The NoTube BeanCounter: Aggregating User Data for Television Programme Recommendation

Chris van Aart[1], Lora Aroyo[1], Dan Brickley[13], Vicky Buser[2], Libby Miller[2], Michele Minno[3], Michele Mostarda[3], Davide Palmisano[3], Yves Raimond[2], Guus Schreiber[1], and Ronald Siebes[1]

[1] VU University Amsterdam, the Netherlands
[2] BBC - Future Media & Technology, London, UK
[3] Asemantics Srl, Rome Italy

Abstract. In this paper we present our current experience of aggregating user data from various Social Web applications and outline several key challenges in this area. The work is based on a concrete use case: reusing activity streams to determine a viewer's interests and generating television programme recommendations from these interests. Three system components are used to realise this goal: (1) an intelligent remote control: iZapper for capturing viewer activities in a cross-context television environment; (2) a backend: BeanCounter for aggregation of viewer activities from the iZapper and from different social web applications; and (3) a recommendation engine: iTube for recommending relevant television programmes . The focus of the paper is the BeanCounter as the first step to apply Social Web data for viewer and context modelling on the Web. This is work in progress of the NoTube project[4].

1 Introduction: Television Meets the Social Web

Television watching has always been a social activity, but as channels and delivery mechanisms multiply, the enablers of social watching - simultaneous broadcast times for everyone and limited choice - are decreasing[7]. There is a trend towards watching television or using television-related services[1] such as Electronic Programme Guides (EPGs) on PC, handheld or other networked device.

At the same time, other Web-based opportunities for social interaction are increasing. The next generation television audience usually has a substantial *online presence* through social networking and other online services. In those environments, users are typically represented by their user profiles (explicitly filled in) and the log of their online activities, e.g. "Jane uploaded a photo", "Bob is listening to French rap music". Such data can be (in)directly useful for deriving specific user interests in *various* domains. Most of this data, however, is still locked within a single application, or is only partially reused across applications, e.g., usernames and basic personal data. Moreover, much of this personal data

[4] http://www.notube.tv

is also duplicated across the various social sites. The user is left with *multiple user profiles*, which are not or weakly related (syntactically only) to each other.

Maintaining rich profiles of a wide variety of content (in various formats and areas) is critical for providing adequate personalisation services. *Recommendation services* are a typical example of applying such rich user profiles. However, most approaches today are still limited to simple recommendations ("Take a look at this programme"), similarity-based recommendations ("if you watch this programme, you will probably like this other one in the same category") or collaborative filtering (such as Amazon's "Customers who bought this item also bought..."). Services for television recommendations, such as 4IP's "Test Tube Telly"[5], tend to be restricted to using data from one system only. An advanced mechanism for social recommendations will now need to consider a completely different landscape, with different profiles of users, user groups and audience segments. Opening and sharing of profiles and attention data between different players in the market creates a different environment - *an open social graph where a wider user base can potentially help to improve the quality of recommendations and reduce the costs of moderation and spam filtering.*

In the *NoTube project* [3] we develop a flexible end-to-end architecture, based on semantic technologies, for personalised creation, distribution and consumption of television content. The project takes a user-centric approach to investigate fundamental aspects of consumers' content-customisation needs, interaction requirements and entertainment wishes, which will shape the future of "television" in all its new forms. In this paper we focus on the first part of this project - combining multiple heterogeneous sources of data, with the addition of semantics, in order to create machine-readable profiles for users. These can later be used to drive better, more focused and more personalised television and other media recommendation services in a personalised, service-based EPG (Electronic Programme Guide)[4]. NoTube aims to overcome a number of limitations in current EPGs. Imagine how to recommend and search for programmes that are: (a) non-fiction, (b) produced between 1989 and 1995, (c) involve locations in Eastern Europe. In current EPGs (a) is resolvable, while (b) is not, but it is possible if the programme information has been properly indexed along a given timeline; (c) also requires appropriate metadata, and a sophisticated knowledge model that allows to represent and reason about part/whole relations between places, countries and regions (e.g., Sofia is in Bulgaria; Bulgaria is in Eastern Europe). In the project we hope to gain insights into *end-user control*, *user understanding of aggregation of data about them*, *privacy-preserving architectures*, and *constraints on reuse of data* as data of this kind is both potentially privacy invasive and also valuable.

