

**POLITECNICO DI MILANO**  
FACOLTÀ DI INGEGNERIA DELL'INFORMAZIONE  
Corso di Laurea in Ingegneria Informatica

**Lastnews: a semantic recommender system  
framework for web-contents**

Relatore: Prof. Marco Colombetti  
Correlatore: Ing. Davide Eynard

Tesi di laurea di:  
Andrea Mirone  
Matr. 735835

Anno Accademico 2010 - 2011



*Ego vero omnia in te cupio transfundere, et in hoc aliquid gaudeo discere, ut doceam;  
nec me ulla rebus delectabit, licet sit eximia et salutaris, quam mihi uni sciturus sum.*

*Si cum hac exceptione detur sapientia, ut illam inclusam teneam nec enuntiem  
reiciam: nullius boni sine socio iucunda possessio est.*

*I want to tell you everything I know, and this is the joy of learning something, to be  
able to teach it: nothing I learned, even if useful or important will bring me happiness,  
if I'm the only person to know it. If I was given all of the wisdom on the only  
condition I didn't share it with others, I would refuse: there is no joy in having  
something if you can't share it*

SENECA – EPISTULAE 6, 4

I CENTURY A.C.



# Contents

<b>Abstract</b>	<b>7</b>
<b>1 Introduction</b>	<b>9</b>
<b>2 Semantic web technologies</b>	<b>13</b>
2.1 Lower technologies of Semantic Web Stack . . . . .	14
2.2 OWL . . . . .	15
2.2.1 Syntax and semantics of description logic . . . . .	15
2.2.2 Relations between OWL, RDF and description logic . . . . .	19
2.3 SPARQL . . . . .	19
<b>3 Recommender System</b>	<b>27</b>
3.1 Memory based methods . . . . .	28
3.2 Model based methods . . . . .	30
3.3 Content Based Filtering . . . . .	34
3.4 Collaborative methods . . . . .	35
3.5 Hybrid methods . . . . .	37
3.6 Recommender system evaluation . . . . .	39
<b>4 Lastnews architecture</b>	<b>43</b>
4.1 Storage layer . . . . .	44
4.1.1 Openrdf-elmo . . . . .	45
4.2 Recommendation . . . . .	45
4.3 User feedback . . . . .	47
4.4 Item cycle . . . . .	48
<b>5 Lastnews Knowledge Base</b>	<b>51</b>
5.1 Representational language . . . . .	51

5.1.1	FOAF ontology . . . . .	51
5.1.2	SIOC ontology . . . . .	51
5.1.3	NEPOMUK Annotation ontology . . . . .	52
5.2	Item classification . . . . .	53
5.3	Item properties . . . . .	56
5.4	Item Status . . . . .	57
5.5	User properties . . . . .	59
5.6	Resource properties . . . . .	59
5.7	Plugin model . . . . .	60
<b>6</b>	<b>Lastnews Demo</b>	<b>63</b>
6.1	Tools used . . . . .	63
6.2	Implementation details . . . . .	64
6.2.1	Main objects implementing Lastnews . . . . .	64
6.2.2	GUI . . . . .	68
6.3	Plugins . . . . .	69
6.3.1	Plugin Behaviors . . . . .	70
6.3.2	Dependencies . . . . .	71
6.3.3	Twitter Retriever . . . . .	71
6.3.4	Bayesian Evaluator . . . . .	73
6.3.5	Roulette selector . . . . .	75
6.4	Configuring and running the demo . . . . .	77
<b>7</b>	<b>Conclusions</b>	<b>81</b>
	<b>Appendices</b>	<b>83</b>
<b>A</b>	<b>A short history of data indexing and the Web</b>	<b>85</b>
A.1	Pre-Internet era . . . . .	85
A.2	The Web 0.x . . . . .	86
A.3	The Web 1.0 . . . . .	87
A.4	The Web 2.0 . . . . .	88
A.4.1	Social Networks sites . . . . .	89
A.4.2	Bookmarking sites . . . . .	90
A.4.3	Micro-blogging sites . . . . .	91
A.4.4	Wiki sites . . . . .	91
A.4.5	Curation sites . . . . .	91

**CONTENTS** **5**

---

A.5 Real time web and beyond . . . . . 92

**B Libraries required by Lastnews** **93**

**Bibliography** **95**

**List of figures** **101**

**List of tables** **103**



# Abstract

Recommender systems are software tools able to provide advice to user about items they might find interesting. With the rise of social networks more and more users publish data on the Web making this kind of instruments a fundamental resource to overcome the information overload. In this work we present a framework for implementing and mixing recommender system algorithms. Starting from existing work we defined an ontology describing concepts useful for capturing user preference and implemented a software architecture that makes use of it to perform gathering and recommendation of contents from social network sites.

L'ascesa dei social network e la conseguente esplosione della quantità di informazioni che vengono ogni giorno prodotte e ritrasmesse sul Web ha dato origine in tempi recenti al fenomeno noto come "information overload", ovvero l'incapacità di cogliere contenuti importanti causata da eccesso di informazione. Negli ultimi anni l'intelligenza artificiale ha fornito fondamentali contributi nell'aiutare gli utenti a filtrare informazioni rilevanti dalla rete: tecniche mediate da altri campi come l'information retrieval e il data mining sono state applicate con successo per costruire agenti software capaci di fornire suggerimenti efficaci su oggetti potenzialmente d'interesse per l'utente. Questi software, noti come sistemi di raccomandazione, sebbene ormai studiati da diversi anni, sembrano avere ancora spazio per miglioramenti: di fatto essi sfruttano una moltitudine di approcci e di tecniche anche molto diverse tra loro nell'affrontare il problema e non si ha ancora una metodologia chiaramente vincente.

Parallelamente vi è un'attiva area di ricerca, fortemente sostenuta dal W3C, che si propone di migliorare la ricerca di informazioni attraverso la definizione di standard utili a dotare di semantica formale i contenuti presenti sulla rete, andando a costruire il cosiddetto "Web 3.0" o web semantico.

L'idea alla base di questo lavoro è che questi due approcci possano essere uniti con successo al fine di costruire filtri capaci di identificare in maniera efficiente i contenuti interessanti per un utente. Per questo motivo questa tesi presenta una rassegna delle principali tecniche adottate nel campo dei sistemi di raccomandazione e descrive tecnologie semantiche promosse dal W3C per poi passare alla definizione di un'architettura software che supporti le tecnologie semantiche descritte e che permetta all'utente finale di utilizzare e combinare insieme algoritmi usati nei sistemi di raccomandazione.

# Chapter 1

## Introduction

One of the major questions posed by information society is how to handle the immense quantity of data that is generated everyday. Advances in technology that allows to store and easily access huge quantity of information brought to the development of what is nowadays known as *information overload*. This term, often referred also as *information overloading*, designates a psychological phenomenon in which a subject is made unable to identify interesting information because of too many data being available to him. This problem was clearly seen by Jacob Jacoby, Donald Speller and Carol Kohn Berning, who in 1974 conducted an experiment on 192 housewives to demonstrate how more information about brands would lead to poorer decision making [1], but already Diderot, the author of the first Encyclopedia himself, foreseen this problem when he wrote:

*As long as the centuries continue to unfold, the number of books will grow continually, and one can predict that a time will come when it will be almost as difficult to learn anything from books as from the direct study of the whole universe. It will be almost as convenient to search for some bit of truth concealed in nature as it will be to find it hidden away in an immense multitude of bound volumes.[61]*

Since the invention of writing mankind has faced the problem of organizing information in a way that could make easy to find a specific piece of information thought to be relevant in a certain moment, but the rise of Internet and especially Social networks and other kinds of sites and services that promote the creation of user generated content brought the problem to a new level.

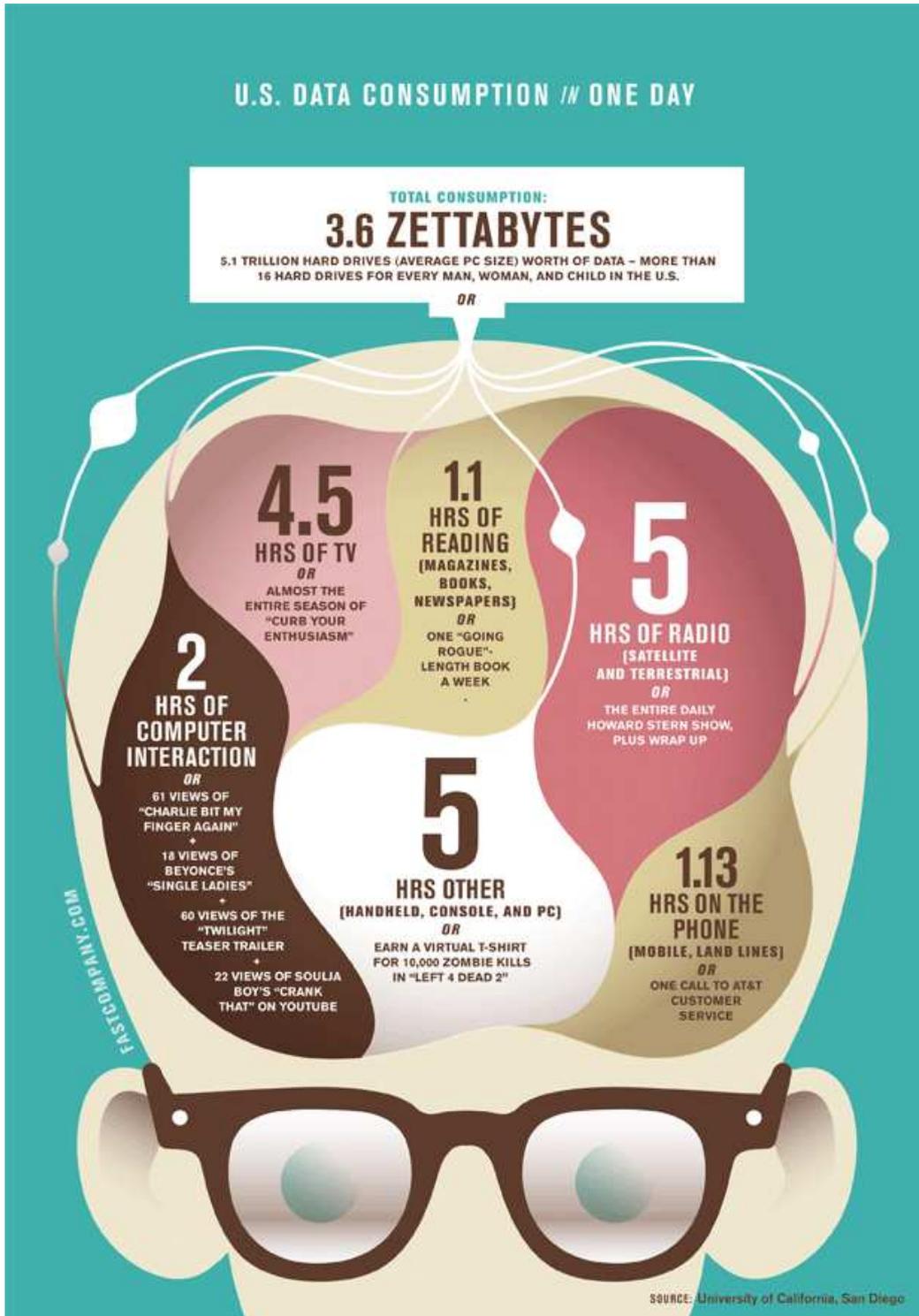


Figure 1.1: Daily data consumption of a US citizen according to UCSD study [2]

During its evolution the Web has grown exponentially in size. Google CEO Eric Schmidt claimed that every two days we create as much information as we did from the dawn of civilization up until 2003, and that the real issue is User Generated Content [3]: beyond this bold statements, even if it is quite hard to determine exactly how much data is actually published on the net and what rate does it grows, some rough estimations of how many web-pages are on the net can be done. First Google index, in 1998, was made up of 26 million page entries and just two years later reached 1 billion pages; by 2008 it counted 1 trillion unique URL [4]. Anyway different URL can map to the same page through fragments and query strings: to get the number of web-pages from the number of URL technology magazine Tech crunch (<http://techcrunch.com>) suggested a conversion ratio of 25:1, leading to a total 40 billion pages[5]. At the same time Google released this data, there were 170 million total registered domains[6] so the average number of pages per domain can be computed as  $40 \cdot 10^9 / 170 \cdot 10^6 = 235$  pages per domain. Multiplying by actual number of registered top level domain names, reported to be of 209.8 billion[7], total number of pages results to be 49.3 billion. Another interesting study that can cast a light on how the Web is growing was conducted by eMarketer and predicts the grow of Internet users who generate content. Their results, summarized in Table 1.1, count everyone who generated content – at least monthly – by accounting for the overlap within the respective categories measured.[8]

	2008	2009	2010	2011	2012	2013
User-generated video	15.4	18.1	20.6	22.7	24.9	27.2
Social networking	71.3	79.7	87.7	94.7	100.1	105.3
Blogs	21.2	23.9	26.7	28.5	30.2	32.1
Virtual worlds	11.6	13.9	15.4	16.9	18.4	19.9
<b>Total</b>	82.5	88.8	95.3	101.7	108.0	114.5

Table 1.1: US User-generated Content creators grouped by content type (millions)

At the same time data on the Net is growing more and more new trends emerge in how those data are indexed and ranked (see A): link mining, folksonomies, and more recently semantic technologies are imposing as necessary instruments to tame the information overload. One of the most active fields of research in this area is the one dealing with “Web 3.0” a term which refers to

the use of semantic technologies embedded in web-sites in order to make the semantics of published data understandable by automated procedures such that, in example, a search for Java (the programming language) does not report a page on Java (the island) as a pertinent match.

This thesis proposes a framework to facilitate implementation, deployment and use of recommender system algorithms in order to tackle the problem of filtering interesting contents. This is achieved through a plugin-based infrastructure, Lastnews, that allows the final user to take control of recommendation algorithms and blend them according to her needs. The work makes use of recent semantic technologies promoted by W3C such as OWL and SPARQL, which will be explained in detail in Chapter 2. The first is a language which allows to define ontologies, which are formal descriptions of set of concepts within a domain and the relationship between them; the second is a query language that operates on RDF graphs, a widely used W3C standard to describe information. After explaining these technologies a short survey about recommender systems, how they are classified and some metrics to evaluate them are presented in Chapter 3. Next two chapters discuss the general architecture and features of Lastnews framework (4) and details of the vocabulary and ontology describing the domain of the application 5. Chapter 6 then describes implementation details of the framework demo and finally in Chapter 7 conclusions about the work are drawn.

# Chapter 2

## Semantic web technologies

In order to give a semantic to data on the Web a stack of recommended technologies is currently being defined by the W3C. This has to be highly portable, flexible, trustworthy, have formal semantics and meet computational and expressiveness needs in order to be effective. Different levels of the stack provide these requirements: lower level deals with portability, middle with semantics, higher with trust about retrieved information. The latter layer is still to develop, while lower and middle implementation is quite stable. Next a short description of all of them is given, with focus on the middle layer.

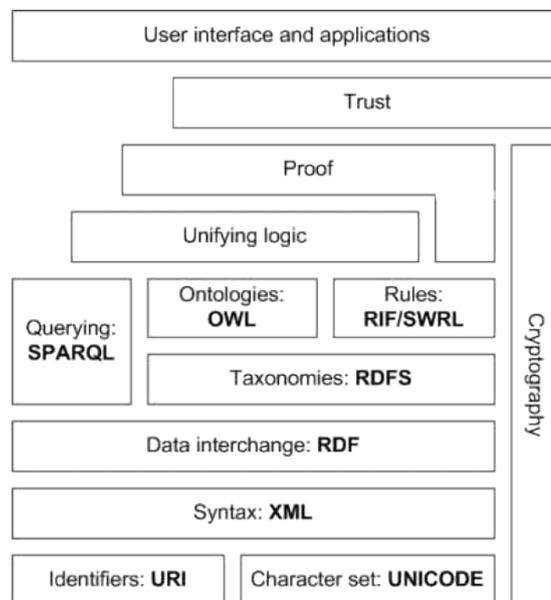


Figure 2.1: Semantic Web stack as described by the W3C

## 2.1 Lower technologies of Semantic Web Stack

**Unicode character set** , a character encoding that is suitable for multilingual environment supporting almost every writing system on Earth.

**URI** Universal Resource Identifier define a syntax in order to provide a way to uniquely name objects. A valid URI is a string of the form `<scheme name>` : `<hierarchical part>` [ `? <query>` ] [ `# <fragment>` ]

**XML** , which stands for eXtensible Markup Language, is a meta-language useful to describe tree structures with user defined vocabulary and syntactic rules. Elements of XML vocabulary are identified by URI, often shortened in the form *ns:localName* where ns represent a short form for the URI part common to all words belonging to a defined dictionary (*namespace*). Namespaces are specified in XML documents too.

**RDF** , which stands for Resource Description Framework, is a formalism to describe data in terms of triples of the form subject, relation, object. Such a triple represents the smallest representable unit of information and is referred as a “statement”. A valid RDF document is a set of statements; these, since every object of a relation can be also a subject of some other, constitute a graph. Subjects and properties represented in RDF are uniquely identified by a URI: objects can both point to a URI or be directly represented by typed strings. RDF triples are usually serialized using XML syntax, but also other syntaxes, that better represent the triples structure of RDF, are possible. Everything in an RDF document belong to the top class *rdf:Resource*, while more specific classes can be assigned via the *rdf:type* relation.

