Object.</brdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="#Location"/>
<brdfs:term_status>unstable</brdfs:term_status>
<brdfs:subClassOf rdf:resource="#Context"/>
<brdfs:label>Location</brdfs:label>
<brdfs:comment>the location where the user was at the time the activity was performed. This may be an accurate geographic coordinate, a street address, a free-form location name or a combination of these.</brdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="#MediaCollection"/>
<brdfs:term_status>unstable</brdfs:term_status>
<brdfs:subClassOf rdf:resource="#Object"/>
<brdfs:label>MediaCollection</brdfs:label>
<brdfs:comment>Generic collection of media items.</brdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="#MediaContent"/>
<brdfs:term_status>unstable</brdfs:term_status>
<brdfs:subClassOf rdf:resource="#Object"/>
<brdfs:label>MediaContent</brdfs:label>
<brdfs:comment>a media item.</brdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="#MakeFriend"/>
<brdfs:term_status>unstable</brdfs:term_status>
<brdfs:subClassOf rdf:resource="#Verb"/>
<brdfs:label>MakeFriend</brdfs:label>
<brdfs:comment>the Actor sets the creation of a friendship that is reciprocated by the object.</brdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="#MarkAsFavorite"/>
<brdfs:term_status>unstable</brdfs:term_status>
<brdfs:subClassOf rdf:resource="#Verb"/>
<brdfs:label>MarkAsFavorite</brdfs:label>
<brdfs:comment>the Actor marked the Object as an item of special interest.</brdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="#Mood"/>
<brdfs:term_status>unstable</brdfs:term_status>
<brdfs:subClassOf rdf:resource="#Context"/>
<brdfs:label>Mood</brdfs:label>
<brdfs:comment>the mood of the user when the activity was performed. This is usually collected via an extra field in the user interface used to perform the activity. For the purpose of this schema, a mood is a freeform, short mood keyword or phrase along with an optional mood icon image.</brdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="#Note"/>
<brdfs:term_status>unstable</brdfs:term_status>
<brdfs:subClassOf rdf:resource="#Object"/>
<brdfs:label>Note</brdfs:label>
<brdfs:comment>is intended for use in "micro-blogging" and in systems where users are invited to publish a timestamped "status".</brdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="#Person"/>
<brdfs:term_status>unstable</brdfs:term_status>
<brdfs:subClassOf rdf:resource="#Object"/>
<brdfs:label>Person</brdfs:label>
<brdfs:comment>an individual user of the service.</brdfs:comment>
</rdfs:Class>
<status:term_status>unstable</status:term_status>

Deliverable 3.1

FP7 – 231761

StartFollowing

is a song rather than merely a generic audio stream.

In the latter case, the song SHOULD also be annotated with the "audio" object type and use its properties. Type "song" SHOULD only be used when the publisher can guarantee that the object is a song rather than merely a generic audio stream.

Share

the Web Service where the activity is performed by the Actor.

RSVP

indicates that the actor has made a RSVP ("Rpondez s’il vous plaît") for the object, that is, he/she replied to an invite. This specification only defines the meaning of this verb when its object is an event. The use of this verb is only appropriate when the RSVP was created by an explicit action by the actor. It is not appropriate to use this verb when a user has been added as an attendee by an event organiser or administrator.

Save

the Actor has called out the Object as being of interest primarily to him- or herself. Though this action MAY be shared publicly, the implication is that the Object has been saved primarily for the actor’s own benefit rather than to show it to others as would be indicated by the “share” Verb.

Replies

a location on Earth.

Place

a collection of images.

PhotoAlbum

a graphical still image.

Photo

A user account. This is often a person, but might also be a company or fictitious character that is being represented by a user account.

User

is a user account. This is often a person, but might also be a company or fictitious character that is being represented by a user account.

Person

projects. Type "song" SHOULD only be used when the publisher can guarantee that the object is a song rather than merely a generic audio stream.

Song

an ordered list of time-based media items, such as video and audio objects.

Playlist

a collection of images.

PhotoAlbum

a graphical still image.

Photo

a collection of images.

PhotoAlbum

a graphical still image.

Photo

a graphical still image.

Photo

a user account. This is often a person, but might also be a company or fictitious character that is being represented by a user account.

Person

projects. Type "song" SHOULD only be used when the publisher can guarantee that the object is a song rather than merely a generic audio stream.

Song

an ordered list of time-based media items, such as video and audio objects.

Playlist

a collection of images.

PhotoAlbum

a graphical still image.

Photo

a user account. This is often a person, but might also be a company or fictitious character that is being represented by a user account.