In this paper we describe the design and implementation aspects of the *BeanCounter* as the user data collecting component, aiming to illustrate its potential application in the Social Web domain. The main design rationale is to provide a flexible and extensible architecture that exposes robust, scalable and reliable services to handle different kind of responses of different social applications plat-

[5] http://testtubetelly.channel4.com/

forms. A set of APIs allows for modelling the targeted responses to gather them, represent them with a set of suitable RDF vocabularies, and integrate them with other pulled information in a fully transparent way. Thus, we outline a loosely coupled architecture based on service-specific adaptors (*tubelet*) and application servers (*modelet*), which allow for the selection of data source and RDF vocabulary and the generation of RDF-ised user data, integrated with data coming from other adaptors. We aim to create a set of language-independent, RESTful APIs that allow for writing adaptors, analysers, data management and enrichment components to different user data services. An important requirement is maintaining a transparent privacy policy useful to users with different levels of technical skills.

In the following sections we report on our work on the BeanCounter with usage scenarios, alignment approaches, design and implementation details of the BeanCounter implementation, and lessons learned. In these lessons we outline some of the challenges related to realising not only the data collection component, but the whole workflow for achieving reuse of user data from Social Web.

2 Beancounter Usage Scenarios

2.1 Scenarios for end-users

Bob has accounts at several social websites: Facebook, Twitter, YouTube, and Last.fm. He regularly uses NoTube 'favourites' feature. For each website he has a separate identity and has indicated a basic set of personal information and interests. Each application also carries his (partial) user profile, with his interests related to that application, e.g., music preferences or travel history. There is limited integration of such user profiles, typically not under user's control and lacking transparency of how data is interpreted in different applications. It is difficult for Bob to find out what each system knows about him and how this knowledge is derived. Moreover, he cannot easily find out his interests and personal statistics based on what he watches, listens to or consumes on different devices (e.g., interactive television, online TV guide, Netflix, YouTube). Using the BeanCounter, Bob can:

- **log into the BeanCounter** using his openID (or create an account).
- **link a few of his accounts and devices to the BeanCounter**, e.g., his Twitter and YouTube accounts. He could also link an iZapper (a combined EPG, remote, and context capturing device) to his account. After linking, the service-specific adaptors ('tubelets') pull the data, convert it to an activity stream in RDF using service-specific algorithms, and store it in a triple store.
- **view some statistics about himself**. Bob can immediately see some interesting information about himself in the *BeanCounter web Interface* (see Figure 1). A machine-readable profile can also be created.
- **edit the result**, e.g., delete all the data or remove sources. He can decide which part of the profile to make public, if any. The final result is a machine-readable profile that Bob uses to describe himself on the Semantic Web.