**RDFS** , which stands for RDF-Schema, provides constructs to declare taxonomies of classes and properties with specified domain and range plus some properties such as *rdfs:comment* used to annotate the document. Everything defined in RDFS can be used in RDF document (in practice RDF and RDFS vocabularies are used very often within the same document).

## 2.2 OWL

Until now what has been described is a reticular data-model with support for taxonomies and typed elements. This already provides some basic semantics to data, but this is still too little for the needs of the semantic Web. In order to support more complex inferences and precise definition of terms, a language backed by a formalism whose properties are well known is needed. This language is OWL, a short for Ontology Web Language and the formalism chosen is that of Description Logics. These are a family of logics with at least two remarkable features:

**Decidability** given a set of axioms expressed within a DL formalism it is possible to build a model that satisfies them in finite time.

**High modularity** among the family of DLs it is possible to choose more or less expressive set of operators according to need of expressiveness and performances.

Two more features are not intrinsic to DLs but are relative to how reasoning is performed on sets of DL axioms in the context of semantic Web applications:

**Open World assumption** the reasoner assumes the existence of elements of the domain of the interpretation which are not known. This practically means that facts that can not be demonstrated true are not automatically considered false, which is what happens when assuming Closed World Assumption (an assumption that holds, in example, for ordinary databases).

**Non unique name assumption** the reasoner assumes that the same element of the domain of the interpretation can be referred to with different names. This practically leads to the possibility to infer that two elements of the domain are the same.

### 2.2.1 Syntax and semantics of description logic

**Definition 1** *Description logic are a formalism described by the triple  $\langle \rho, \varphi, \xi \rangle$  where*

*$\rho$  is an alphabet of symbols which denote binary relations.*

*$\varphi$  is an alphabet of symbols which denote atomic concepts.*

$\xi$  is an alphabet of symbols which denote concept constructors (operators), useful to define the semantics of concepts and relations

**Definition 2** An interpretation  $\mathcal{I}$  is a function  $\Sigma \rightarrow \wp(\Delta)$  where  $\Sigma$  is an alphabet of symbols used to describe a model  $\mathfrak{M}$  and  $\Delta$  the universe of objects represented by those symbols.

To have more compact notation if  $C$  is a symbol in  $\Sigma$ ,  $\mathcal{I}(C)$  would be written as  $C^{\mathcal{I}}$ . Every description logic is identified by an acronym which denotes the concept constructors that are admitted in that DL. Below is given an account of such constructors.

**Concepts** : a concept denoted by an element  $C_i$  of the alphabet  $\varphi$  represents a subset of  $\Delta$ . ( $C_i^{\mathcal{I}} \subseteq \Delta$ ). Concepts that are not expressed as a function of other concepts through constructors are called atomic concepts.

**Roles** : a relation denoted by an element  $R_i$  of alphabet  $\rho$  represents a subset of  $\Delta \times \Delta$  ( $R_i^{\mathcal{I}} \subseteq \Delta \times \Delta$ ). Roles can be referred also with the more generic term “property”.

**Subsumption among concepts** : is denoted by expressions of the form  $C_1 \sqsubseteq C_2$  ( $C_2$  subsumes  $C_1$ ), that is interpreted as  $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ . Equivalence can be expressed as  $C_1 \sqsubseteq C_2$  together with  $C_2 \sqsubseteq C_1$

**Subsumption among roles**: is equivalent to subsumption among concepts. DLs which support it present a  $\mathcal{H}$  in the acronym which identifies them. In order to avoid confusion subsumption among roles can be written with a dot  $\dot{\sqsubseteq}$ .  $(R_1 \dot{\sqsubseteq} R_2)^{\mathcal{I}} \equiv R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$

**Top** : denoted by symbol  $\top$  is a concept satisfied by every element in  $\Delta$  :  $\top^{\mathcal{I}} \equiv \Delta$

**Bottom** : is a concept which is not satisfied by any element in  $\Delta$ . it is denoted by symbol  $\perp$  and  $\perp^{\mathcal{I}} \equiv \emptyset$

**Negation** : denoted by symbol  $\neg$  is interpreted as the complementary of a concept:  $(\neg C)^{\mathcal{I}} \equiv \Delta \setminus C^{\mathcal{I}}$ . Use of this operator can be limited to atomic concepts or not restricted. In the latter case logics that support it are denoted by a  $\mathcal{C}$ .

**Intersection of concepts** : denoted by formula  $C_1 \sqcap C_2$  together with  $\perp$  allows to express concept disjunction, as in formula  $C_1 \sqcap C_2 \sqsubseteq \perp$ . Is interpreted as  $(C_1 \sqcap C_2)^{\mathcal{I}} \equiv C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$ .

**Union of concepts** : denoted by formula  $C_1 \sqcup C_2$  and interpreted as  $(C_1 \sqcup C_2)^{\mathcal{I}} \equiv C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$ . Languages that support it present a  $\mathcal{U}$  in their acronym.

**Nominals** : are concepts represented by a singleton. Require one more alphabet  $\omega$  of symbols  $o_i$  for which is valid  $o_i^{\mathcal{I}} \equiv K \subset \Delta \wedge |K| = 1$ . The possibility to use nominals is denoted by a  $\mathcal{O}$ . Together with operator  $\sqcup$ , allows to define finite elements sets.

**Universal quantification** : allows to specify functions range, and is denoted by formula  $\forall R.C$  where  $R$  is a relation and  $C$  a concept. Is interpreted as  $(\forall R.C)^{\mathcal{I}} \equiv \{x : x \in \Delta \wedge (x, y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}$

**Existential quantification** : allows to assert the existence of a relation between two elements. Can be restricted, as in formula  $\exists R.\top$  or not: in the latter case is possible to use an arbitrary concept instead of  $\top$ . DLs that support this feature present a  $\mathcal{E}$  in their acronym. Semantic is  $(\exists R.C)^{\mathcal{I}} \equiv \exists y : (x, y) \in R \wedge y \in C^{\mathcal{I}}$ .

**Transitivity** : allows to define properties such that  $\{(x, y), (y, z)\} \subset R^{\mathcal{I}} \Rightarrow (x, z) \in R^{\mathcal{I}}$ . Together with other operators is denoted by letter  $\mathcal{S}$

**Inverse roles** : are roles  $R^-$  such that  $(x, y) \in R^{\mathcal{I}} \Leftrightarrow (y, x) \in R^{\mathcal{I}}$ . Possibility to define them is denoted by letter  $\mathcal{I}$ . Together with role subsumption allows to define symmetric roles.

**Functional roles** are roles so that for each element of the domain exist at maximum 1 corresponding element in the range  $(x, y), (x, z) \in R \Rightarrow y = z$ . It is denoted by a  $\mathcal{F}$  in DL name.

**Cardinality restrictions** : denoted by expression such as  $\geq nR.\top$  or  $\leq nR.\top$  where  $n$  represent a number are an extension of the previous feature, by making imposed cardinality a variable. Formally (for the first formula)  $(nR.\top)^{\mathcal{I}} \equiv |\{(x, y) : x \in \Delta \wedge (x, y) \in R\}| \leq n$  DLs that admit them have a  $\mathcal{N}$  in their names. These restrictions can not be imposed to complex roles without losing decidability.

**Definition 3** A role is said to be complex if it contains a chain of properties, so if it is transitive, inverse of some other property or superproperty of a complex property 2.2.1

**Qualified cardinality restrictions** : are an extension of the previous allowing to specify cardinality for each concept subsumed by domain of a relation. This feature is denoted by letter  $\mathcal{Q}$ . Interpretation is  $(nR.C)^{\mathcal{I}} \equiv |\{(x, y) : x = k \wedge y \in C \wedge (x, y) \in R\}| \leq (\geq)n$

**Concrete domains** extend the language with datatypes such as string, date, etc. DLs that support them present a  $(D)$  in their names.

**Self related Concept** is the concept  $\exists R.Self$  that denotes the set of all elements that are in relation  $R$  with themselves  $((\exists R.Self)^{\mathcal{I}} \equiv x : (x, x) \in R)$

**Reflexivity** : allows to define properties such that  $\forall x \in \Delta (x, x) \in R^{\mathcal{I}}$ . This property is denoted by the formula  $Ref(R)$

**Irreflexivity** : allows to define roles such that  $\nexists x : (x, x) \in R^{\mathcal{I}}$ . It is denoted by formula  $Irr(R)$

**Asymmetric roles** : roles such that  $(x, y) \in R^{\mathcal{I}} \Rightarrow \neg(y, x) \in R^{\mathcal{I}}$ . They are declared with formula  $Asymm(R)$

**Antisymmetric roles** : roles such that  $(x, y), (y, x) \in R^{\mathcal{I}} \Rightarrow y = x$ . They are declared with formula  $Antisymm(R)$

**Complex role inclusion** allows to write subsumption between roles of the form  $S_1 \circ S_2 \circ \dots \circ S_n \circ R \sqsubseteq R$  or  $R \circ S_1 \circ S_2 \dots S_n \sqsubseteq R$  or  $S_1 \circ S_2 \circ \dots \circ S_n \sqsubseteq R$ . Where the operator  $\circ$  is associative and  $(S_1 \circ S_2)^{\mathcal{I}} \equiv \{(x, z) \in \Delta \times \Delta : \exists(x, y) \in S_1^{\mathcal{I}} \wedge \exists(y, z) \in S_2^{\mathcal{I}}\}$ . This kind of formula breaks decidability if  $R \sqsubseteq S_k$  is entailed by the model for some  $k : 1 \leq k \leq n$ . Expressions like  $S_1 \circ \dots \circ S_n$  are often referred as “chain of properties”.

**Top and bottom relation** , denoted by symbol  $\dot{\top}$  and  $\dot{\perp}$  are the universal and empty relation  $\dot{\top} \equiv \Delta \times \Delta, \dot{\perp}^{\mathcal{I}} = \emptyset$

**Concept and role assertion** : expressed by  $C(a)$  and  $R(a, b)$  where  $a$  and  $b$  are nominals denotes  $a^{\mathcal{I}} \in C$  and  $b(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$

### 2.2.2 Relations between OWL, RDF and description logic

OWL is a syntactic variant of DL formalism whose first version was released in 2004 by the W3C Web ontology working group. First version of OWL was based on description logic belonging to  $\mathcal{SH}$  family and since then recommended the use of different sets of features in order to better fit expressiveness or efficiency constraints depending on context. OWL-DL was the full featured decidable variant of the language, supporting  $\mathcal{SHOIN}$  logic, while OWL-Lite was a more computationally-efficient subset supporting  $\mathcal{SHLF}$ . Using OWL without taking care of non structural constraints needed to ensure decidability e.g. declaring cardinality restriction on transitive properties brings to yet another variant of the language, known as OWL-Full, which is not decidable. It is important to note that differences between OWL-DL and OWL-Full are not on the constructs, but on admissible ways of using them. Free mixing of RDF/RDFS and OWL is admitted too in OWL-full [9]. Since OWL is a vocabulary extension of RDF any RDF graph forms an OWL-Full document: OWL just assigns additional meaning to certain RDF triples.

Version 1.1 of OWL, providing support for  $\mathcal{SROIQ}$  logic was released on December 2006 [10], while latest W3C recommendation about the language is OWL2. This new major release of the language is back-compatible with the previous, introduces new features and specifies new subsets of the language.[11] These are:

**OWL2 EL** , designed for very large set of properties and classes, support  $\mathcal{EL}^{++}$  logic.[12]

**OWL2 QL** , designed to answer conjunctive queries in logspace time, is based on DL-Lite[13]

**OWL2 RL** designed to be easily implemented using a Rule-based approach to reasoning and offering good scalability. It's based on DLP logic [14]

## 2.3 SPARQL

SPARQL is a query language specifically designed to query RDF graphs and takes advantage of triples pattern in order to express queries. A triple pattern is an RDF triple which may contain variables. Combining multiple triple patterns it is possible to build graph patterns: in fact two variables with the same name are

Main DLs			
DL acronym	Allowed operators	Computational complexity (satisfiability)	Notes/OWL dialect
$\mathcal{EL}$	$\top \mid \sqcap \mid \exists R.C$	polynomial time	
$\mathcal{EL}^{++}$	$\sqsubseteq \mid \top \mid \sqcap \mid \exists R.C \mid \perp \mid oneOf() \mid S_1 \circ S_2 \circ \dots \circ S_n \circ R \sqsubseteq R \mid range(R) \sqsubseteq C \mid domain(R) \sqsubseteq C$	polynomial time	Supported by OWL2 EL, must satisfy constraint $R_1 \circ \dots \circ R_n \sqsubseteq S$ with $n \geq 1$ and $range(S) \sqsubseteq C \implies range(R_n) \sqsubseteq C$
$\mathcal{ALC}$	$\sqsubseteq \mid \top \mid \sqcap \mid \exists R.C \mid \forall R.C \mid \neg C$	PSpace-complete	$\equiv \mathcal{ALUE}$
$\mathcal{S}$	$\sqsubseteq \mid \top \mid \sqcap \mid \exists R.C \mid \forall R.C \mid \neg C \mid tr()$	PSpace-complete	equivalent to modal logic S4
$\mathcal{ALCHIF}$	$\mathcal{ALC} + \dot{\sqsubseteq} \mid R^- \mid func()$	PSpace-complete hard	
$\mathcal{SHIF}$	$\mathcal{ALCHIF} + tr()$	ExpTime-complete	OWL-Lite
$\mathcal{SHIN}$	$\mathcal{SHIF} + \exists nR$	ExpTime-complete	
$\mathcal{SHOIN}$	$\mathcal{SHIN} + oneOf()$	NExpTime-complete	OWL-DI
$\mathcal{SROIQ}$	$\mathcal{SHOIN} + nR.C \mid \exists R.Self \mid Ref() \mid \dot{\top} \mid \dot{\perp} \mid S_1 \circ S_2 \circ \dots \circ S_n \circ R \sqsubseteq R \mid Asymm() \mid Irr()$	NExp-time hard	OWL 1.1

Table 2.1: Main DLs that can be represented by OWL

bound to the same element while processing a query. For example with Listing 2.1 as dataset, Listing 2.2 returns the variable/value pair `?blog = <http://daveideynard.it/blog>`

```

<!-- Dataset http://www.example.com/myFoaf -->
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  <foaf:PersonalProfileDocument rdf:about="http://www.
    example.com/myFoaf">
    <foaf:maker rdf:resource="#me"/>
    <foaf:primaryTopic rdf:resource="#me"/>
  </foaf:PersonalProfileDocument>
  <foaf:Person rdf:ID="me">
    <foaf:name>Andrea Mirone</foaf:name>
    <foaf:nick>ilmirons</foaf:nick>
    <foaf:knows>
      <foaf:Person rdf:ID="de">
        <rdfs:seeAlso>
          <rss:channel rdf:about="http://del.icio.us/rss/
            aittalam">
            <dc:title>del.icio.us/aittalam</dc:title>
            <dc:description>del.icio.us bookmarks as an RSS
              1.0 news feed</dc:description>
            <foaf:maker rdf:resource="#de"/>
          </rss:channel>
        </rdfs:seeAlso>
        <foaf:name>Davide Eynard</foaf:name>
        <foaf:weblog rdf:resource="http://daveideynard.it/
          blog"/>
      </foaf:Person>
    </foaf:knows>
  </foaf:Person>
</rdf:RDF>

```

Listing 2.1: A short ontology using FOAF and DC vocabularies

```

PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
2 SELECT ?blog
WHERE { ?x foaf:name "Andrea_Mirone" .
        ?x foaf:knows ?y .
        ?y foaf:weblog ?blog .
      }

```

Listing 2.2: Ask for weblogs of people known by Andrea Mirone

SPARQL syntax is not based on XML, but on another notation for RDF, named *turtle* (wordplay to denote Terse RDF Triple Language)[15]. Listing 2.2 shows a *select-query*: this is not the only kind of query allowed by SPARQL. Other kinds are possible, but they all share the same general query structure. Each query initially defines the namespaces it uses through the keyword “PREFIX”; then it declares the query type, in this case with keyword, “SELECT”, that tells the query engine to perform a projection over the following variables, denoted by a prefixed question mark; finally the “WHERE” clause declares the graph pattern to match. This is a list of triples dot separated and enclosed in braces. Differently from RDBMS there are no null-valued properties, but a property can be totally absent for a resource. In example in Listing 2.1 both “me” and “de” are of type *foaf:Person*, but have different properties: in particular “de” has a weblog, while “me” has not. SPARQL has an explicit construct to deal with this kind of situation: query in Listing 2.3 binds all names of resources of type<sup>1</sup> *foaf:Person* to “?name” variable, while their blogs are bound to “?blog” variable just if present. In other words query does not fail for those elements of Person which do not have a blog.

```

PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?name , ?blog
WHERE { ?x a foaf:Person .
4     OPTIONAL {
        ?x foaf:weblog ?blog .
      }
      }

```

Listing 2.3: Ask for name and weblogs of people

<sup>1</sup>‘a’ is a short for *rdf:type* according to turtle/SPARQL syntax

Another interesting feature of SPARQL is its ability to specify the dataset to be used to perform the query. The “FROM” and “FROM NAMED” clauses are used for this purpose: the first specifies an anonymous graph, while the other provides a set of IRIs<sup>2</sup>/graph bindings. More than one “FROM” and “FROM NAMED” clauses are allowed in the same query. It is also possible to know from which graph data were extracted adding a “GRAPH” clause to the query. This is shown in Listing 2.5, which queries both Listing 2.1 and Listing 2.4

```

3 <!-- Dataset http://www.example.com/libbyFoaf -->
  <?xml version="1.0"?>
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-
    ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xmlns:rss="http://purl.org/rss/1.0/"
        xmlns:foaf="http://xmlns.com/foaf/0.1/"
8        xmlns="http://xmlns.com/foaf/0.1/"
        xmlns:dc="http://purl.org/dc/elements/1.1/"
    >
    <foaf:PersonalProfileDocument rdf:about="http://www.
      example.com/libbyFoaf" />
    <foaf:Person rdf:ID="libby">
13  <foaf:name>Libby Miller</foaf:name>
    <foaf:nick>libby</foaf:nick>
    <foaf:depiction rdf:resource="http://swordfish.rdfweb.
      org/people/libby/libby.jpg" />
    <foaf:homepage rdf:resource="http://nicecupoftea.org"/>
    <foaf:weblog rdf:resource="http://planb.nicecupoftea.org
      /"/>
18  </foaf:Person>
  </rdf:RDF>

```

Listing 2.4: Another FOAF dataset

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
```

<sup>2</sup>IRIs are a generalization of URIs, allowing the use of all characters from Universal Character Set (UCS) instead that being limited to a subset of ASCII. IRI stands for Internazionalized Resource Identifier

```

SELECT ?g, ?name
FROM NAMED <http://www.example.com/libbyFoaf>
FROM NAMED <http://www.example.com/myFoaf>
5 WHERE {
    GRAPH ?g {
        ?p a foaf:Person .
        ?p foaf:name ?name .
    }
10 }

```

Listing 2.5: A SPARQL query that retrieve people names keeping track of datasets they come from

Use of named graph in a single document is not provided by the original RDF recommendation [16], but can be achieved using TriG, an extensions of turtle grammar that allows to group triples in named block. Turtle/TriG has been widely used through this work. In Listing 2.6 the TriG serialization of data used in the examples is shown.