Person

projects. Type "song" SHOULD only be used when the publisher can guarantee that the object is a song rather than merely a generic audio stream.

Song

an ordered list of time-based media items, such as video and audio objects.

Playlist

a collection of images.

PhotoAlbum

a graphical still image.

Photo

a user account. This is often a person, but might also be a company or fictitious character that is being represented by a user account.

Person

projects. Type "song" SHOULD only be used when the publisher can guarantee that the object is a song rather than merely a generic audio stream.

Song

an ordered list of time-based media items, such as video and audio objects.

Playlist

a collection of images.

PhotoAlbum

a graphical still image.

Photo

a user account. This is often a person, but might also be a company or fictitious character that is being represented by a user account.

Person

projects. Type "song" SHOULD only be used when the publisher can guarantee that the object is a song rather than merely a generic audio stream.

Song

an ordered list of time-based media items, such as video and audio objects.

Playlist

a collection of images.

PhotoAlbum

a graphical still image.

Photo

a user account. This is often a person, but might also be a company or fictitious character that is being represented by a user account.

Person

projects. Type "song" SHOULD only be used when the publisher can guarantee that the object is a song rather than merely a generic audio stream.

Song

an ordered list of time-based media items, such as video and audio objects.

Playlist

a collection of images.

PhotoAlbum

a graphical still image.

Photo

a user account. This is often a person, but might also be a company or fictitious character that is being represented by a user account.

Person

projects. Type "song" SHOULD only be used when the publisher can guarantee that the object is a song rather than merely a generic audio stream.

Song

an ordered list of time-based media items, such as video and audio objects.

Playlist

a collection of images.

PhotoAlbum

a graphical still image.

Photo

a user account. This is often a person, but might also be a company or fictitious character that is being represented by a user account.

Person

projects. Type "song" SHOULD only be used when the publisher can guarantee that the object is a song rather than merely a generic audio stream.

Song

an ordered list of time-based media items, such as video and audio objects.

Playlist

a collection of images.

PhotoAlbum

a graphical still image.

Photo

a user account. This is often a person, but might also be a company or fictitious character that is being represented by a user account.

Person

projects. Type "song" SHOULD only be used when the publisher can guarantee that the object is a song rather than merely a generic audio stream.

Song

an ordered list of time-based media items, such as video and audio objects.

Playlist

a collection of images.

PhotoAlbum

a graphical still image.

Photo

a user account. This is often a person, but might also be a company or fictitious character that is being represented by a user account.

Person

projects. Type "song" SHOULD only be used when the publisher can guarantee that the object is a song rather than merely a generic audio stream.

Song

an ordered list of time-based media items, such as video and audio objects.

Playlist

a collection of images.

PhotoAlbum

a graphical still image.

Photo

a user account. This is often a person, but might also be a company or fictitious character that is being represented by a user account.

Person

projects. Type "song" SHOULD only be used when the publisher can guarantee that the object is a song rather than merely a generic audio stream.

Song

an ordered list of time-based media items, such as video and audio objects.

Playlist

a collection of images.

PhotoAlbum

a graphical still image.

Photo

a user account. This is often a person, but might also be a company or fictitious character that is being represented by a user account.

Person

projects. Type "song" SHOULD only be used when the publisher can guarantee that the object is a song rather than merely a generic audio stream.

Song

an ordered list of time-based media items, such as video and audio objects.

Playlist

a collection of images.

PhotoAlbum

a graphical still image.

Photo

a user account. This is often a person, but might also be a company or fictitious character that is being represented by a user account.

Person

projects. Type "song" SHOULD only be used when the publisher can guarantee that the object is a song rather than merely a generic audio stream.

Song

an ordered list of time-based media items, such as video and audio objects.

Playlist

a collection of images.

PhotoAlbum

a graphical still image.

Photo

a user account. This is often a person, but might also be a company or fictitious character that is being represented by a user account.

Person

projects. Type "song" SHOULD only be used when the publisher can guarantee that the object is a song rather than merely a generic audio stream.

Song

an ordered list of time-based media items, such as video and audio objects.

Playlist

a collection of images.

PhotoAlbum

a graphical still image.

Photo

a user account. This is often a person, but might also be a company or fictitious character that is being represented by a user account.

Person

projects. Type "song" SHOULD only be used when the publisher can guarantee that the object is a song rather than merely a generic audio stream.

Song

an ordered list of time-based media items, such as video and audio objects.

Playlist

a collection of images.

PhotoAlbum

a graphical still image.

Photo

a user account. This is often a person, but might also be a company or fictitious character that is being represented by a user account.

Person

projects. Type "song" SHOULD only be used when the publisher can guarantee that the object is a song rather than merely a generic audio stream.