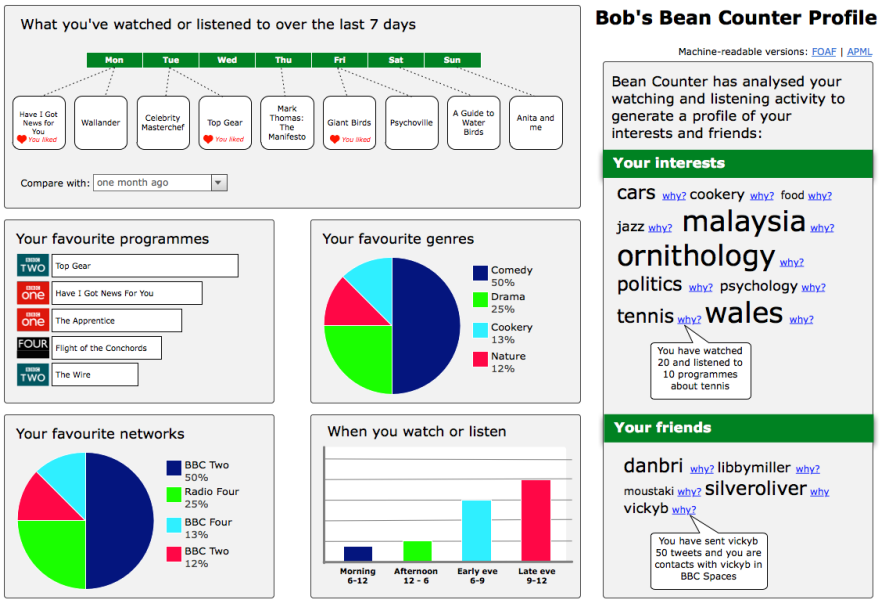

Fig. 1. The BeanCounter UI for End-Users

2.2 Scenarios for technical end-user

Jane is privacy-conscious, technically minded and able to do some programming. She has the latest television-PVR that outputs data using the XMPP protocol; her friends have similar devices. She uses an iZapper to indicate programmes she watches, records or likes and dislikes. For example, Jane can:

Download and run her own version of the BeanCounter and **attach her BeanCounter to her and her friends' PVR accounts, with their permission**. In this way, she can make queries over the combined set of interests to see if she can make some recommendations about what to watch, by querying the dataset to see what is mostly commonly watched and then querying schedules to filter what's on next week.

2.3 Scenarios for developers

George is a developer who wants to **use the BeanCounter for tracking attention data from YouTube**. Suppose an instance of the BeanCounter is currently deployed and is running on `beancounter.asemantics.com`, but is pulling data only from Twitter. George wants to track YouTube favourites as well. He studies the response (Atom feed) from the YouTube API and writes couple of Java classes to represent this data. He chooses an appropriate authentication mechanism (e.g., http Basic Auth, or OAuth). Then he needs to

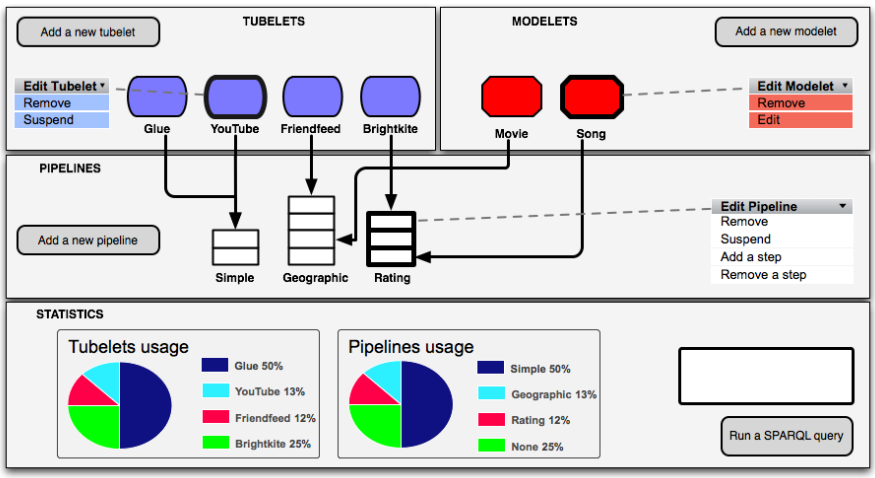

Fig. 2. BeanCounter UI for Developers

select the appropriate RDF vocabulary to represent this user data. FOAF is suitable to represent the static part of the YouTube profile (e.g., the property *foaf:interest* links users to the tags associated with every YouTube favourite video. All he needs is to add a minimal set of annotation to his Java classes stating this. Once the lines of code are written, he uses the Management Interface `beancounter.asemantics.com/manager` and adds the *YouTube Tubelet* to the BeanCounter. The tubelet is immediately available to be used without restarting the whole service.

Consider now Chris, who is a developer at OpenCalais annotation service [6] and **wants to enrich identi.ca tweets**. The goal is to build a *Identi.ca Tubelet* that pulls the Identi.ca tweets, annotates them using the Opencalais and stores the resulting triples. Similarly to the previous scenario, Chris first needs to make a new tubelet. Then, he writes a small Java class called a *Pipe*, to indicate that Opencalais service is to be used to process the text of the tweet, and finally, he binds the new tubelet to that pipe. Thus, the output of this tubelet will be processed by this pipe and then stored (see Figure 2). The same scenario can be realised with other annotation services, such as KIM annotation service[7].