```

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
@prefix rss: <http://purl.org/rss/1.0/>
@prefix foaf: <http://xmlns.com/foaf/0.1/>
5 @prefix dc: <http://purl.org/dc/elements/1.1/>
@prefix mf: <http://www.example.com/myFoaf/>
@prefix lf: <http://www.example.com/libbyFoaf/>

<http://www.example.com/myFoaf> a foaf:
10 PersonalProfileDocument ;
                                foaf:maker ns:me ;
                                foaf:primaryTopic ns:me ;

<http://www.example.com/libbyFoaf> a foaf:
PersonalProfileDocument ;

15 <http://www.example.com/myFoaf> {
    mf:me a foaf:Person ;
        foaf:name "Andrea Mirone" ;

```

```
20     foaf:nick "ilmirons" ;
       foaf:knows mf:de ;

mf:de a foaf:Person ;
      rdfs:seeAlso <http://del.icio.us/rss/aittalam> ;
      foaf:name "Davide Eynard" ;
25     foaf:weblog <http://davide.eynard.it/blog> ;

<http://del.icio.us/rss/aittalam> a rss:channel ;
                                  dc:title "del.icio.us/
                                           aittalam" ;
                                  dc:description "del.icio
                                           .us bookmarks as an
                                           RSS 1.0 news feed" ;
                                  foaf:maker mf:de ;
30
}

<http://www.example.com/libbyFoaf> {
35
  lf:libby a foaf:Person ;
           foaf:name "Libby Miller" ;
           foaf:nick "libby" ;
           foaf:weblog <http://planb.nicecupoftea.org> ;
           foaf:homepage <http://nicecupoftea.org> ;
           foaf:depiction <http://swordfish.rdfweb.org/
                           people/libby/libby.jpg> ;
40
}
}
```

Listing 2.6: TriG serialization of data used in the examples

Beyond select queries, which get lists of variables/value binding, SPARQL supports three more kinds of interrogation. The simplest is the *ask-query*, which verifies if a graph pattern is present in a dataset and return a boolean; another, known as *describe-query* return an RDF description of resources which match the graph pattern: this is particularly useful as there is no guarantee of which properties are present for a certain resource; last one is *construct-query* which

allows to build a new graph from data extracted. Furthermore SPARQL inherits also some features from traditional SQL: among these there are tests, through the “FILTER” clause, which permits number comparison and regular expression matching, “DISTINCT” clause, allowing to get single results when data is repeated, “ORDER BY” etc. Details on these features are omitted as they are not relevant in the scope of this work, but full documentation can be found in [17].

# Chapter 3

## Recommender System

Recommender Systems have quite a long history of study at their back: first papers on the subject began to appear in the mid-1990s [18] [19], but interest in this area remains high because of the multitude of practical applications of this kind of systems, that ranges from commerce to Internet searching. A wide definition of what a recommender system is is reported below:

**Definition 4** *Given a set of users  $U$ , a set of items  $I$  a recommender system is a system  $\mathfrak{S}$  built upon some input  $p$  that allows to calculate a good approximation  $\rho'$  of rating  $\rho : U \times I \rightarrow R \subseteq \mathbb{R}$  which represents how much  $u$  likes  $i$ , in order to select items of interest for  $u$ <sup>1</sup>*

According to what  $R$  is the problem can be viewed as a classification problem, if  $R$  is discrete, or a regression problem if  $R$  is continuous. Some works even do not take into account the values of  $\rho$  but just focus on the ordering imposed to items by the rating function: this is referred as *preference based filtering* [21] The input  $p$  is in general a subset of the function  $\rho$  or other data useful to calculate  $\rho'$ . It may include one or more of the following:

- *Rating matrix*  $\varrho$ : is a subset of  $\rho$  explicitly given by the user, where each element  $\varrho_{ij}$  represents the real-valued rating given by user  $i$  to item  $j$
- *Transaction matrix*  $\tau_i$ : is a binary matrix defined for each item  $i$  where rows are associated to users and columns to specific interactions performed by user on the item. Sometimes a specific weight is associated to each interaction.

---

<sup>1</sup>After evaluating  $\rho' \approx \rho$  the typical approach for selection is recommending for all  $u$  the set  $S = \{i_1 \dots i_n\} = \arg \max_{\iota \in \wp(I): |\iota|=n} \sum_{i \in \iota} \rho'(u, i)$  [20]

- *Item feature matrix*  $\varphi$ : is a matrix where each line is associated with an item and each column to a feature.  $f_{ij}$  is the value of feature  $j$  for item  $i$ .
- *User profiles matrix*  $\pi$ : is a description of the users where each user is represented by a row and each feature by a column. It involves personal data such as gender, age, interests, etc. that has to be provided explicitly by the user and is used to perform so called *demographic filtering*.

Many taxonomies for recommender systems have been proposed: in example Montaner et al. present a 8-dimensional classification[22], but most of the literature on the topic agrees to divide recommender systems in two categories according to technique used for  $\rho$  approximation and three categories according to how items to be recommended are chosen. According to approximation technique recommendation methods can be divided in

- Model based
- Memory based

While according to how items are selected it is possible to distinguish:

**content-based recommender systems** where the items recommended are chosen according to similarity of content w.r.t. content already appreciated.

**collaborative recommender systems** if the user is recommended items that people similar to her liked in the past.

**hybrid recommender system** if they combine both approaches.

In the next sections each of these categories will be presented with a short description of the main methodologies used to implement it.

### 3.1 Memory based methods

Memory based methods are also referred as “heuristic based” for the fact they apply some fixed function to data in order to approximate  $\rho$ , as opposed to estimation methods where instead values for  $\rho'$  are calculated with no fixed formula but following some minimization/maximization criterion. In the specific case of memory based methods aggregate functions are used to summarize (sub)sets of known ratings  $\{q_{u,i}\}$ .

**Definition 5** *An aggregate function is a function where multiple values are grouped together to form a single value which is representative for the entire set of input.*

Functions commonly used in the field of recommender systems are weighted average, where the weight of each term is proportional to similarity between items or users  $e_j, e$  being compared or mode among  $k$  most similar elements when considering discrete ratings.

$$\rho'(u, i) = \frac{1}{|\{e_j : e_j \sim e\}|} \sum_{e_j: e_j \sim e} \text{sim}(e_j, e) \varrho(e_j, e) \quad (3.1)$$

For comparing objects from a set, what is needed, apart from the similarity measure  $\text{sim}$ , is a representation of the object which summarizes its features. A widely used representation is that of a feature vector, that is a row or a column of previously defined 3 item and profile matrix or, more in general, a vector whose elements represent values for some features of the object the vector represents. This leads directly to an approach of modeling similarity known as *vector space model*, which defines (dis)similarity as distance between points in a multi-dimensional space whose dimensions are the features of interest. The trouble with this technique is that for non trivial dataset number of dimensions and elements tend to be too large to ensure good performances. This is particularly true when what is to be modeled is text, as in items. To overcome this problem dimensionality reduction techniques such as Locality Preserving Index are used. Locality Preserving Indexing is an algebraic method able to construct a similarity matrix  $S \in \mathbb{R}^{n \times n}$  starting from a set of elements  $x_1, x_2 \dots x_n \in \mathbb{R}^m$ . The principle of this technique is to map elements that are near in the original space to elements that are near in the reduced space, and do so minimizing the global reconstruction error. Transformation vectors to perform LPI can be obtained by solving the minimization problem 3.2

$$a_{opt} = \arg \min_a \sum_{i,j} (a^T x_i - a^T x_j)^2 S_{ij} = \arg \min_a a^T X L X^T a$$

with constraint

$$a^T X D X^T a = 1 \quad (3.2)$$

Where  $L = D - S$  and  $D_{ii} = \sum_j S_{ij}$  represents a measure of local density around  $x_i$ . Similarity matrix is constructed as:

$$S_{ij} = \begin{cases} \frac{x_i^T x_j}{\|x_i^T x_j\|} & \text{if } x_i \text{ is among the } p \text{ nearest neighbors of } x_j \\ & \text{or } x_j \text{ is among the } p \text{ nearest neighbors of } x_i \\ 0 & \text{otherwise} \end{cases} \quad (3.3)$$

Thus, the objective function in 3.2 gets an heavy penalty if sets points that are near far apart in the new space.

## 3.2 Model based methods

Model based methods act differently from memory based because they do not apply a formula on saved instances, but try to build a model which summarizes information about what the user likes. Model based methods tend to be less memory greedy than heuristic based and often also show better performances, but there is no theoretical evidence of their superiority over memory based. Bayesian networks are one of the most popular model based methods.

**Definition 6** *A Bayesian network is a couple  $\langle G, \Pi \rangle$  where  $G$  is a direct acyclic graph with nodes representing variables and edges representing conditional dependency relationships while  $\Pi$  is the set of conditional probability of the form  $P(x = k | x_1 = k_1 \wedge x_2 = k_2 \wedge \dots \wedge x_n = k_n)$  where  $x_1 \dots x_n$  are direct ancestors of  $x$ . A Bayesian network where all nodes are connected just with the node representing the output and considers just the most likely prediction among a set of discrete possible output is a Naïve Bayesian classifier.*

From a statistical point of view Bayesian classifiers are Bayesian networks that rely on the assumption that output depends on all other variables of the model, but knowing about the value of a non-output variable does not affect the other non-output variables probability. This assumption is generally false but despite this Bayesian classifiers perform very efficiently also when compared with more complex Bayesian networks. These models exploit Bayes Theorem 3.4 which states that probability of an event  $E_1$  given another event  $E_2$  is given by the probability of  $E_2$  given  $E_1$  scaled by the ratio between probability of  $E_2$  and  $E_1$ .

In formula:

$$P(E_1|E_2) = \frac{P(E_2|E_1) P(E_1)}{P(E_2)} \quad (3.4)$$

Following Bayes terminology  $P(E_1|E_2)$  is called *posterior probability*  $P(E_2|E_1)$  is the *likelihood*  $P(E_1)$  the *prior* and  $P(E_2)$  the *marginal likelihood*. So plugging the feature vector  $\vec{V}$  of an item as the event observed ( $E_2$ ) and a binary variable for liked/disliked as event to predict it is possible to calculate  $P(\text{liked}|\vec{V})$  estimating probabilities from the dataset. In fact it holds:

$$P(\text{liked}) = \frac{|\{i : \text{liked}(i)\}|}{|I|} \quad (3.5)$$

$$P(\vec{V}) = \frac{|\{v : v = \vec{V}\}|}{|I|} \quad (3.6)$$

$$P(\vec{V}|\text{liked}) = \prod_i P(v_i|\text{liked}) \quad (3.7)$$

Where  $|\{i : \text{liked}(i)\}|$  represents the number of items liked by the user, and  $v_i$  the value of the  $i$ th element of the feature vector. Equation 3.7 holds because of the assumption of independence. Other model methods include artificial neural networks and clustering.

Artificial neural networks are general function approximators inspired by the way neurons work in nature [23]: they are net of nodes (neurons) whose output is activated when weighted sum of inputs is greater than a threshold:

$$O = g \left[ \left( \sum_{i=1}^n w_i \cdot x_i \right) - w_0 \right] = g \left( \sum_{i=0}^n w_i \cdot x_i \right) \text{ with } x_0 = 1 \quad (3.8)$$

where  $w_i \cdot x_i$  is the  $i$ th input,  $w_0$  represents the threshold and  $g$  the activation function which is chosen in order to ensure some features on the output. Activation functions generally restrict the range of possible output values to the interval  $[0 - 1]$  or  $[-1 - 1]$  (such as step function or sig-function) and may be also required to be continuous (in this case tanh or sigmoid are used<sup>2</sup>).

Multilayer structures of such nodes can model very complex non linear space boundaries to perform prediction or approximate an unknown function.

**Definition 7** *Neural networks with multi-layer topology are networks of neurons connected such that for each path from a neuron to the output the length is the*

<sup>2</sup>  $\text{sigmoid}(x) = \frac{1}{1+e^{-kx}}$   $k > 0$        $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

---

**Algorithm 1** Backpropagation algorithm for a 2-layer neural network

---

**Input:** *training\_examples*: set of  $\langle \vec{x}, \vec{y} \rangle$  I/O pairs,  $\eta$ : learning rate,  $\epsilon_{error}$  an acceptable error,  $N_{out}$ ,  $N_{hidden}$  : sets of output, hidden neurons

**Output:**  $W$  matrix of weight for neurons connections

{The input from unit  $i$  into unit  $j$  is denoted  $x_{ji}$ , and the weight from unit  $i$  to unit  $j$  is denoted  $w_{ji}$ }

```

for all  $W_{ji}$  do
   $W_{ji} \leftarrow random(-\epsilon, \epsilon)$ 
end for
5: repeat
  for all  $\langle \vec{x}, \vec{y} \rangle \in training\_examples$  do
    input  $\vec{x}$  and calculate output  $o_u$  of every unit  $u$  of the network.
    for all  $k \in N_{out}$  do
      {calculate error term for output units}
10:    $\delta_k \leftarrow o_k(1 - o_k)(y_k - o_k)$ 
    end for
    for all  $h \in N_{hidden}$  do
      {calculate error term for hidden units}
       $\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh} \delta_k$ 
15:    end for
    for all  $W_{ji}$  do
       $\Delta W_{ji} \leftarrow \eta \delta_j x_{ji}$ 
       $W_{ji} \leftarrow W_{ji} + \Delta W_{ji}$ 
    end for
20:  end for
  until  $\frac{1}{|N_{out}|} \sum_{k \in N_{out}} (y_k - o_k)^2 < \epsilon_{error}$ 
return  $W$ 

```

---

same.

A layer, is a group of neurons which have the same distance from output (and hence input). We call the layer that gives final output “output layer” and “hidden layers” all other layers between output and input.

Neural networks tune their parameters (weights for perceptrons outputs  $w_j$ ) via supervised learning: given a set of correct input/output pairs they iterate over it updating  $w_j$  values using algorithm based on backpropagation, sketched in 1

Unlike neural networks clustering approaches implement an unsupervised learning strategy to build a model useful for performing item recommendation. The problem of clustering is, given raw data, that to identify sets of similar items (clusters) and/or some kind of description that summarizes them, where “description” is to be intended in a wide meaning. For example one of the most popular algorithms for clustering, K-means 2, uses the centroid of a group of elements of the vector space as description of a cluster. This methodology has been successfully used in works such as [24].

---

**Algorithm 2** K-mean clustering algorithm

---

**Input:**  $k$ : number of clusters to create,  $dataset = \{p_1, p_2 \dots p_n\}$ : set of points in  $N$ -dimensional space  $\mathfrak{S}$  to partition  
**Output:**  $\{c_1, c_2 \dots c_k\}$ : centroids of identified clusters