Song

an ordered list of time-based media items, such as video and audio objects.

Playlist

a collection of images.

PhotoAlbum

a graphical still image.

Photo

a user account. This is often a person, but might also be a company or fictitious character that is being represented by a user account.

Person

projects. Type "song" SHOULD only be used when the publisher can guarantee that the object is a song rather than merely a generic audio stream.

Song
<rdf:comment>The Actor began following the activity of the Object. In most cases, the Object of this Verb will be a user, but it can potentially be of any type that can sensibly generate activity.</rdf:comment>
Deliverable 3.1

FP7 – 231761

[Image 446x791 to 517x804]
Deliverable 3.1
FP7 – 231761

<rdfs:domain rdf:resource="#Service"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>

<owl:DatatypeProperty rdf:about="#playerApplet">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:domain rdf:resource="#Object"/>
  <rdfs:domain rdf:resource="#Actor"/>
  <rdfs:comment>The URL and metadata for some kind of applet that will allow a user to view the video. The URL is represented as the value of the href attribute on an atom:link element with rel alternate and an appropriate type. Publishers SHOULD include media:width and media:height attributes on the atom:link element describing the ideal dimensions of the linked applet. <rdfs:comment>
</rdfs:comment>
</owl:DatatypeProperty>

<rdfs:domain rdf:resource="#Video"/>
<rdfs:domain rdf:resource="#Photo"/>
<rdfs:domain rdf:resource="#MediaCollection"/>
<rdfs:domain rdf:resource="#Bookmark"/>

<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>

<rdfs:domain rdf:resource="#ServiceUrl"/>

<owl:DatatypeProperty rdf:about="#RSVPConnotation">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<rdfs:domain rdf:resource="#RSVP"/>

<rdfs:domain rdf:resource="#Bookmark"/>

<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>

<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>

<owl:DatatypeProperty rdf:about="#serviceUrl">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#url"/>
</owl:DatatypeProperty>

<rdfs:domain rdf:resource="#RSVP"/>

<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>

<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>

<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>

A.2 Weighted Interests

  <dc:title>A vocabulary for weighted interests</dc:title>
  <rdfs:comment>Draft vocabulary for describing preferences within contexts</rdfs:comment>
  <rdfs:Class rdf:about="http://xmlns.notu.be/wi#WeightedInterest">
    <rdfs:label>A Weighted Interest</rdfs:label>
    <rdfs:comment>A weighted interest object</rdfs:comment>
  </rdfs:Class>
  <rdfs:Class rdf:about="http://xmlns.notu.be/wi#Context">
    <rdfs:label>A Context</rdfs:label>
    <rdfs:comment>A context object</rdfs:comment>
  </rdfs:Class>
</owl:Ontology>
A Context object

A preference

A link between an agent and a weighted interest

A link between an agent and a topic of no interest to them

A link between a context and evidence supporting the interpretation of preferences in a context