Another example is Anna, a BBC producer, who *wants to find out whether people liked the new episode of 'Torchwood'*. Suppose the BBC has an instance of the BeanCounter running and aggregates and anonymises data using the public APIs from Twitter and Facebook combined with statistics from iPlayer. Anna writes custom queries for Torchwood and a custom analyser that evaluates whether each activity data item was positive or negative. With the 'aggregation'

[6] http://viewer.opencalais.com
[7] http://www.ontotext.com/kim

feature - over all user data in the system - she creates custom queries to find out what the overall opinions were and when people tended to watch it.

3 BeanCounter Metadata and Resources

More and more data gets published in RDF[8] most notably as linked data[6], [5]: generic knowledge such as DBpedia[8], domain data such as instances from the BBC Programmes Ontology[9], linguistic data such as W3C Wordnet[10], service data like instances of WSMO-lite[11], or user data like FOAF[12] instances or MySpace metadata[13]. This knowledge can be accessed in various ways[9]: SPARQL endpoints, Java APIs, and REST-style Web-services. Programmers combine these sources to create migration services for other programmers or end-user (mashup) applications. In developing the BeanCounter we have experienced five key aspects relevant to the enrichment of the user attention data:

- **Selection of resources:** The growth of the Semantic data cloud allows us to integrate more external data sources relevant to the NoTube domain. The selection of the right sources is a non-trivial challenge. First, we need to list candidate sources that possibly could contribute to our user-scenarios. Second, for those sources we need to estimate the quality in terms of completeness, reasoning complexity, errors and stability. Third, we need to determine the effort needed to align the schemas to connect the source with the other NoTube vocabularies.
- **Creation of alignments:** The growth of the Semantic data cloud allows more and more interesting alignments for the NoTube domain between the different data sources in the cloud. For example, in one of the case studies we are working on making alignments in SKOS to aligning the Last.fm music categories with programme data described in the BBC Programmes ontology.
- **Creation of connections:** The linked data cloud contains commonly used URIs representing entities suitable for reuse, for example Dbpedia concepts. Where possible we reuse URIs in this way to create a more connected graph. The challenge is to find and easily reuse the relevant URIs
- **Usage of the alignments and connections:** When the alignments are created, we cannot assume that all data that we want to align will be in one single SPARQL repository. For this we need to develop alignment services that themselves generate RDF on request or have another interface (e.g., REST-style). For example, a type of alignment that gets special attention in NoTube because of non-English content provided in the case-studies: multi-lingual aspects related to mapping Korean, Italian, Dutch and German

[8] http://dbpedia.org/
[9] http://purl.org/ontology/po/
[10] http://www.w3.org/TR/wordnet-rdf/
[11] http://www.wsmo.org/ns/wsmo-lite/
[12] http://xmlns.com/foaf/spec/
[13] http://grasstunes.net/ontology/myspace/myspace.html

content to English (and back). For example, we are working on mapping the Dutch Cornetto 'Wordnet' to the W3C Wordnet, to obtain more (English) information for topics annotated originally in Dutch.

– **Domain dynamics:** The world is dynamic, so also the metadata describing this world, such as user profile data, programmes and similar. It is the challenge to get the alignments as quickly as possible and to inform the interested parties. For this a publish/subscribe mechanism where people can place a 'listener' on the topics of interest would make life easier. Current research such as Jqbus (a SPARQL service over the XMPP protocol) is ongoing to tackle this challenge.