```

{Initialize partition centroids at random}
for all  $i \in \{1 \dots k\}$  do
     $c_i \leftarrow random(x \in \mathfrak{S})$ 
5:   $S_i \leftarrow \emptyset$ 
end for
repeat
    {Assign each point to set calculating distance from centroid}
    for all  $p \in dataset$  do
10:   $\bar{i} \leftarrow \arg \min_{i \in \{1 \dots k\}} \|p - c_i\|^2$ 
         $S_{\bar{i}} \leftarrow S_{\bar{i}} \cup \{p\}$ 
    end for
    for all  $i \in \{1 \dots k\}$  do
         $oldC_i \leftarrow c_i$ 
15:   $c_i \leftarrow \arg \min_{x \in \mathfrak{S}} \sum_{p_j \in S_i} \|p_j - x\|^2$ 
    end for
until  $\forall i \in \{1 \dots k\} \quad c_i = oldC_i$ 
return  $\{c_1 \dots c_k\}$ 

```

---

### 3.3 Content Based Filtering

Content based filtering takes the moves from well consolidated information retrieval techniques. In this kind of approach the estimation of  $\rho$  has the form

$$\rho'(u, i) = \mathcal{F}(\varrho_{u,\cdot}, [\tau_{i,u,\cdot}])$$

where the vector in square brackets  $\tau_{i,u,\cdot}$  is optional and denotes the possibility to use logging of transactional behavior of the user as input, which represents an extension of classical information retrieval technique.

The principle behind Content based filtering is that a user is likely to prefer items similar to what she already read in the past than others: so similarity measures come into play. The vector space model is often used in this field together with keywords as features of item vectors. What keyword is relevant w.r.t. a particular item is chosen according to an index: most used for this purpose is TF-IDF [25] which summarizes the information of two other indexes, namely Term Frequency and Inverse Document Frequency. The first is defined as zero if the term does not appear in the item and nonzero otherwise: the second is defined as a scaling factor such that if a term appears in just a few documents its scaling factor will be large and vice-versa. Example formulas for such indexes, as defined in the Cornell SMART system[26] are the following:

$$\text{TF}(i, t) = \begin{cases} 0 & \text{if } \text{freq}(i, t) = 0 \\ 1 + \log(1 + \log(\text{freq}(i, t))) & \text{otherwise} \end{cases} \quad (3.9)$$

$$\text{IDF}(t) = \log \frac{1 + |I|}{|I_t|} \quad (3.10)$$

$$\text{TF-IDF}(i, t) = \text{TF}(i, t) \times \text{IDF}(t) \quad (3.11)$$

Where term frequency  $\text{freq}(i, t)$  is the number of occurrences of term  $t$  in item  $i$ ,  $|I|$  is the number of items in the system, and  $|I_t|$  the number of items in which  $t$  is present. TF-IDF substantially rewards those terms which are common just in a few items. After feature vectors  $v_i$  are built they can be compared using an appropriate similarity function, such as the cosine measure 3.12

$$\text{sim}(\vec{v}_1, \vec{v}_2) = \frac{\vec{v}_1 \cdot \vec{v}_2}{|\vec{v}_1| |\vec{v}_2|} \quad (3.12)$$

Where  $\vec{v}_1 \cdot \vec{v}_2$  is the standard vector dot product and the norm is defined as  $|\vec{v}_1| = \sqrt{\vec{v}_1 \cdot \vec{v}_1}$ . Very often also dimensionality reduction techniques are used to ensure better performances as the vocabulary of keyword is always quite large. After items are reduced to features vector many other standard techniques of data mining can be applied: approaches based on clustering, decision trees or artificial neural networks have all been developed.

Cons of content based filtering include the fact they tend to overfit, as they feed themselves always with similar items. This problem can be overcome introducing some randomness in the process. Another limitation is that this technique is difficult to apply on items which are not texts, such as media, and the need of a lot of ratings to perform good predictions (new-user problem).

### 3.4 Collaborative methods

In collaborative methods rating of items w.r.t. to a user is calculated starting from ratings given by users similar to the one asking for recommendation. Formally

$$\rho'(u, i) = \mathcal{F}(\{\varrho_{u_j, i}\}) \text{ where } u_j : u_j \sim u$$

Where  $u_j \sim u$  denotes that for the chosen similarity function  $\text{sim} : U \times U \rightarrow \mathbb{R}$   $\text{sim}(u_j, u) > \text{threshold}$ . Typically  $\mathcal{F}$  is an aggregate function such as weighted standard deviation from user mean 3.13 while for similarity Pearson correlation 3.14 is used.

$$\rho'(u, i) = \overline{\varrho_{u, \cdot}} + k \sum_{u_j : u_j \sim u} \text{sim}(u_j, u) (\varrho_{u_j, i} - \overline{\varrho_{u_j, \cdot}}) \quad (3.13)$$

$$\text{sim}(u_j, u) = \frac{\sum_{i \in I_{u_j, u}} (\varrho_{u, i} - \overline{\varrho_{u, \cdot}}) (\varrho_{u_j, i} - \overline{\varrho_{u_j, \cdot}})}{\sqrt{\sum_{i \in I_{u_j, u}} (\varrho_{u, i} - \overline{\varrho_{u, \cdot}})^2 \sum_{i \in I_{u_j, u}} (\varrho_{u_j, i} - \overline{\varrho_{u_j, \cdot}})^2}} \quad (3.14)$$

Where  $\overline{\varrho_{u_j, \cdot}}$  represents the sample mean of known ratings given by user  $u_j$  to all items and  $k$  is a normalizing factor. Note that 3.13 tends to be better w.r.t. weighted average as it tends to filter differences in the way of interpreting the rating that are common among different users, a normalization issue that is not present in item based recommendation where all items compared are rated by the same user.

Many modification and implementation strategies have been proposed in order

to keep performance high: a very simple and widely used strategy for example is to calculate all user similarities in advance and update them only once in a while. This results to be effective as the network of peers is quite stable over time. Algorithmic rather than implementation approaches include technique such as default rating [27]. This improvement relies on evidence that when there are few votes for either the user who is asking recommendation or the one she matches, correlation 3.14 perform poorly as it takes into consideration just items rated by both. To overcome this problem the idea is to assign some default rating where they are missing and then perform the similarity comparison taking into consideration the union of voted items instead of intersection. This is equivalent of assuming users would agree on items that neither voted. This brings the following modification on Equation 3.14:

$$sim(u, u_j) = \frac{(n+k)(\sum_{i \in I} \rho_{u,i} \rho_{u_j,i} + kd^2) - (\sum_{i \in I} \rho_{u,i} + kd)(\sum_{i \in I} \rho_{u_j,i} + kd)}{\sqrt{((n+k)(\sum_{i \in I} \rho_{u,i}^2 + kd^2) - (\sum_{i \in I} \rho_{u,i} + kd)^2)((n+k)(\sum_{i \in I} \rho_{u_j,i}^2 + kd^2) - (\sum_{i \in I} \rho_{u_j,i} + kd)^2)}} \quad (3.15)$$

with rating  $d$  expressing neutrality or slightly negative impression about the item,  $n = |I_{u,u_j}|$  and  $k$  the number of items rated with default value  $d$ .

Collaborative methods, independently from approach used to implement them suffer the new-user problem discussed also for content based recommendation and add also a new-item problem: in fact since only items rated by similar users are used for recommendation items not rated are seldom recommended. This implies also that outliers in the set of users will receive poor recommendations: in order to avoid this 2 solutions have been proposed: the first is demographic recommendation, that is comparing users not only for their preferences but also according to demographic data; the second is the use of stereotypes [28]. Stereotypes are sets of characteristics of a person that are normally found in association with others. In example if a person is a child it is likely to think she will like to play, go outside, ignore much complicated words etc. All these things would be considered true until evidence of the contrary. Stereotyping in recommender systems works the same way, adding informations that are likely according to data provided.

## 3.5 Hybrid methods

Hybrid methods try to take the best of both item and collaborative approaches. They can be implemented in different flavors: one is to develop separately collaborative and item based recommenders and then mix the results with formulas such as linear combinations. Also voting schemes were used for this purpose, in example in [29] or techniques which choose the best recommender case by case according to some recommendation quality metric such as confidence about the predicted rating. Second approach is to port content-based characteristic on collaborative based system: examples of this are content-based profiles or techniques typical of information retrieval applied in the space of profiles. In example [30] uses Latent Semantic Indexing a very popular dimensionality reduction technique in order to create a view of collections of user profiles represented as term vectors. LSI exploit an algebraic decomposition of matrixes called Singular Value Decomposition that acts on the term-document (user) matrix  $X \in \mathbb{R}^{m \times n}$ , whose rank is  $r$ , decomposing it in three matrixes,  $U \in \mathbb{R}^{m \times r}$ ,  $S \in \mathbb{R}^{r \times r}$ ,  $V \in \mathbb{R}^{n \times r}$  such that:

$$X = US_dV^T$$

with

$$\begin{aligned} S_d &= \text{diag}(s_1, s_2 \dots s_r) \\ U &= [\vec{a}_1, \dots \vec{a}_r] \quad (\text{matrix of left singular vectors}^3) \\ V &= [\vec{v}_1, \dots \vec{v}_r] \quad (\text{matrix of right singular vectors}) \end{aligned}$$

Where  $s_1 \geq s_2 \geq \dots \geq s_r$  are the singular values of  $X$ . LSI truncates  $U$  to the first  $k$  vectors to produce a  $k$ -dimensional subspace in which original points are mapped. The principle is that of extracting the most representative features and so minimize reconstruction error. Objective function of LSI, being  $a$  the transformation vector, can be stated as follows:

$$a_{opt} = \arg \min_a \|X - aa^T X\|^2 = \arg \max_a a^T X X^T a$$

with the constraint

$$a^T a = 1$$

Also IDF index 3.10 has been successfully ported in the context of user space, modeling the idea that universally liked items are not as useful as the ones liked just seldom in capturing a user profile [27]. This is done by scaling the ratings by a term  $f_i$  that represents the frequency an item was marked interesting. Correlation formula so becomes:

$$\text{sim}(u_j, u) = \frac{\sum_{i \in I_{u_j, u}} f_i \sum_{i \in I_{u_j, u}} f_i \rho_{u, i} \rho_{u_j, i} - \left( \sum_{i \in I_{u_j, u}} f_i \rho_{u, i} \right) \left( \sum_{i \in I_{u_j, u}} f_i \rho_{u_j, i} \right)}{\sqrt{UV}}$$

where

$$U = \sum_{i \in I_{u_j, u}} f_i \left( \sum_{i \in I_{u_j, u}} f_i \rho_{u, i}^2 - \left( \sum_{i \in I_{u_j, u}} f_i \rho_{u, i} \right)^2 \right)$$

$$V = \sum_{i \in I_{u_j, u}} f_i \left( \sum_{i \in I_{u_j, u}} f_i \rho_{u_j, i}^2 - \left( \sum_{i \in I_{u_j, u}} f_i \rho_{u_j, i} \right)^2 \right)$$

$$f_i = \log \frac{|U|}{n_i}$$

Assuming that  $n_i$  is the number of users who rated item  $i$  and  $|U|$  is total number of users. Note that if everyone has expressed preference for an item  $\bar{i}$   $f_{\bar{i}} = 0$ , that is equivalent to say that the item brings no information about preferences w.r.t. the dataset. This is formally true when ratings are binary.

Third approach is to mix in a single model items and profile informations. Hybrid methods can also be augmented by knowledge-based system, as in [31], where an ontology is used to make item classification or in Entrée, a restaurants recommendation system [32] that uses ontology to perform user profiling. It was shown empirically that hybrid methods tend to perform better than pure ones [29].

Approach	Recommendation Technique	
	Memory-based	Model-based
Content-based	TF-IDF, clustering	Bayesian classifiers, clustering, artificial neural networks
Collaborative	Nearest neighbor, clustering	Bayesian networks, clustering, artificial neural networks, probabilistic models
Hybrid	linear combination of predicted ratings, voting schemes, inverse user frequencies, dimensionality reduction techniques	building unifying model

Table 3.1: A summary of techniques used for recommender systems

### 3.6 Recommender system evaluation

Evaluating recommender systems and their algorithms is inherently difficult for several reasons. First, different algorithms may be better or worse on different data sets. Many collaborative filtering algorithms have been designed specifically for data sets with specific characteristic, like having more users than items or the contrary, and may reveal entirely inappropriate in a domain where these preconditions do not hold. The second reason is that the goals for which an evaluation is performed may differ. Much early evaluation work focused on the accuracy of collaborative filtering algorithms in predicting withheld ratings. Even early researchers recognized, however, that when recommenders are used to support decisions, it can be more valuable to measure how often the system leads its users to wrong choices. Other work has speculated that there are properties different from accuracy that have a larger effect on user satisfaction and performance. Without going too much in details in this Table 3.6 a short list of measures used for estimating performance. These measures were often designed to evaluate performance for other problems such as information retrieval, classification, and prediction. According to Definition 4 item recommendation can be seen as a particular instance of all of these problems; in fact it is possible to imagine it as:

- An information retrieval problem with no explicit query
- A classification problem over the classes interesting/not interesting
- A prediction problem trying to predict possible values for unrated items

The measures presented were chosen because of their widespread and well known properties: their typical field of application is also reported.

Function	Formula	Application	Description
Mean Square Error	$\frac{1}{ I } \sum_{\substack{i \in I \\ u \in U \\ : \varrho(u,i) \neq \emptyset}} (\varrho(u,i) - \rho'(u,i))^2$	Prediction	Measure the distance between learned rating function $\rho'$ and actual function according to points known a posteriori (items rated by the user after a rating was predicted)
Accuracy	$\frac{\text{Correctly\_class}}{ I }$	Classification	Is the ratio between correctly classified (liked/not-liked) and total number of classified items. Does not take into account the fact positive can be rare so that a all-not-interesting/all-interesting approach can perform good
Sensitivity	$\frac{\text{lab\_pos}}{\text{act\_pos}}$	Binary classification	Is the ratio between items labeled positive (liked) by the system and actual positive. Eliminate the bias for rare positive items that affects accuracy.
Recall	$\frac{ \{\text{Interesting}\} \cap \{\text{Recommended}\} }{ \{\text{Interesting}\} }$	Information Retrieval	Is the ratio between the number of items that were recommended and user showed to like and total number of items the user showed to like. Like sensitivity correct the bias for rare positive items.
Precision	$\frac{ \{\text{Interesting}\} \cap \{\text{Recommended}\} }{ \{\text{Recommended}\} }$	Information Retrieval	Is the ratio between the number of items that were recommended and user showed to like over the total number of item recommended.
F-score	$\frac{\text{recall} \cdot \text{precision}}{(\text{recall} + \text{precision})/2}$	Information Retrieval	Is the harmonic mean between recall and precision and summarizes both, penalizing system which drastically sacrifices one measure in favor of the other.
ROC-curves	$f : [0 - 1] \rightarrow [0 - 1]$	Signal detection theory	ROC-curves are 2-d graphics with percentage of false positive ( $\{\text{Recommended} \cap \overline{\text{Interesting}}\}$ ) and percentage of true positive ( $\{\text{Interesting} \cap \text{Recommended}\}$ ) as variables on the axis $x$ and $y$ respectively. The independent variable is a fraction of the Item set, ordered from most likely to less likely to be interesting. Area below a ROC curve is a measure of performance.

Table 3.2: Some of the measures used in evaluation of recommender systems



# Chapter 4

## Lastnews architecture

Lastnews was designed to be an infrastructure for experimenting and mixing together various recommender system and dealing with common problems linked to web-content surfing. Among these there are:

1. the fact that content comes from many sources at different rates and the wish to aggregate it in a single place.
2. cross-posting, that is the attitude of users at posting the same item in different sites, so that it happens to see it multiple times.
3. URL shortening, that is the possibility to link items with different URLs, another habit which hides information from the user, making her unable to understand where that link points to and, like already mentioned cross-posting, multiplies the possibility of getting the same content more than once.
4. efficient categorization and storing of interesting contents.
5. privacy about what someone is interested in, but also the possibility to share it with others.

The need to handle various sources and the aim of being a general purpose infrastructure for web-content recommendation led to the choice of developing a plugin-based software. Plugins in Lastnews are the components devoted to handle both retrieval and recommendation of items. Lastnews itself want to be as much as possible a hollow structure just providing services to these components. Services offered to recommendation systems by Lastnews are essentially:

- A persistent storage for data
- Sensors to perceive user interactions with items
- Multi-user support
- A modular general paradigm for item recommendation.
- A sophisticated and extensible way of modeling item status w.r.t. a user.

## 4.1 Storage layer

RDBMS are generally a natural choice in most of the cases where persistence is needed, but in a multicolored environment like the one of user generated content it presents a big pitfall: contents from different sources in fact are likely to have different kind of attributes and also plugins which implement recommender systems could easily require to add new data to the items. This is not impossible creating tables ad-hoc for each kind of content and other new tables for plugins requiring extra data, but seemed not to be practical.

Data model chosen to organize the storage is a reticular one, because of its intrinsic support to easily add/remove properties to objects, and in particular RDF, backed by an OWL ontology. The choice of using OWL was driven by different considerations

**OWL offers support to knowledge based recommender system** and being an highly formal standard allows to reuse concepts or implement new ones with minimum risk of not understanding what something really represents.

**OWL natively support URL as ids** OWL offers a natural instrument to represent objects that are already identified by URI

**OWL can address automatically cross-posting and short URLs problems** thanks to non unique name assumption<sup>2.2</sup> and given suitable axioms OWL allows to infer if some items are the same.

### 4.1.1 Openrdf-elmo

To support semantic storage of data Lastnews takes advantage of Elmo framework [33] which supplies a wrapper between Java OOP model and a RDF-triples repository. This framework allows each element of a class in the repository to be materialized inside the program as a persistent object whose methods are roles defined by the ontology. Having in mind a MVC approach methods relative to these persistent objects but that are not getters or setters have to be implemented in separate objects, called *behaviors* in Elmo terminology. A factory object (ElmoManager) permits then to handle behaviors and data merging, which is performed at runtime and allows new methods to be added to existing objects. Behaviors also enable the definition of *interceptors*, methods which are invoked just before or after others, modifying their behavior and supplying an effective way to implement cross-cutting concerns.