A topic of the weighted interest

A link between a WeightedInterest and Context

A context location

A document describing a device

A time period of a context

A link between a WeightedInterest and Context

A location

A device

A time period

A Context

A link between a WeightedInterest and Context

A link between a WeightedInterest and Context

A location

A device

A time period

A Context

A link between a WeightedInterest and Context

A link between a WeightedInterest and Context

A location

A device

A time period

A Context

A link between a WeightedInterest and Context

A link between a WeightedInterest and Context

A location

A device

A time period

A Context

A link between a WeightedInterest and Context

A link between a WeightedInterest and Context

A location

A device

A time period

A Context

A link between a WeightedInterest and Context

A link between a WeightedInterest and Context

A location

A device

A time period

A Context

A link between a WeightedInterest and Context

A link between a WeightedInterest and Context

A location

A device

A time period

A Context

A link between a WeightedInterest and Context

A link between a WeightedInterest and Context

A location

A device

A time period

A Context

A link between a WeightedInterest and Context

A link between a WeightedInterest and Context

A location

A device

A time period

A Context

A link between a WeightedInterest and Context

A link between a WeightedInterest and Context

A location

A device

A time period

A Context

A link between a WeightedInterest and Context

A link between a WeightedInterest and Context

A location

A device

A time period

A Context

A link between a WeightedInterest and Context

A link between a WeightedInterest and Context

A location

A device

A time period

A Context

A link between a WeightedInterest and Context

A link between a WeightedInterest and Context

A location

A device

A time period

A Context

A link between a WeightedInterest and Context

A link between a WeightedInterest and Context

A location

A device

A time period

A Context

A link between a WeightedInterest and Context

A link between a WeightedInterest and Context

A location

A device

A time period

A Context

A link between a WeightedInterest and Context

A link between a WeightedInterest and Context

A location

A device

A time period

A Context

A link between a WeightedInterest and Context

A link between a WeightedInterest and Context

A location

A device

A time period

A Context

A link between a WeightedInterest and Context

A link between a WeightedInterest and Context

A location

A device

A time period

A Context

A link between a WeightedInterest and Context

A link between a WeightedInterest and Context

A location

A device

A time period

A Context

A link between a WeightedInterest and Context

A link between a WeightedInterest and Context

A location

A device

A time period

A Context

A link between a WeightedInterest and Context

A link between a WeightedInterest and Context

A location

A device

A time period

A Context
<owl:Class rdf:about="http://ontologi.es/days#TuesdayInterval"/>
<owl:Class rdf:about="http://ontologi.es/days#TuesdayInstant"/>
<owl:Class rdf:about="http://ontologi.es/days#WednesdayInterval"/>
<owl:Class rdf:about="http://ontologi.es/days#WednesdayInstant"/>
<owl:Class rdf:about="http://ontologi.es/days#ThursdayInterval"/>
<owl:Class rdf:about="http://ontologi.es/days#ThursdayInstant"/>
<owl:Class rdf:about="http://ontologi.es/days#FridayInterval"/>
<owl:Class rdf:about="http://ontologi.es/days#FridayInstant"/>
<owl:Class rdf:about="http://ontologi.es/days#SaturdayInterval"/>
<owl:Class rdf:about="http://ontologi.es/days#SaturdayInstant"/>
<owl:Class rdf:about="http://ontologi.es/days#SundayInterval"/>
<owl:Class rdf:about="http://ontologi.es/days#SundayInstant"/>
<owl:Class rdf:about="http://ontologi.es/days#WeekdayInterval"/>
<owl:Class rdf:about="http://ontologi.es/days#WeekdayInstant"/>
<owl:Class rdf:about="http://ontologi.es/days#WeekendDayInterval"/>
<owl:Class rdf:about="http://ontologi.es/days#WeekendDayInstant"/>
<owl:Class rdf:about="http://ontologi.es/days#DayInterval"/>
<owl:Class rdf:about="http://ontologi.es/days#DayInstant"/>
<owl:Class rdf:about="http://ontologi.es/days#HolidayInterval"/>
<owl:Class rdf:about="http://ontologi.es/days#HolidayInstant"/>
</rdfs:Class>

<rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
<rdfs:comment>The weight on the topic</rdfs:comment>
<rdfs:domain rdf:resource="http://xmlns.notu.be/wi#WeightedInterest"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="http://xmlns.notu.be/wi#scale"/>
<rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
<rdfs:comment>The scale with respect to the weight - of the form 0..9. Scale can be any range of integers</rdfs:comment>
<rdfs:domain rdf:resource="http://xmlns.notu.be/wi#WeightedInterest"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<!-- other classes referred to but not defined here -->
<!-- the labels are required in order for some of the queries to work properly -->
<rdf:Class rdf:about="http://www.w3.org/2001/XMLSchema#int"/>
<rdf:isDefinedBy rdf:resource="http://www.w3.org/2001/XMLSchema#"/>
<rdfs:label>Integer</rdfs:label>
</rdf:Class>
<rdf:Class rdf:about="http://www.w3.org/2001/XMLSchema#string"/>
<rdf:isDefinedBy rdf:resource="http://www.w3.org/2001/XMLSchema#"/>
<rdfs:label>String</rdfs:label>
</rdf:Class>
<rdf:Class rdf:about="http://www.w3.org/2004/02/skos/core#Concept"/>
<rdf:isDefinedBy rdf:resource="http://www.w3.org/2004/02/skos/core#"/>
<rdfs:label>Concept</rdfs:label>
</rdf:Class>
<rdf:Class rdf:about="http://xmlns.com/foaf/0.1/Document"/>
<rdf:isDefinedBy rdf:resource="http://xmlns.com/foaf/0.1/"/>
<rdfs:label>Document</rdfs:label>
</rdf:Class>
<rdf:Class rdf:about="http://xmlns.com/foaf/0.1/Agent"/>
<rdf:isDefinedBy rdf:resource="http://xmlns.com/foaf/0.1/"/>
<rdfs:label>Agent</rdfs:label>
</rdf:Class>
<rdf:Property rdf:about="http://www.w3.org/2003/01/geo/wgs84_pos#location"/>
<rdf:isDefinedBy rdf:resource="http://www.w3.org/2003/01/geo/wgs84_pos#"/>
<rdfs:label>Location</rdfs:label>
</rdf:Property>
</rdf:NDF>