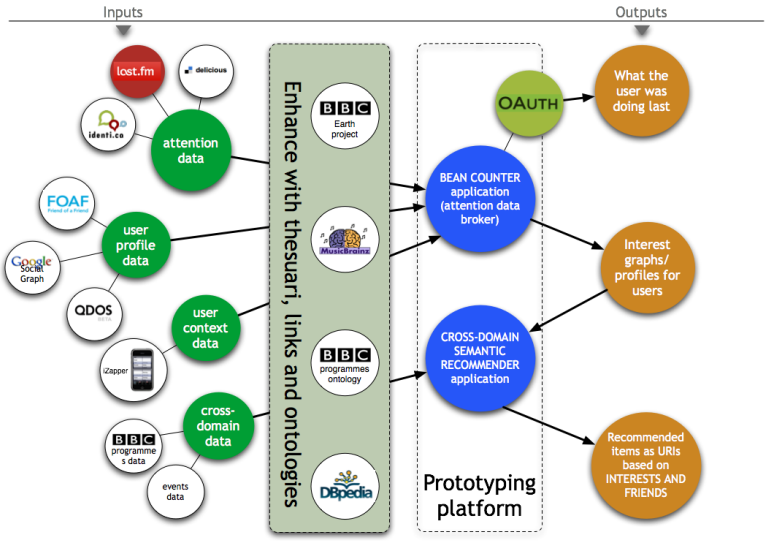

Fig. 3. NoTube Recommendation Architecture

The BeanCounter is a user data aggregation component: it aggregates data about a user from Web sources (e.g., Facebook, Last.FM, Twitter) and produces a machine-readable interest profile for this user. The BeanCounter produces also a human-readable version of this profile by emphasising on the transparency and privacy-preserving aspects, as well as on the adequate and appealing presentation of the information. However, in the context of this paper, we focus on its input to semantic recommendation services. In the prototype version of the overall NoTube architecture (see Figure 3) the BeanCounter is accompanied by a controlling device *iZapper* that captures a viewer's log in specific contexts (e.g., I am at home, working, in the evening) and a viewer/recommender service *iTube*, which uses the input from the BeanCounter and iZapper in order to generate and present the recommended relevant television programmes to the

user. iTube is typically a media centre environment, which can make use of the semantic recommendation service's output via an API. It could also be a display on a computer or a smaller device. Currently, we plan to use the iFanzy[2] recommendation service, although the architecture will allow for many different recommendations services to be used.

4 Prototype Architecture

4.1 iZapper: the controlling device

Traditionally televisions are controlled with dedicated remote controls. To develop a viewer's profile, the viewer has to explicitly configure it. iZapper (see Figure 4) is a native iPhone application intended to control a media centre (typical device and channel control), capture a viewer's context, and to record viewer's activities to gain more information. Several types of contextual information can be retrieved from a user's iPhone, such as location (via GPS or wifi endpoint), period (via time) and optionally tasks (via agenda).

Fig. 4. iZapper User Interface: (a) Generating and (b) Representing User Activities

Television activities, such as watching, recording, ranking and bookmarking will be recorded. All this information will be pushed to the BeanCounter. Every

viewer has access to his personal iZapper. The BeanCounter will use the iZapper as one source of user information for context, activities and profile. The context is the circumstances and surroundings of a viewer, composed of location, period and task, and is relevant for the fact that viewers' ratings are potentially made to hold in a specific context.

4.2 The BeanCounter Architecture

The BeanCounter itself can be modelled as a *container* of components (called *tubelets*) responsible for calling and managing a service the user wants to import into the system. Thus, each Web source (e.g., FriendFeed, Twitter, Glue, last.fm) is wrapped and accessed by a *tubelet*, that is specifically built to handle data from that source. Whatever the format is adopted by the Web service response (e.g., ATOM, JSON or some sort of XML), a tubelet can parse such content and map it to RDF, allowing a native integration. All the logic for tubelet management are embedded in the container determining the life cycle of each tubelet.