#### Repository

Elmo supports various backends for storage: to minimize need for configuration Lastnews works with a single local file, but other solutions such as RDBMS or remote servers are possible. Data are stored in a machine readable format which represents RDF triples organized in graphs. Each user of the application has her own private graph identified by the URI `lastnews:<username>-Private` and shares with other users a public graph denoted by URI `lastnews:Common`. Elmo supplies persistency to objects following the paradigm of *persistence ignorance*: this means that each object is associated with an instance of ElmoManager class, which is in turn connected to one specific graph or the whole repository, and takes care of storing object asynchronously.

While Elmo defines its own query language for RDF triple (SeRQL) and offers SPARQL support as alternative, Lastnews supports only the latter. This choice was driven by the wish to guarantee maximum interoperability and standardization.

## 4.2 Recommendation

The way Lastnews implements item recommendation is called RAES, which is an acronym for Retrieve-Annotate-Evaluate-Select that are the four phases the process is made up of. Each of these is handled by a different kind of plugin, that

acts independently from others.

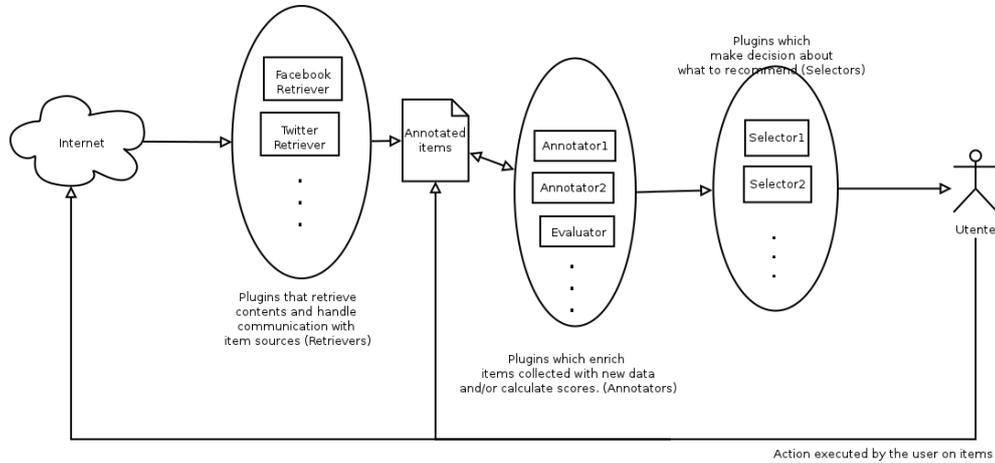


Figure 4.1: Data flow in Lastnews application

In the retrieve phase Retriever plugins fetch items from a specific source. Each of them has a separate configuration and is responsible for all contents from a certain source. How much time has to pass between subsequent polling of the source is set separately for each Retriever. Retrievers often make use of external libraries in order to communicate with sources, define new concepts in the ontology corresponding to the items they get, and set the properties directly available from the source, such as creator or date of creation, for these items. Second step of the process is Annotation. During this phase a second type of plugin, namely Annotator, add additional info to the items retrieved. An example of Annotator could be a plugin making item classification according to the words present in one of its properties, or translating short URLs in their long form. Third phase is still performed by annotators, but of a particular subclass, the Evaluator class. These plugins set a score parameter, that is a score between 0 and 1 according to how much that item would be rated high by the user. At the end of evaluation phase an average of all score parameters is assigned as final score for the item. Last Selector plugins chooses what items are worth recommending to the user.

Separating evaluation, that is prediction of rating according to Definition 4, from recommendation is a key feature of the system. The two tasks also have different properties: while rating should have less variance in the kind of items recommended, as it is based on a lot of data collected over time, selectors,

implementing essentially a rule-based strategy can better model “crisp”, maybe temporary user defined strategies such as “recommend everything with string ‘recommender system’ in it”. Rules are what really affects recommendation and top-N-rated approach, generally given for granted is just one possible rule: allowing to implement pure rule-based systems brings more flexibility and also fits well in a knowledge based infrastructure.

## 4.3 User feedback

User feedback is implicit and collected through Lastnews GUI 6.2.2. User can perform basically 6 operations on an item. Just a qualitative description of how these actions impact on the rating is given, leaving the plugin developer free to adapt exact proportions according to her needs. Built-in operations are:

**Delete** gives negative feedback about the item. Deleted Item are removed from Timeline and not presented again to the user. If the source supports negative feedback about items, item is marked as disliked on the site too<sup>1</sup>

**Like** gives strong positive feedback about the item. Also in this case feedback is forwarded.

**Put aside** (read later) Set the item as not to be shown in the timeline, but not to be excluded from future selections. An item marked as put aside can appear again in Timeline. This action can give little positive or no feedback, depending on specific Evaluator.

**Read** (click) gives no feedback about the Item, but just implies some kind of action has been performed on it by the user.

**Save** gives positive feedback to the Item and marks it not to be deleted from the repository.

**Share** Shares the item on the source site, gives strong positive feedback.

Each of these actions can trigger execution of functions defined by plugin packages.6.3

---

<sup>1</sup>To handle the feedback forwarding is responsibility of the plugin which retrieved the content

## 4.4 Item cycle

Items in Lastnews can be in many statuses, and during a session pass from some statuses to others according to user interaction and selectors choices. For clarity status identifiers will be reported underlined through the text. After being retrieved an Item is parsed in search of at-tags referring to the actual user of the application. If they are found the Item is classified as an at-user Item and its score is set to one. at-user items are a subset of the most general class mustRead which collects all items with score one. Else if no at-tag is found the Item is passed to Annotators that can deal with its specific type. Freshly retrieved items are evaluated together with items that were set as aside by user or application. Items are marked as neverSeen in this phase. After evaluation average of scores set by plugins is assigned as final score for the Item. If it reaches a score of one it is marked as mustRead, status that overwrites neverSeen as these two statuses are incompatible. Items are then passed to selectors. Given an Item each Selector can:

- send the Item to Timeline, marking it as notRead, seen and inTimeline
- set the Item not to be shown in Timeline but not excluding it from possible future selection: this is accomplished marking it as autoAside
- delete the Item, that is mark it as deleted and inArchive<sup>2</sup>. This means the Item will no longer undergo the process of evaluation and selection but will remain in a separate area of the repository.
- Set the Item as inArchive.

Item in Timeline can then be (marked as) shared, liked, saved, deleted (shared, liked, saved, deleted). All of these statuses implies Item is read. Every time new content is retrieved the process starts again so some items are moved from Timeline to Aside and vice-versa.

---

<sup>2</sup>second status is entailed by first

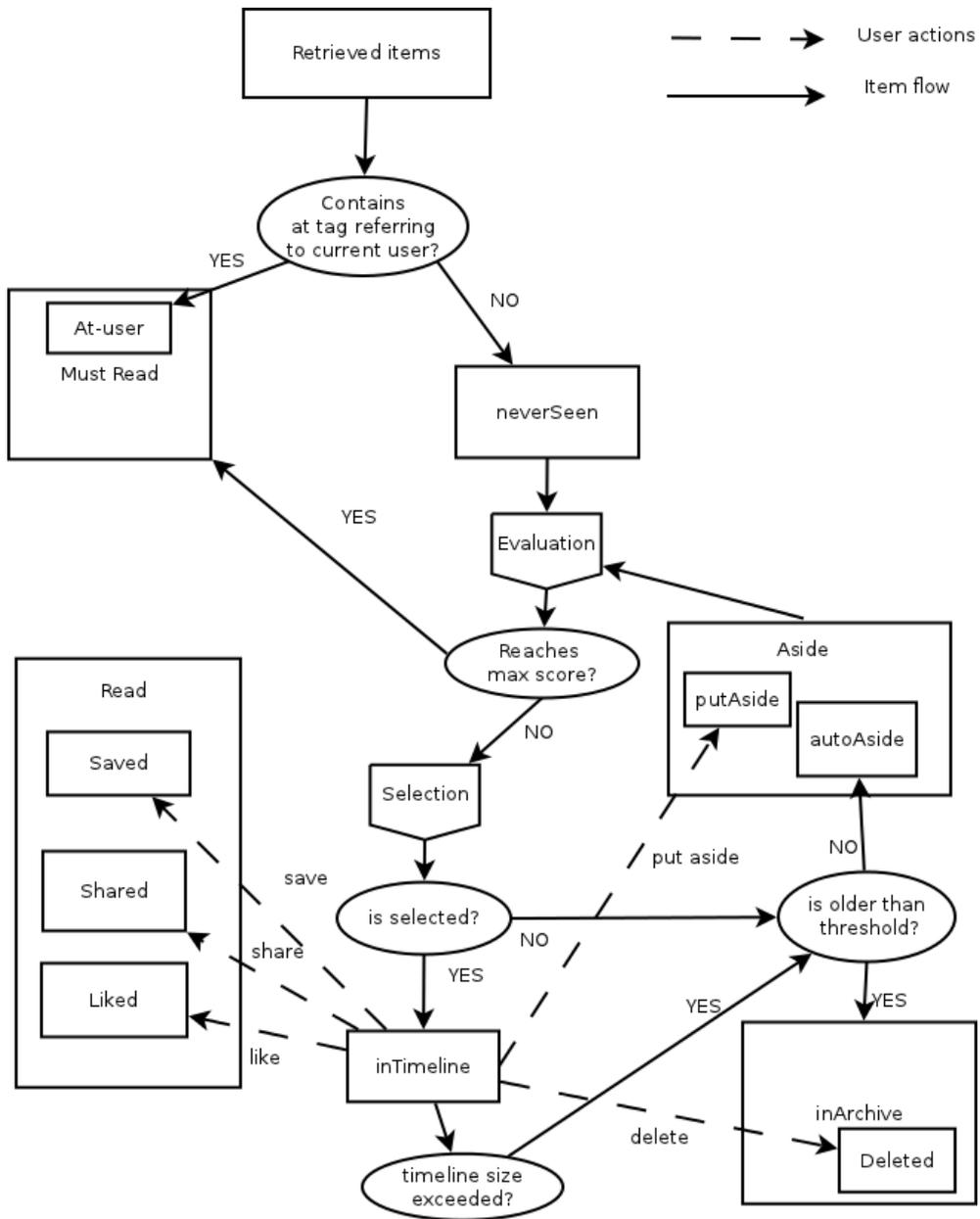


Figure 4.2: Item status flow in Lastnews



# Chapter 5

## Lastnews Knowledge Base

### 5.1 Representational language

The language chosen to represent informations available to the recommender is a subset of OWL 1.0 that supports *SHIN* description logic. The preferred serialization syntax is TriG. The kind of logic used is more of a consequence than a choice: in fact Lastnews ontology imports other ontologies whose combined representational power leads to *SHIN* equivalent algorithmic complexity logic *SHIF* [34]. Below is a short description of ontologies that are directly imported or that were taken into consideration when defining Lastnews concepts.

#### 5.1.1 FOAF ontology

FOAF is an ontology developed by Dan Brickley and Libby Miller “about linking networks of information with networks of people”[35]. It can be seen as a bridge between physical persons and online accounts and so a powerful tool to reconcile information about who wrote or appreciated something on the Web. FOAF, which stands for Friend Of A Friend, can also describe relations among people and indicate their interests. It has *ALCHIF* expressivity.

#### 5.1.2 SIOC ontology

SIOC, which is pronounced as “shock”, is an ontology which provides vocabulary related to online communities. It has a modular design and can be extended by other complementary ontologies. Its core, defined in the namespace `http://rdfs.org/sioc/ns#` supplies the most important concepts used in Lastnews.

Figure 5.1 gives a representation of some classes and their relationships. Apart

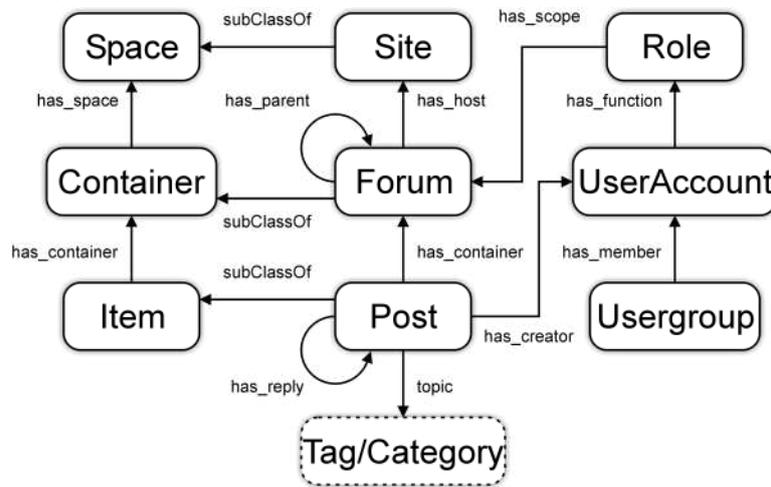


Figure 5.1: Main concepts of SIOC ontology

from core at present SIOC has 3 modules:

**Access** defines concepts relative to user permissions in a certain context

**Services** defines concepts useful to inform the audience of the presence of web-services offered by the site, such as RESTful APIs

**Types** defines various sub-classes of concepts defined in the core ontology, especially content types such as MicroblogPost, Poll, etc.

Just the latter is used in the scope of this work. SIOC uses also some terms from the Dublin Core Metadata Initiative (DCMI) and SKOS ontology.

### 5.1.3 NEPOMUK Annotation ontology

NEPOMUK Annotation Ontology (NAO) is an ontology developed in the scope of the NEPOMUK<sup>1</sup> project, whose aim is to turn the personal computer into a collaborative and semantic-enabled environment, the “semantic desktop”. These ontologies do not rely on standard OWL vocabulary but are instead written using ad-hoc NEPOMUK Representational Language (NRL), which is in turn implemented in RDF. Also the semantic of NRL differs from standard OWL semantics

<sup>1</sup>Networked Environment for Personalized, Ontology-based Management of Unified Knowledge

as the first makes use of the Closed World Assumption. For these reasons NAO-concepts are not used directly inside Lastnews ontologies, but equivalent ones have been re-implemented and many others can be mapped to SIOC or FOAF ontology object properties. Table 5.1.3 gives a short account of these corresponding concepts.

## 5.2 Item classification

As the platform deals with different kinds of contents not all items will have the same set of features: a preliminary problem that is to solve in order to rank posts is so to make input uniform. To do this we have to provide a model for the items and explore what items' features are available or computable, how to represent them and how do they relate to each other and to the probability of an item of being read; final aspect to investigate will be then what kinds of items are comparable to each other and how. A both interesting and problematic aspect to model is that a web-content can embed another via a link. This is a very common use in posts and especially in tweets. Because of this two main classes of items have been defined: items that link another content and item which do not. The first are referred as *Suggestion* in Lastnews ontology, as they suggests something else to read and provide a link to it. Long blog posts with links are not considered suggestions, as the link is not the focus of the item. Lastnews ontology express one equivalence axiom about Suggestion:

$$\exists \text{ln:suggests} \equiv \text{ln:Suggestion} \quad (5.1)$$

Furthermore items retrieved from the net are assigned to Suggestion class by the software if both of these conditions are met:

1. the item content (*sioc:content* property object) is no longer than 140 characters
2. the item embeds exactly one link (that is the link suggested)

Suggestions can further specialize in *DirectSuggestion*, that are Suggestions explicitly sent to the attention of a user. A *DirectSuggestion* can be too identified through some rules and axioms. A *DirectSuggestion* is a suggestion which is sent

NAO Concept	NAO Domain	NAO Range	Lastnews Concept	LN Domain	LN Range	Description
nao:Score	xsd:float	rdfs:Resource	ln:Score	sioc:Item	xsd:float	An authoritative score for an item valued between 0 and 1.
nao:Party	-	-	foaf:Group	-	-	Represent a single or a group of individuals
nao:created	rdfs:Resource	xsd:DateTime	terms:Created	-	rdfs:Literal	States the creation, or first modification time for a resource
nao:creator	rdfs:Resource	nao:Party	sioc:has_creator	-	sioc:User	Refers to the single or group of individuals that created the resource. NAO property is a sub-Property of Dublin Core term in this case
nao:description	-	rdfs:Literal	dc:description	-	-	A free-text account of the resource.
nao:hasTopic	-	rdfs:Literal	sioc:topic	-	-	Defines a relationship between two resources, where the object is a topic of the subject
nao:lastModified	rdfs:Resource	xsd:dateTime	sioc:last_activity_date	sioc:Item	xsd:dateTime	States the last modification time for a resource or content attached to it (e.g. replies)

Table 5.1: Concept reproduced from NAO ontology

from someone to someone. This is stated in the ontology and is a double implication (that is also a Suggestion which is sent from someone to someone is a DirectSuggestion).

$$\begin{aligned} \text{ln:Suggestion} \sqcap \exists \text{ln:from.sioc:User} \sqcap \\ \exists \text{ln:to.sioc:User} \equiv \text{ln:DirectSuggestion} \end{aligned} \quad (5.2)$$

Moreover also items satisfying all of following conditions are classified as *DirectSuggestion*

1. the Item satisfies 1 and 2
2. the Item cites exactly one user (that will be also the object of *to* property)
3. the Item has a creator (that will be also the object of *from* property)

Note that conditions 1 and implications about the object of properties are not expressible by OWL semantics, but would require the use of rules, and thus a logic more expressive than *SHIN*. If author of the suggested Item is known it is assigned to *ln:linked\_author*: range of this property is the concept *ln:Actor*, which is defined as the union of *sioc:User* and *foaf:Person*:

$$\text{ln:Actor} \equiv \text{foaf:Person} \sqcup \text{sioc:User} \quad (5.3)$$

Often in fact it may happen not to know the account of a Person but her name and surname (i.e. scraping articles from an online magazine could be possible to get these data instead of an account id) Apart from Suggestion and DirectSuggestion items are classified according to SIOC-TYPES module hierarchy of concepts. This classification offers ten subclasses of Item. These are described in Table 5.2 In general each plugin of type Retriever defines in its package a specific class of Item that inherits from one of the classes defined in SIOC-TYPES, together with some type-specific properties. In example *TwitterRetriever* defines the type *Tweet* as a subclass of *MicroblogPost* (6.3.3). These more specific concepts are always defined inside a namespace that is unique to the plugin, and imported in the main Knowledge Base when the plugin is installed. What kind of items can be compared with each other, instead, is a problem the ontology does not address directly: all comparisons among items is performed just by mean of some plugin, so each of them is left the responsibility to specify what are the items it can deal with (5.7).

### 5.3 Item properties

The *Item* concept is defined by SIOC ontology as something that can be in a *Container*. While this is a very generic definition Lastnews was thought to deal essentially with text items. The property which contains the body of an Item is the datatype property<sup>2</sup> *sioc:content*, which has Item as domain and *rdfs:Literal*

<sup>2</sup>Languages which support concrete domains distinguish among properties which refer to other objects of the model and properties which refer to values of a concrete domain. In OWL

Class	Superclass	Description
Poll	Item	Describes a posted item that contains a poll or survey content.
Post	Item	An article or message that can be posted to a Forum.
Answer	Post	A Post that provides an answer in reply to a Question.
BestAnswer	Post	A Post that is the best answer to a Question, as chosen by the UserAccount who asked the Question or as voted by a Community of UserAccounts.
BlogPost	Post	A Post that is the best answer to a Question, as chosen by the UserAccount who asked the Question or as voted by a Community of UserAccounts.
BoardPost	Post	Describes a post that is specifically made on a message board.
Comment	Post	Comment is a subtype of <i>sioc:Post</i> and allows one to explicitly indicate that this SIOC post is a comment. Note that comments have a narrower scope than <i>sioc:Post</i> .
InstantMessage	Post	Describes an instant message, e.g. sent via Jabber.
MailMessage	Post	Describes an electronic mail message, e.g. a post sent to a mailing list.
MicroblogPost	Post	Describes a post that is specifically made on a microblog.
Question	Post	A Post that asks a Question.
WikiArticle	Post	Describes a wiki article.

Table 5.2: Item classification according to SIOC-TYPES

as range. Literals are typed string defined by rdf-schema vocabulary: in the case of *sioc:content* property they simply translate to Java String when processed by the application. Analyzing the object of this property other properties are inferred and set at the moment the Item is retrieved. Among these *lastnews:cite* with range *sioc:User*, whose objects are the users cited in the item via at-tags and *sioc:links\_to*, whose objects are the URLs found in the text. Finally also *lastnews:size* is directly inferred from content as it is the length of the text in characters. An Item has also many datatype properties whose object is a date. All of these are subproperties of *terms:date*, from Dublin Core Metadata Initiative, namely *terms:Modified*, for dates in which the Item was modified, *terms:Created*, for when the item was created, *sioc:last\_reply\_date*, for when the Item last received a reply, *sioc:last\_activity\_date*, the most recent date among *terms:Modified*, *terms:Created* and *sioc:last\_reply\_date*, and finally *lastnews:retrieved*, that is always present and denotes when the item was fetched by a Retriever. Being retrieved implies also having a *lastnews:reader*, that is the user who retrieved the item.

The final score according to which an Item is recommended is stored in property *ln:has\_score*, which is calculated in function of *ln:scoreParameter*, a super property of specific scores assigned by Evaluator plugins. Another thing important to be stored are the “likes”, the “share”, the “dislike” an Item receives: for each of this common functions available on social networking sites on the Internet Lastnews ontology defines various properties. Like, dislike and share are processed independently, separating like dislike etc. expressed inside the platform Lastnews or on external sites: in the first case there are two properties for each of these actions, one whose object is the user who performed the action and the second counting how many user performed it. In the second case actions are handled via containers, as explained in 5.4. Replies to an Item are stored too using the *sioc:has\_reply* property.

## 5.4 Item Status

Item status w.r.t. a user was explained in section 4.4. In this section an explanation of how this translates in the ontology is given. Making classes of items in the same status seemed inappropriate because statuses are relative to users, so

---

the first are properties of type *ObjectProperty*, the latter of type *DatatypeProperty*

the solution could have been acceptable just if reasoning was performed in each user context separately. On the other side having a hierarchy of Statuses permits to perform reasoning about the status of an Item. The solution adopted to keep reasoning sound without limiting context and expressing status relative to a User was to reify the statuses as a hierarchy of Containers belonging to a *ln:LNUser* collecting items all in the same state. These special containers are all subclasses of *ln:StatusContainer* and typed according to the status they represent. Their hierarchy is reported in Figure 5.2 Using a Container instead of a brand new class

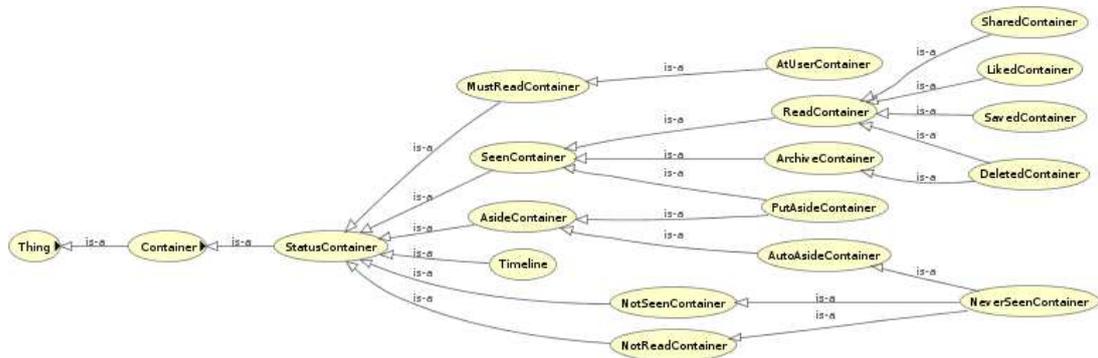


Figure 5.2: StatusContainer Hierarchy

created ad-hoc was chosen for two reasons: the first is that the use of containers reflects the concept of timeline, frequent in applications dealing with items; the second is that it allows reuse and embedding in other applications which support SIOC, as the concept of Container was defined in that ontology. Properties equivalent to *ln:liked\_by* but referring to *ln:LNUusers* can be constructed via SPARQL as shown in Listing 5.1