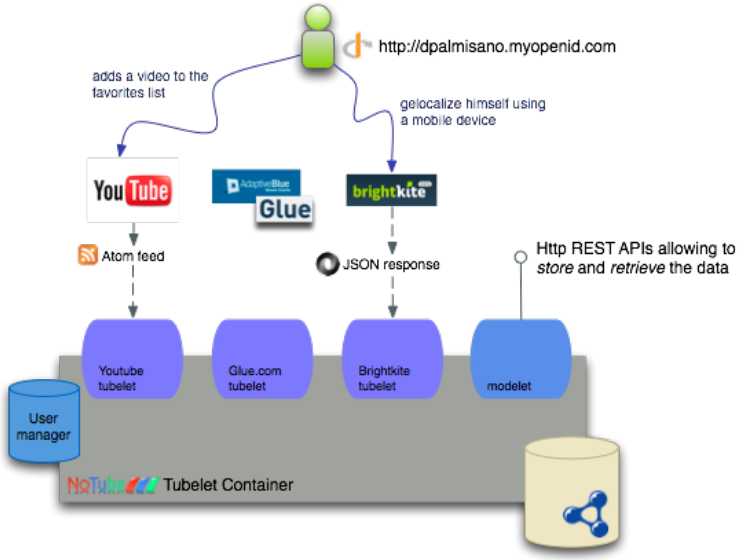

Fig. 5. The BeanCounter prototype Architecture

The activation of each tubelet involves (1) a *Scheduler:* scheduling the activation of the tubelet, (2) a *User ID Manager:* mapping all the different credentials of the user and (3) a computation of the range of data to be pulled out of the source. The tubelet takes as input the result of the specific source API (e.g., information about the user in that source), parses it and creates some Java beans

that model the data pulled from that source. The beans are serialised by the tubelet and passed as input to a *Pipeline*. The pipeline processes the serialised beans and produces an RDF representation of the data. It is essentially a pipe with a number of (optional) steps which eventually will produce RDF to be fed into the RDF storage. Apart from the mandatory step for RDF conversion, other additional steps could follow to perform some ad-hoc actions on this semantic representation. The described data flow is oriented to a specific Web source. Data are pulled directly from the source and some automatic lifting is made in order to get a semantic representation of the source content (as a set of triples).

Another flow is more model-driven rather than source-driven. A container is devised, where a *modelet* is the equivalent of a tubelet. A modelet can accept structured user data in a cross-source fashion. Every time a new piece of information is needed, a new modelet is plugged into the modelets container. A modelet feeds then into the proper java beans, which just as for the first flow are serialised and passed to a pipeline. A key aspect of the BeanCounter is illustrated by this last component. It allows for aggregation of data from different Web sources, where the same entity - movie, city, person - within two sources should be unambiguously identified and mapped in both sources. Adding a new modelet to the modelets container is possible by calling specific APIs defining the desired modelet (e.g., how it is structured, which fields) and then hot-plugging it into the container.

5 Discussion

During the requirements phase and implementation of two BeanCounters and the iZapper, we have identified several challenges to realise effective reuse of Social Web user data in domain-specific recommender system (for TV programmes). Some challenges have already been addressed in the development of BeanCounters. From this work result also some practical guidelines for semantic data integration. We now give a brief overview of those points, starting with Lessons Learned for Semantic Data Aggregators:

- **Pipelines**. To ingest data into the system through pipelines, which are dynamically editable (e.g., add new pipe elements in specific points of the chain). Some pipes could be responsible for collecting statistics on certain triple data, while others could handle the building of specific indexes.
- **Container Programming Pattern.** This pattern manages the addition and removal of components, as well as their life cycle, inter-communication and activation. It provides an abstraction that hides the complexity of the system, allowing developers to ignore low level functionalities. Some system components are subjected to change depending on the evolution of the services they are written for, so the system should support the addition and the replacement of dynamic components without requiring a restart.
- **Push, Pull and Trigger**. A flexible system should gather data (1) manually, (2) by scheduled activities for getting data from external services, and (3) in trigger mode with manual activation of collecting component

- **Hybrid Storage.** A triple store is good for persisting semi-structured data, but does not support structured data indexing (e.g., dates or geo coordinates). To get a good response time in data retrieval and filtering, a hybrid data storage handling indexes persisted in canonical Relational database tables and associated to RDF triples could be useful.
- **Avoid working directly with triples.** Using an RDF2Object mapping library allows the data handling logic to be expressed programmatically as object manipulation. It also encourages the (re)use of ontologies as libraries.
- **Native Support to Track Data Evolution.** Data retrieved in two subsequent points of time typically mixes old already ingested with new information. This introduces resource wastage and sometimes also data inconsistencies. Therefore, native support for managing the "delta" of data to avoid this kind of problems could be useful.