```

1 PREFIX sioc: <http://rdfs.org/sioc/ns#>
  PREFIX ln: <http://www.semanticweb.org/ontologies
    /2009/9/lastnews.owl#>
  CONSTRUCT {?i ln:liked_by_lnuser ?u}
  WHERE {
    ?cont a ln:LikedContainer .
6    ?cont sioc:has_owner ?u .
    ?i sioc:has_container ?cont .
  }

```

Listing 5.1: A SPARQL query that builds a property which connects items to

LNUser which liked it

## 5.5 User properties

All data belonging to users is stored in semantical repository by Lastnews. User data are handled mainly through FOAF concepts. The concept of User, intended as an account, is present in both SIOC and FOAF, but the two are defined as equivalent by SIOC. Lastnews ontology further specializes it in LNUser, which is the domain of some more properties useful for the application. Every LNUser must have a username unique in the context of application and a password, which is stored in encrypted form. Those data are objects to the *ln:password* datatype property, and *foaf:accountNames*. It is to note that the second is not a functional property: Lastnews keeps it as not functional and deals with his arity via software when it is expected to have just one object. *ln:password* and *foaf:accountNames* properties are not exclusive of LNUser, but both can have *sIOC:User* instances as their subject. Accounts are connected to physical persons via the *sIOC:account\_of* property whose range is *foaf:Person*. As Lastnews is a platform for personal recommendation it does not provide functionalities for accounts of type LNUser held by more than one Person: anyway this is not explicitly forbidden by the ontology. As service homepage are a widely used method to recognize accounts referring to the same service<sup>3</sup> *foaf:accountServiceHomepage* property is set to <http://www.lastnews.com/home> for all LNUser. The ontology itself states this is not a URL, but just a URI.

Lastnews introduces also a property to describe user relationship: *ln:contact*, which is the symmetric equivalent of *sIOC:follows*.

## 5.6 Resource properties

Lastnews defines some general properties that can be assigned to every Resource. These are:

- *ln:is\_private*, with range *xsd:boolean*, denotes if its subject is accessible just to the user who retrieved or created it. This property comes handy when

---

<sup>3</sup>As said before each plugin is left free to define its own types to deal with items and users, so it is possible to have different classes for users of the same service

performing SPARQL queries over the whole repository and not just on a particular graph. As Openrdf-Elmo uses factory objects to deal with each graph it is also useful in order to reassign each Item to the convenient graph after such queries.

- *ln:is\_url*, with range *xsd:boolean* denote if the URI which identifies a resource is also an URL for that resource. This is checked when item is retrieved trying to fetch content from the specified URI, or can be hard-wired by plugins if URL pattern for items is known.
- *ln:screen\_name*, with range *xsd:string* is the name concept instance is referred to in the GUI of the program.
- *ln:is\_short\_url*, with range *xsd:boolean* which states if a resource has a short link as URI/URL and is a subproperty of *ln:is\_url*
- *textitshort\_url\_service*, with range *sioc:Site*, useful in order to identify where to retrieve data about original URL.

## 5.7 Plugin model

Plugins are the core of Lastnews, and are represented by a hierarchy of Concepts in the ontology. Plugins define the kind of Item they work with through the property *ln:operates\_on*, with range *rdfs:Literal*. More specifically, in the context of Lastnews application, this Literal refers to a Java Class. From an application point of view a plugin is a jar package which contains the plugin code. The URL of this package is essential for deployment operation and is stored in the property *ln:package*. Plugins can also specifies dependencies of two kinds through the properties *ln:requires\_package* and *ln:requires\_plugin*. The first indicates that a plugin package defining some concept has to be present in order for the plugin to work, the second that another plugin must be installed and enabled in order to make it works. Property *ln:is\_enabled* tells if a plugin is enabled.

Plugin hierarchy follows the classification of plugins already mentioned in 4.2 and reported in Figure 5.3. Plugin instances are stored in private graph of users and may have some relationship with concepts defined in their own namespaces. Standard namespaces for a plugin follow the pattern `http://www.lastnews.com/plugins/<pluginname>/<concept>`. Retrievers typically hold an account

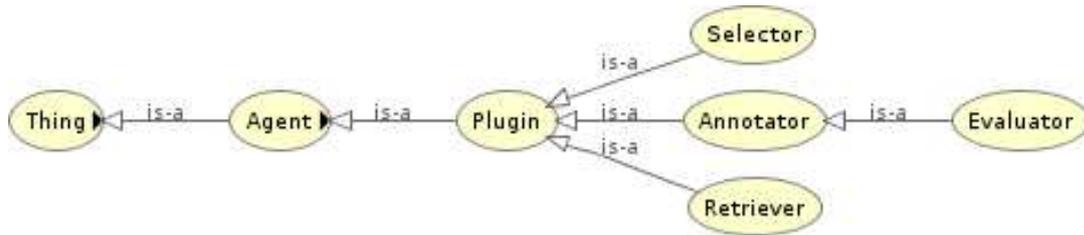


Figure 5.3: Plugin hierarchy of concepts

to perform retrieval operation from a source (property *foaf:holdAccount*) and have a *ln:generate\_private\_content* property to tell if they store items in common or private graph. Some plugins can not avoid defining some additional properties: evaluators, in example, assign their score through a subProperty of *ln:scoreParameter*, whose domain is *Item* and range a float between zero and one.



# Chapter 6

## Lastnews Demo

In order to show some of the concepts exposed in the previous chapters a small demo of the framework was developed. Its main purpose is to give an idea about how such an application could look like. This demo includes the main application, complete OWL ontologies describing the concepts exposed in chapter 5 and more, a Retriever plugin, an Evaluator and a Selector plugin. An (almost) empty plugin package was realized too in order to perform testing.

### 6.1 Tools used

Many tools were used in order to implement Lastnews demo. In particular all project components were coded using Eclipse IDE [36] and built taking advantage of Apache Maven project management tool[37]. The latter is a tool to automate project building and testing according to XML files (pom files) which specifies how to perform each task involved in project development and release such as dependencies resolution and packaging. Final product of a project managed by Maven is some piece of software (an *artifact* in Maven terminology) or a collection of them, in the case the project is made up by more than one module, as in Lastnews. Each of these artifacts can be identified by three coordinates in a Maven repository. Eclipse IDE is an Integrated Development Environment which supports Maven and SCM integration, UML drawing tool and ontology editing through plugins. More specifically m2e [38], subclipse [39], eUML2 [40], Swede [41], were used for Maven integration, versioning through SVN, UML an OWL respectively. Other very important tools used for OWL are Protégé ontology editor [42], and Pellet reasoner [43], used to define the ontology and check consistency.

Lastnews depends on many libraries too. A list of them is reported in Appendix B

## 6.2 Implementation details

### 6.2.1 Main objects implementing Lastnews

#### ContextManager

ContextManager object, singleton, provides all methods and objects that allow to perform operations on Repository, acting as a layer between Lastnews and Openrdf-Elmo APIs. It is also responsible for getting parameters from configuration file. Its most important methods are:

- **Init()**, **Init(*String*)** initializes the fields of the object and bind it to a repository. This is to be called before every other method of the object. Argument is the path of configuration file. If none is given default value is *./lastnews.conf*
- **setMasterKey(PrivateKeySpec)**, **getMasterKey()** are getter and setter for encryption key used by the user.
- **inPrivateRepository(QName)**, **in Repository(QName)** verifies if a Qualified name is present in private or global repository.
- **getGlobalGraph()**, **getCommonGraph()**, **getPrivateGraph()** return an ElmoManager instance to handle repository or common/private graph.
- **getRepository()** returns the Repository, on which is possible to perform queries and operations acting with triples instead of objects.
- **setUser(LNUser)** binds this ContextManager to the User who logged in. Just after this method is called all fields of the ContextManager are initialized and all of its methods can act correctly. The ContextManager is said to be “bound”
- **isBound()**, **isInitialized()** tell if the ContextManager is initialized/bound.
- **getUser()** **getUserURI()**, **getUserId()** return the User, the User URI, a User id useful to create unique names.

- **getPluginManager()** return the instance of the PluginManager.
- **getBasicConfiguration** returns basic configuration of Lastnews as a Property object. This configuration includes just general features, plugin configuration is stored in repository
- **clearRepository()** deletes all data from repository.
- **getAgeThreshold** returns the age after which Items are put in Archive, expressed in days.

### PluginManager

PluginManager, singleton, is the object responsible for handling the plugins and in particular:

- when initialized scans a directory in search of plugins and installs them
- sorts the plugins so that they are executed in the correct order
- allows the user to enable, disable, uninstall plugins.
- connects plugins methods to User Interface events.

PluginManager also handles the recommendation process through its inner class SelectionProcess. This class implements *observer pattern* and gets notified each time some Retriever fetches items from a source. When this happens SelectionProcess gets all Annotators and Evaluators suitable for the type of Item just retrieved through the method *getSuitableAnnotators(Class<? extends Item>)* and annotates the items calling *annotate(Item)* method on each of them. Then a call to Item method *setLastnewsHas\_score()* averages all subproperties of scoreParameter set by evaluators and set the final score to that value. Next all suitable Selectors are collected and each of them performs a call of *select(List<Item>)* on the list of annotated items. The items they return are passed to the GUI and appear in Timeline. A short summary of principal PluginManager methods is given below:

- **getInstance(String)** receives the path to the directory where plugins packages are located, installs new plugins and initializes the object. Return the instance of the PluginManager.

- **getLoadedPlugin()** returns a set of all loaded plugins
- **isActive(Retriever)** tells whether a Retriever is running or if it is stopped or scheduled for stop.
- **registeredPlugin(String)** takes the path of a plugin jar file and checks if it has been imported into Lastnews Repository (has been installed)
- **scanPlugin** is a private method used to scan plugin directory and install new plugin found.
- **startRetrievers()**, **stopRetrievers()** start/stop Threads for those Retrievers whose *is\_enabled* property is set to true
- **uninstallPlugin(Plugin)** uninstall Plugin passed as argument
- **getEdu()** return the EvaluatorDataUpdater object used by the PluginManager to handle signal dispatching to plugins.
- **getPlugin(String)** returns the plugin whose *ln:screen\_name* is equal to argument
- **getSuitableAnnotators(Class<? extends Item>),getSuitableSelectors(Class<? extends Item>)** gets Annotators/Selectors capable of handling specified subclass of Item.
- **reset()** unbind the PluginManager from the current User stop all Retrievers threads and unload all plugins. Called during LNUser logout.

## UserManager

Is the object responsible for creation and management of the User, be they LNUser or some other subclass of User concept. Below is a list of its methods:

- **createLNUser(String, String, String)** creates a new user of the system, ensures her username (first argument) is unique and that passwords match (second and third argument), then creates one StatusContainer for each type and sets its owner to new LNUser, set default *foaf:accountServiceHomePage* for service Lastnews, stores password encrypted.

- **createBaseUser(String, String, Class<T>, ElmoManager)** is used to create a generic User for some (yet) unspecified service. Used by those plugins which need to create new kinds of User into the repository. Class<T> would be the class of the new User instance which is returned. Other parameters are username (the first) and serviceHomepage (the second). An ElmoManager must also be provided to instruct the function about where the new instance will be created (private or common graph)
- **createUser(String, String, String, String, Class<U>, ELmoManager)** like the previous but with encrypted password support (third and fourth arguments is the password).
- **getAllAccountOf(LNUser)** get all accounts whose owner (of type Person) is the same owner of provided LNUser.
- **isLoggedIn()** tells whether active user is logged in (ContextManager is bound)
- **loadLNUser(String, String)** login a LNUser provided username and password

### Item

Item is the main type to which every content downloaded from the web is mapped. Here a short account for those referring to its behavior (implemented in ItemSupport) is given. As many methods have similar functions they are presented grouped.

- **isAside(), isAtUser(), isAutoAside(), isDeleted(), isInArchive(), isInTimeline(), isLiked(), isMustRead(), isNeverSeen(), isPutAside(), isRead(), isSaved(), isSeen(), isShared()** check whether the Item is in State (StatusContainer of type) Aside, AtUser, AutoAside, etc.
- **setAside(), setAtUser(), setAutoAside(), setDeleted(), setInArchive(), setInTimeline(), setLiked(), setMustRead(), setNeverSeen(), setPutAside(), setRead(), setSaved(), setSeen(), setShared(), setUnread(), unsetInTimeline()** set status Aside, AtUser, AutoAside etc. on item or unset it. Status setters takes care of removing incompatible statuses too.

- **save(), share(), like(), remove()** are the actions triggered to perform operation on Item: they do not only set a status but forward the action to the source too.
- **getSharedBy(), getLikedBy()** return the set of LNUser who liked/shared the item
- **mustReadInterceptor(InvocationContext)** interceptor, ensures content with score 1 are classified as MustRead and vice-versa all MustRead have score 1.
- **setDislikeCount(InvocationContext), setLikeNum(InvocationContext), setReaderNum(InvocationContext), setSharerNum(InvocationContext)** are interceptors that automatically set properties corresponding to the arity of other relations (i.e. *ln:reader\_num* and *ln:reader*)
- **setSiocLast\_activity\_dateInterceptor** interceptor to set *sioc:last\_activity\_date*

### 6.2.2 GUI

In order to implement the GUI it was chosen to use Qt/Jambi framework, which provides Java bindings to QT libraries, originally written in C++. QT is a multi-platform toolkit maintained by Nokia which offers many advantages for an application like Lastnews. In particular it includes a very powerful HTML rendering engine (namely Webkit which is used also independently of QT in many projects), and introduces a paradigm known as signal/slot. Signal are what the name says, signals that are delivered to objects which subscribe to them; slots are methods to be triggered by those signals. Widgets have many predefined signal such as “triggered”, “clicked”, etc. and the programmer can create new ones. Each signal can also carry the parameters that are to be passed to his slot. In Lastnews in example each action performed by the user on an item is mapped to a signal which passes the action and the item it was performed on to an EvaluatorDataUpdater that dispatches it to every subscriber. This makes easy adding new triggers via plugins. Evaluation process too uses signals to notify user interface of new items and many User Interface events take advantage of signals too.

## 6.3 Plugins

A plugin is a jar file with a format analogous to that used by Elmo to register new concepts and some extra files needed to specify formally the role of the plugin inside the ontology. Files follow strict naming conventions that allow the packages to be installed and removed easily. Packages are installed on a system basis: installing a plugin makes it available to all users, removing it makes it unavailable for all users. Removing a plugin delete all of its instances and all of the concepts defined in its namespace but subclasses of Item. The package too is deleted from memory at the end of the process. Plugins are installed after the User has logged in, when the PluginManager6.2.1 is created. A folder specified via a configuration file is scanned and each jar found is processed. Plugin packages are recognized because they present the following structure:

- A `com.lastnews.plugins.<plugin_name>` package which contains classes and interfaces which implement the plugin. These are:

**Plugin interface** , an interface mapped to the specific plugin concept, which is expected to end with one of the suffixes '-Selector', '-Retriever', '-Annotator', '-Evaluator', '-Plugin', which specifies all rdf-mapped properties of the plugin.

**Plugin behavior** , an interface which declares all methods that the plugin need to expose but are not already declared in its type interface such as method other than retrieve in a Retriever. Specific plugin behaviors should not be often needed.

**Plugin support** is the class which implements Plugin behavior, if present, and the method relative to specific plugin type. It does not follow any naming convention.

**Support concepts** some plugin may depends on concepts needed just by that plugin. They can be specified and implemented here, if needed.

**package-info** this is a naming convention used by Java: classes named package-info are empty classes used just to set an annotation on a package. The annotation to be set is `@rdf("namespace")`, in order to bind the package to a namespace. It is used by Elmo.

- A `META-INF` folder, which contains files useful for plugin deployment. These are:

**plugin.owl** an OWL ontology serialized in RDF/XML which defines the plugin Concept and its properties. It is copied in the private graph when the plugin is installed. It is located in `META-INF/ontologies` folder.

**extensions.owl** an OWL ontology which defines new concepts used by the plugin. Its triples are copied both in the private and in the common graph. It is not mandatory and must be located in `META-INF/ontologies` folder.

**org.openrdf.elmo.concepts** is a list of classes included in the package which represent a concept

**org.openrdf.elmo.behaviours** is a list of classes included in the package which represent a behavior

**org.openrdf.elmo.ontologies** is a list of couples of the form `<ontology>` = `<namespace>` for each file in `META-INF/ontologies` subdirectory.

- A `com.lastnews.plugins.<plugin_name>.ui` package which contains classes and interfaces used to implement the GUI which allows to configure the plugin. It is not mandatory, and may contain one or more classes:

**ConfigureInterface** is the entry point of configuration interface. It is recognized by naming convention and automatically integrated in the UI. It must be a subclass of `QWidget`.

### 6.3.1 Plugin Behaviors

Apart from methods mapped to RDF properties Plugin also implement behaviors that allow them to perform the task they were designed for. *PluginBehavior* is the interface that defines general functions common to all plugins and useful for handling them inside the framework. A short description of them is given below:

- **configure(Object[])** configures and stores the parameters necessary for the plugin. These will be then used when calling **init()** method. It is triggered by `ConfigureInterface`.
- **init()** is run just once when plugin is activated, before plugin is started. Init internal environment of the plugin (for example in a retriever binds the plugin to the user account). Parameters needed are fetched from the

configuration previously set with **configure(Object[])** method. If no configuration is found an `UnconfiguredPluginException` is thrown.

- **register()** is called just once in Plugin lifetime, during its installation; settles the plugin in the repository initializing RDF properties such as *ln:operates\_on* and *ln:is\_enabled*

Annotators and Selectors implement one more method according to their type of plugin. These methods are, with not much fantasy **annotate(Item)** and **select(List<? extends Item>)** which return an annotated Item and a list of selected Items, respectively. Retrievers implement their characteristic method **retrieve()**, but also extends interface `Runnable` and abstract class `RetrieverSupport` which implements a general strategy for polling sources and Observer design pattern, used to notify `SelectionProcess` when new items are retrieved. A complete view of Plugin classes and interfaces is given in Figure 6.3.1.

### 6.3.2 Dependencies

A plugin *a* can depend on another plugin *b* in two different ways:

- *a* may require some concepts defined in *b* namespaces
- *a* may post-process some items that were already annotated by *b*

In the first case all is necessary is just to have the plugin jar of *b* installed; in the second *b* has to be installed and run before *a*. As there are two dependency type there are two properties to handle them: *ln:requires\_package* and *ln:requires\_plugin*: the former return a set of URLs referring to jar required, the latter Classes of plugins that have to be active. Having dependencies on packages also open the possibility to specify “extensions” packages that have no executable part but just new concepts that may be reused by more plugins. Dependencies also impose an ordering on plugins, such that a plugin *a* that depends on *b* is always executed after *b*.

### 6.3.3 Twitter Retriever

Twitter retriever is a plugin intended to handle items coming from Twitter Microblogging service. Below is a short description of concepts implemented by this plugin.



**TwitterRetriever** an interface with no methods, used just for RDF mapping.

Extends *Retriever*, *TwitterRetrieverBehavior*, *SocialRetrieverBehavior* and *PluginBehavior*

**TwitterRetrieverBehavior** declares the method `getAuthorizationURL()`, which is needed to register the plugin as a Twitter API client according to OAuth [44] protocol.

**TwitterRetrieverSupport** Implements methods inherited by *TwitterRetriever* and acts as a wrapper between *Twitter4j*, the Java library used to handle Twitter API and Lastnews RDF-based representation of Tweets. Annotate date of creation, creator, replies, content and content related properties such as *ln:cite*. In order not to flood Twitter with API requests<sup>1</sup> data about Users other than the account used to login are downloaded only when a User is first seen posting a Tweet.

**Tweet** is the Item type handled by this Retriever. It extends *MicroblogPost* and defines *plugintwitter:statusId* property holding the numeric id of the Tweet, as defined by Twitter.

**TwitterUser** defines properties to store data useful for authentication with OAuth, and the numeric user id assigned by Twitter.

**TweetSupport** extends *ItemSupport* and act as a wrapper between *Twitter4j* and Lastnews libraries, allowing to forward User actions such as like to Twitter.

**TwitterUserSupport** overrides *generateUserURI()* in order to generate User URLs for Twitter users.

### 6.3.4 Bayesian Evaluator

Lastnews Bayesian Evaluator implements a content based approach to item recommendation. Features vectors used for items are binary vector of variables made by couples in  $P \times U$  where  $P$  is the set of properties  $\{ \text{ln:liked\_by}, \text{ln:reader}, \text{ln:shared\_by}, \text{sioc:has\_creator}, \text{ln:liked\_by\_lnuser} \}$ <sup>2</sup> and  $U$  the set of known Twitter accounts union the set of known LNUser. Each time a User likes

<sup>1</sup>Requests are limited at 350 per hour

<sup>2</sup>*ln:liked\_by\_lnuser* is defined as in 5.1

(dislikes) an Item  $i$  the plugin looks for all property-object couples in  $P \times U$  that are true when  $i$  is the subject and update the number  $PCount_{\langle p, u \rangle}$  ( $NCount_{\langle p, u \rangle}$ ) of times an Item for which  $\langle p, u \rangle$  was true was liked (disliked). This is used to approximate the probability  $P(\langle p, u \rangle | Liked)$  as

$$\frac{PCount_{\langle p, u \rangle}}{|Liked|} \quad (6.1)$$

where  $|Liked|$  represents the total number of items liked. The same applies for disliked items too. As there could not be disliked items with feature  $\langle p, u \rangle$  each time a new feature is seen count is incremented by two for  $xCount_{\langle p, u \rangle}$  where  $x$  is  $P$  or  $N$  according to what the user expressed about the item and by one for the other counter (Laplacian smoothing).  $|Liked|$  and  $|Disliked|$  too are initialized to 1 in order not to have zero probabilities.

When it comes to assign a score to an item conditional probabilities are calculated starting from this data and posterior probability (or prior if no  $\langle p, u \rangle$  matching some already seen couple is found) is assigned as value for scoreParameter *plugins-bayes:bayesianEvaluatorScore*. Full algorithm is reported in 3. BayesianEvaluator implements the interface OnlineLearner which is needed in order to connect with signals relative to actions performed on items. In fact it declares the methods that get automatically called by EvaluatorDataUpdater when dispatching the corresponding signal. These are:

- **newLikedItem(Item)** called when user likes an Item
- **newSharedItem(Item)** called when user shares an Item
- **newReadItem(Item)** called when user click on an Item
- **newDeletedItem(Item)** called when user deletes an Item
- **newPutAsideItem(Item)** called when user put an Item aside for reading later
- **newSavedItem(Item)** called when user saves an Item
- **newSuggestionAccepted(Suggestion,Item)** called when a user clicks on an Item suggested by a suggestion
- **newSharedItem(Item)** called when user shares an Item

Bayesian Evaluator ratings are binary: a triggering `newDeletedItem(Item)` gives a negative feedback on that `Item` while a click on `newLikedItem(Item)` `newSharedItem(Item)`, `newSavedItem(Item)` gives positive feedback; `newSuggestionAccepted(Suggestion,Item)` gives positive feedback on both `Suggestion` and `Item`; `newReadItem(Item)` and `newPutAsideItem(Item)` are considered neutral action and so give no feedback. In order to handle feedback data and perform annotation Bayesian Evaluator defines some support classes:

**BCEntry** mapped to an RDF concept collects counts about positive and negative feedback seen together with a specific property-value pair. These are stored as integers in properties `bevaluator:pCount` and `bevaluator:nCount`. Properties `bevaluator:attributeName` and `bevaluator:value` with range `xsd:string` and `rdf:Resource` respectively define what the property and the object the `BCEntry` instance refers to. `BCEntry` is the range of property `bevaluator:BCData`, whose domain is `BayesianEvaluator`.

**ItemBayesianEvaluationInterface** is an interface which specifies the property Bayesian Evaluator annotates on items, namely `bevaluator:bayesianEvaluatorScore`, subproperty of `ln:scoreParameter`. This interface is automatically handled by Elmo and merged in class `Item` once the plugin is installed.

In addition to being connected to `BCEntries` `BayesianEvaluator` also stores some data directly in its properties: total counts of negative and positive feedback is stored in properties `bevaluator:totalNegative` and `bevaluator:totalPositive`, respectively. Finally `BayesianEvaluatorBehavior` defines two properties to store negative and positive feedback: these are **`addNegative(Item)`** and **`addPositive(Item)`**.

### 6.3.5 Roulette selector

Roulette Selector is a very simple kind of selector: it picks a fixed number of items, defined in `roulselector:maxSelected` property with a probability that is proportional to their score. A sketch of the algorithm<sup>3</sup> is reported in 4

---

<sup>3</sup>Version reported is suboptimal but gives an idea of how it works

---

**Algorithm 3** Calculates the probability of being liked according to available data counts and Bayes rule

---

**Input:**  $POS = \{PCount_{\langle p,u \rangle}\}$ , set of PCounts,  $NEG = \{NCount_{\langle p,u \rangle}\}$ , set of NCounts,  $|Liked|$ ,  $|Disliked|$ ,  $IT\_PRO = \{\langle p_i, u_i \rangle\}$ , set of property-object couples in  $P \times U$  set on Item  $i$

**Output:** *bayesianEvaluatorScore*, the probability of being liked

$$\log PProb \leftarrow \sum_{\langle p,u \rangle \in IT\_PRO} \log \left( \frac{PCount_{\langle p,u \rangle}}{|Liked|} \right) + \log \left( \frac{|Liked|}{|Liked| + |Disliked|} \right)$$

$$\log NProb \leftarrow \sum_{\langle p,u \rangle \in IT\_PRO} \log \left( \frac{NCount_{\langle p,u \rangle}}{|Disliked|} \right) + \log \left( \frac{|Disliked|}{|Liked| + |Disliked|} \right)$$

5:

$$\log Odd \leftarrow \log PProb - \log NProb$$

$$bayesianEvaluatorScore \leftarrow \frac{e^{\log Odd}}{1 + e^{\log Odd}}$$


---

---

**Algorithm 4** Select items proportionally to their score

---

**Input:**  $E = [e_1, e_2 \dots e_n]$ : list of evaluated items, *maxItemSelected*: maximum number of items to be selected

**Output:**  $S$ : set of selected items

$i \leftarrow 0$

**while**  $i < \text{maxItemSelected}$  and  $E \neq \emptyset$  **do**

$\text{scoreSum} \leftarrow 0$

**for all**  $e_i \in E$  **do**

5:         $\text{scoreSum} \leftarrow \text{scoreSum} + \text{score}(e_i)$

$\text{probabilities}[i] \leftarrow \text{scoreSum}$

**end for**

$\text{choice} \leftarrow \text{random}(0, \text{scoreSum})$

$\text{index} \leftarrow \text{insertionPoint}(\text{choice}, \text{probabilities})$

10:     $\text{item} \leftarrow \text{itemAt}(E, \text{index})$

$\text{remove}(E, \text{item})$

$\text{add}(S, \text{item})$

$i \leftarrow i + 1$

**end while**

15: **return**  $S$

---

## 6.4 Configuring and running the demo

The demo implements plugins described above and the Semantic annotation infrastructure of Lastnews, a console to query repository through SPARQL, an interface to explore StatusContainers, register users, perform actions on items and manage plugins.

The program reads its initial configuration from a file named `lastnews.conf` that has to be located in the same directory from where the application is launched. That file contains basic configuration for Lastnews, such as where to find the plugins. A sample is given below.