Challenges for User Data Reuse on the Social Web:

- **Presentation of Aggregated User Data**
  - How to maintain user's motivation to add new data and achieve good confluence of the content and the context?
  - What are non obtrusive ways of making the user aware of the privacy implications of her actions?
  - How to make user aware of the current (1) presentation context, (2) context model related to privacy and (3) adaption strategies?
  - How to keep user in control of profiles, content and context data used?
- **Privacy**
  - How to handle privacy at the architectural level?
  - How to achieve user management without asking users, e.g., for services username/password, or to create username/password for the BeanCounter?
  - How to help users understand the privacy issues when reusing and aggregating their data?
  - How to manage data reuse, e.g., by using creative commons licensing?
- **Context Capturing**
  - What are useful contexts, e.g., temporal, spatial, task-, device-related?
  - How to capture user's context (semi-)automatically, i.e., minimise the explicit user input, while maximising the background collection of user and context data, as well as deductions from this data?
- **Data Aggregation and Enrichment**
  - How to achieve unified and simple access to dynamic, growing and distributed multimedia content of diverse formats?
  - How to determine, which information from the activity stream is useful for determining user's interests and what enrichment is relevant, useful and accurate in different user contexts?
  - What is a minimal amount of data to start up the personalisation process, preventing a 'cold start'?
  - How to represent user data statistics, strength of user interest and user context in machine-readable format, e.g., RDF?
  - How to exploit current standards in television content metadata available both for providers and consumers

6 Conclusions

Our current experience of aggregating user data from various Social Web applications is presented in this paper. The BeanCounter is as a user data aggregation component for the NoTube architecture. The usage scenarios motivate the relation between TV recommendation and Social Web. To aggregate heterogeneous data, we have incorporated alignment support in our design and prototype implementation. Finally, we presented the lessons learned from the prototype implementation regarding the different challenges that need to be tackled and further work to be done.

References

1. L. Ardissono, A. Kobsa, and M. Maybury. *Personalized Digital Television: Targeting Programs to Individual Viewers (Human-Computer Interaction Series, 6)*. Kluwer Academic Publishers, Norwell, MA, USA, 2004.
2. L. Aroyo, P. Bellekens, M. Björkman, and G.-J. Houben. Semantic-based framework for personalised ambient media. *Multimedia Tools Appl.*, 36(1-2):71–87, 2008.
3. L. Aroyo, A. Kaptein, D. Palmisano, A. Conconi, L. Nixon, L. Vignaroli, S. Dietze, C. Nufer, and M. Yankova. Notube making tv a medium for personalized interaction. In *EuroITV 2009 Networked Television*, 2009.
4. P. Bellekens, L. Aroyo, G.-J. Houben, A. Kaptein, and K. van der Sluijs. Semantics-based framework for personalized access to tv content: The ifanzy use case. In *ISWC/ASWC*, pages 887–894, 2007.
5. C. Bizer, T. Heath, K. Idehen, and T. Berners-Lee. Linked data on the web (ldow2008). In *WWW*, pages 1265–1266, 2008.
6. T. B.-L. Christian Bizer, Tom Heath. Linked data - the story so far. *Journal on Semantic Web and Information Systems (IJSWIS)*, Special Issue on Linked Data(4), 2009.
7. T. Coppens, L. Trappeniers, and M. Godon. Amigotv: towards a social tv experience. In *EuroITV*, pages 159–162, Brighton, U.K., 2004.
8. E. M. Frank Manola. Rdf primer. 2004.
9. G. Kobilarov, T. Scott, Y. Raimond, S. Oliver, C. Sizemore, M. Smethurst, C. Bizer, and R. Lee. Media meets semantic web - how the bbc uses dbpedia and linked data to make connections. In *ESWC*, pages 723–737, 2009.

7 Acknowledgments

This work is supported by the EU IP Project NoTube, see http://www.notube.tv.