```
_____ lastnews.conf example _____  
1 # Configuration file for LastNews.  
2 timeline_max_size=15 # Max size of timeline  
3 rdfextdir=plugins # directory where plugin are searched  
4 rdfstore=localstore # directory where repository data is stored  
5 # age_limit=1 time before an item is put in archive, in days.  
6 # proxy_address=127.0.0.1 proxy address, not mandatory.  
7 # proxy_port=8080 proxy port, not mandatory.
```

This is the only external configuration file as normally configuration data is stored in the repository. First thing to do when starting Lastnews is creating a new User account. After the User is registered and logged in it is possible to configure plugins. Options to do so are accessible from File > Options > Plugin. If the plugin packages were in the configured directory it should be possible to see plugins mentioned in 6.3 are available. To configure them the user has to click on a Plugin and then on “configure” button. When finished clicking apply will save configuration and start enabled retrievers. After a while items will appear in “incoming” tab. This tab is filled with item provided by Selectors until full or Selector have no more items to choose.<sup>4</sup> When the timeline is full incoming messages will substitute the items with new ones at each iteration of recommendation process 4.2, starting from read items. On the right side of each item is possible to see its score. Hovering the mouse on an Item will show buttons to perform actions *share*, *like*, *Delete*, *putAside*. Performing those action will fill the StatusContainers accordingly. It is possible to explore them using the “View”

---

<sup>4</sup>As RouletteSelector selects just a limited number of items it is normal not to fill it. Tuning `maxSelectedItems` for RouletteSelector and `max size of timeline` parameters avoid the problem

menu. Lastly SPARQL Console accessible from Tools menu allows to perform queries on the repository directly in SPARQL.

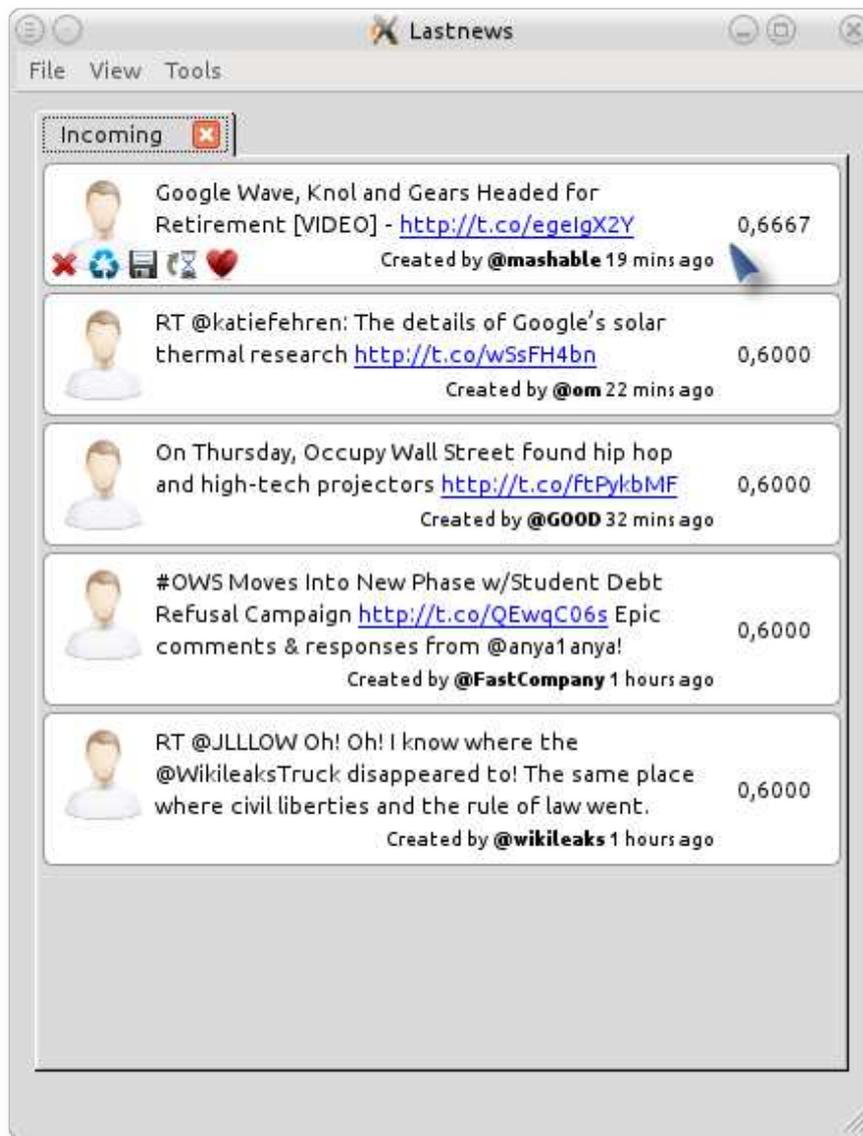


Figure 6.2: Lastnews "incoming" tab showing items in Timeline



# Chapter 7

## Conclusions

During the course of this work an ontology to deal with concepts useful for expressing user preferences in order to support a generic plugin based framework for recommendation has been developed. Recommender systems have been studied for quite a long time by now, still the interest in the field does not seem to fade, while they are getting more and more pervasive especially in contexts such as social Networks and Internet news in general. Despite this there is still no emerging personal software to perform both item aggregation and recommendation, while well known recommender systems got to the point that even small improvements in their performance are very difficult to obtain.[45]. It is our opinion that good results may be achieved making the user himself tune the parameters and methods of recommendation algorithms. So starting from the definition of recommendation problem 4 Lastnews Framework implemented the recommendation process breaking it with high granularity and making it highly configurable, trying to put the focus on user(s) and not on algorithms. Lastnews model, being defined through W3C standard OWL can be easily ported on software other than the demo implemented for this thesis. Emerging functional languages such as Clojure, in particular, may be more indicate than Java for this kind of task, having a paradigm that better support parallelism and ever changing flows of data.

Many extensions to this work are possible: infrastructure to handle users and relationships among them is available but under-exploited. Users may be made capable of writing replies to items they read, introducing new possibilities of feedback. More concepts like the one of “circle”, recently introduced by Google+ service, may be implemented in order to target content more efficiently and perform more complicated Social Network Analysis. Another interesting direction

of research could be expanding actual ontology with SWRL rules, thus allowing to express more complicated concepts and perform better inference. But apart from all extensions the most important thing is probably to experiment and collect data about how much efficient self-tuned recommendation can be compared to traditional systems. These data, when collected will give further ideas and direction for the project.

# Appendices



# Appendix A

## A short history of data indexing and the Web

### A.1 Pre-Internet era

Before the rise of the World Wide Web books were the most relevant medium to store and access knowledge; organizing knowledge was so organizing books. A very clever system, still in use in many libraries nowadays, was invented for this purpose by Melvil Dewey (1851-1831) in 1876 and has overcome 22 revisions until 2004. This system, known as Decimal Dewey Classification, or DDC, is a taxonomy of knowledge that classifies all knowable in a thousand classes. These classes are organized hierarchically so that it is easy to look for a subject knowing which more general matter it is pertinent to. A key feature of the system is that it provides also a decimal-based notation to identify classes, which allows computability.

At the broadest level, the DDC is divided into ten main classes, identified by codes such as  $n00$ , where  $n$  is a digit. Each of this classes is further divided into ten divisions that are in turn divided into ten sections, for a total thousand classes. Second and third ciphers of the code designate subsequent divisions. For example code 961 stands for History of Tunisia and Libya, where '9' stands for the class "History and geography", '6' for "History of Africa" and '1' for "Tunisia and Libya". If much precision is needed a decimal point follows the third digit and division continues proceeding ten by ten [46]. As we can see the system thought by Dewey is a very simple one, a static<sup>1</sup> collection of buckets into buckets

---

<sup>1</sup>Without considering the revision occurred during years

where things to classify are put in, and at the finest level of classification each element is equivalent to another. A great shift in thinking how knowledge can be organized came in 1945: with the end of WWII after a period in which scientists cooperated in the demand of a common cause, many of them realized that sharing knowledge among multiples disciplines could radically improve research; at the same time two other phenomena were noticed: more and more scholarly papers were written and new technologies in order to store and transmit information have been developed. Putting this altogether, in order to allow easy information sharing and at the same time fighting the increasing quantity of academic material being produced, Vannevar Bush imagined a machine called *memex*<sup>2</sup>. This kind of machine was designed according to the principle that alphabetic and numerical classifications are not the ones used by our minds, which in turns operate by association, “in accordance to some intricate Web of trails carried by the cells of the brain”[48]. Memexes are primarily a mean to select information, which store data in a non permanent way, forgetting not frequently accessed items. Documents into a memex are connected to each other by the owner of the memex himself to form *trails*, organized paths of information that could be edited and distributed in form of microfilms. In this kind of setting a new figure should have raised, the *trailblazer* that is a professional trail editor. Bush’s ideas are considered to be the first vision of what the Internet would have been.

## A.2 The Web 0.x

The idea of a text that allowed immediate access to other related texts by association, directly inspired by Bush’s memex [A.1] was developed by Ted Nelson. His idea was that of a file format, called ELF<sup>3</sup> implemented as a collection of interlinked lists that allowed both revision control and non-linear navigation between pieces of information such as strings, tables and images through links. This capability was thought to be particularly useful when writing a document as it made easy to attach related material and annotations. ELF files could have been managed through a language called PRIDE<sup>4</sup> that permitted navigation insertion

---

<sup>2</sup>The term is a portmanteau for memory and index[47]

<sup>3</sup>Where ELF stands for Evolutionary Lists File, not to be confused with the Executable and Linkable format from the System V Application Binary Interface specification

<sup>4</sup>This is the acronym for Personalized Retrieval, Information and Documentation Evolutionary

and deletion of data and links[49]. This was the infrastructure for a new medium, Nelson called *hypertext*.<sup>5</sup>

Independently from him also Douglas Engelbart, the co-inventor of the mouse, was working on a similar idea. His aim was not to provide a better way of handling information, but to provide an instrument to make cooperation among geographically distributed teams easier. What he created was a software named “On-Line system”, which is considered to be the first group-ware ever: it included a journal organized by mean of hypertext links, mouse support and many other modern computer interface concepts. [51]

### A.3 The Web 1.0

In 1984, noticing the lack of a common software to share and present data among physicists at CERN, T. B. Lee, a software engineer who worked there as an independent contractor, wrote a proposal for “a large hypertext database with typed links”[52]. Two things had been stressed in this early proposal:

**hierarchic information flow is unsuitable** for an highly changing environment, even in a hierarchic organization such as CERN.

**ease of access is essential** : the system had to be accessible by remote, by different hardware and distributed. To accomplish these specifications TCP/IP resulted the natural layer on top of which to build.

WWW was so thought as a way to share information among people rather than a system to organize them. The proposal was rewritten in 1990 by Robert Cailliau and work for an effective implementation began at CERN on a NeXT computer. By Christmas 1990 the HTTP protocol, HTML, the first Web server and the first Web browser were created.

Spread of these technologies was highly sped up by the fact that use of Web protocols and related code was granted for free by CERN in April 1993. The first graphical browser MOSAIC was released the same year and opened the door to effective user interaction with web-pages: in the next years Java applet and various kind of scripting began to enrich static content and allowed new kind of

---

<sup>5</sup>Nelson later implemented, together with Andries Van Dam, an hypertext editor called HES that was adopted as a documentation tool by the Apollo mission team at the Houston Manned Spacecraft Center[50]

services relying on the Web, such as web-mail, forums, chats, e-commerce. In September 1994 the World Wide Web Consortium (W3C) was founded, with the aim to promote and regulate open standards for the WWW.[53]

In this kind of environment the problem of indexing and searching rose again: while in the early days catalogs of web-sites organized in form of folders were maintained by hand, increasing number of web-pages made this approach infeasible. Search engines began to appear: among these JumpStation, created at University of Stirling by Jonathan Fletcher in 1993, was the first that implemented all now-standard features: it used a Web robot to crawl and index pages, presented a form as a user interface, answered keyword based queries. Still it didn't perform full text indexing and didn't provide ranking for results. These features were both provided by Google in 1997, with patented algorithm PageRank. This algorithm ranks pages according to formula [54]

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

where  $p_i$  is a page,  $N$  is the total number of pages being considered,  $M(p_i)$  is the set of pages linking to  $p_i$ ,  $L(p_i)$  is the number of outbound links in  $p_i$ , and  $d$  represents a probability that a random surfer visiting a page will click a link on that page instead of starting from a new address. Links resulted to be a successful measure of relevance because they are easy to process and they embed human judge, in the sense that is a human whose interest is to link good content who puts links on pages.

## A.4 The Web 2.0

The term "Web 2.0" does not indicate an update of protocols or any other technical detail of the platform but a change in how the WWW is used by software developers and end-user. The first signs of a change came with the birth of blogs. A blog, that is a short for web-log, is a site composed by a few static pages, a collection of articles (posts), and a user interface that allows the end user to publish them. They're used as a form of public diary or as a way to publish news about some field of interest. Opening the possibility to publish content at a fast pace to much more people than before blogs called for new technologies to get

information from the Web in a convenient way. RSS feeds were a solution for this problem.

RSS, which stands for Really Simple Syndication<sup>6</sup>, is a standard to publish news in an XML format so that they can be retrieved periodically by specifically designed software which can then present them to the user. This was the first widely adopted format for Web syndication. RSS feeds are important to mention because they involve a change of perspective: the Web begins to be perceived not as an archive, but as a channel which transmits data. This is the prodrome of what Tim Berners Lee called the *read/write Web* and is now known as Web 2.0: “a collaborative medium, a place where we [could] all meet and read and write”[55]. Web users are not data consumer anymore, but they become data *prosumer*<sup>7</sup>. Web 2.0 sites are platforms that gives their users the possibility to communicate with each other and to create, view, share, and organize contents. This come in various forms, here are the most relevant for the purpose of this thesis.

#### A.4.1 Social Networks sites

Social Networks sites (or just “Social Networks”, SN) are the prominent phenomenon of Web 2.0. They allow people to share personal information with each other, communicate, and meet new people. In a SN each user build a profile of herself and connect it to other profiles she is interested to be up to date. Linked profiles are referred as *contact* of each other, the “contact” relation being symmetric. The possibility for a user to have contacts will be referred as *Social Network functionality* (SNF); analogous but asymmetric functionality will be referred as *Followership Functionality* (FF) and in this latter case we will have “follower-of” and “followed-by” relations on each side<sup>8</sup>. What makes a site a Social Network site is the focus on profiles. Apart from information about herself, depending on the specific site, the user can publish a variety of contents such as links to web-

---

<sup>6</sup>Formerly RDF Site Summary

<sup>7</sup>This word has varying meaning. In this context, as a portmanteau for producer and consumer, it denotes a person that is both a provider and a consumer of data [56]

<sup>8</sup>While “follower of  $x$ ” is universally recognized as the expression to designate a person who decided to be up to date with content published by that  $x$  there is no agreement on a term to define the followed person w.r.t. the follower. As I found the translation of follower in Italian, my native language, to be “seguace”, a word that is semantically near to “disciple” and “votary”, I would suggest the followed to be a “guru”, but maybe it is a bit too bold

pages, photos, videos, blog-posts, short messages about their current thoughts (referred as *status* of the user) etc. Users may also give feedback via like/dislike votes to other people content and share it again with their contacts. Contents are presented or in the profile page of the publisher or in a user timeline, which aggregates items from contacts.

Among Social networks there are some which focus the profile on user tastes on a particular subject such as movies or music. The user, after registration can rate items concerning the topic and show them on her profile. These sites often provide also a recommender system <sup>3</sup>

#### A.4.2 Bookmarking sites

Bookmarking sites are, together with already mentioned blogs, one of the first sprouts of Web 2.0: they give the user the possibility to share her bookmarks and enrich them with comments or other pieces of information. [del.icio.us](http://del.icio.us) (<http://del.icio.us>) was a pioneer in this kind of service, providing already in 2003 a non-hierarchical classification system for links via *tags*. Tags are one-word meta-data attached to content in order to classify it. They generally denote the main topic of the tagged object, but being totally unconstrained and user provided they can be anything. Such systems where tags can be chosen freely are often referred as *folksonomies*. Tagging and tag clouds, a visualization that print tag bigger according to the number of contents they are attached to, are nowadays a popular Web 2.0 indexing feature. Tag clouds in particular are a useful tool to have an immediate perception of what a user/source of content deals with.

Bookmarking sites also provide SNF, but this is not their focus. In fact this kind of sites are used more as a personal tool to share bookmarks among different computers than as a tool to cooperate in collecting useful links about topics of interest. Some bookmarking sites, such as Diigo (<http://www.diigo.com>), embraced this vocation as a personal productivity tool and provided also tools to place and share notes and highlights on pages; others, such as digg (<http://digg.com>), provided tool to vote positively or negatively links in order to rank them; others too use bookmarks and explicit input by the user to recommend interesting pages: this is the case of StumbleUpon (<http://www.stumbleupon.com>), which asks for a list of topics and then, each time the user presses a button, presents a page on one of those topic. That page can be then bookmarked with tags selected

from a controlled dictionary and rated good or bad, feeding the recommendation algorithm.

### A.4.3 Micro-blogging sites

Micro-blogging sites are a recent phenomenon of Web 2.0 if compared to others, but one of the most interesting. Micro-blogging services give the user Followership Functionality and the possibility to post statuses to followers. These messages, in spite of being short, can provide a lot of information giving the possibility to add two kinds of tag: “hash” and “at” tags. The first are words prefixed with a hash (#) and are used for indexing, while the latter are people ids (usernames) prefixed with a at (@), with the purpose to address a message to someone<sup>9</sup> or linking a profile in a status. Most commonly statuses on micro-blogging sites are used to chat or suggest links. A micro-blog profile page gives very little space to user information focusing on the stream of statuses she posts. They are updated very often<sup>10</sup>, and have a strong diffusion on mobile platforms<sup>11</sup>

### A.4.4 Wiki sites

Wiki sites are sites that can be directly edited by users using a browser and simplified markup language or a WYSIWYG text editor. The software handling this feature also provides versioning of pages, access management and tools to discuss the quality of contents on the site. They are used in a variety of contexts such as online help, collaborative work, knowledge management systems.

### A.4.5 Curation sites

The word “curation” is taken from cultural heritage institution’s jargon and designate the activity of collecting, maintaining and presenting material inherent a certain topic. This is performed by a specialist in that topic, the *curator*. Curation sites are similar to bookmarking ones, but emphasize the aspects relative to making coherent collection of material and presenting them in a convenient way

---

<sup>9</sup>In this case at-tags are prefixed to the rest of the status. This was their original purpose.

<sup>10</sup>On Twitter, the most popular micro-blogging service, it was measured each user posted in average 4.422 status per day[57]

<sup>11</sup>The Twitter client for Android was installed between 5.000.000 and 10.000.000 times as of 18 April 2011[58]

also to users that are different from the one who collected the material. An example is trailmeme ([www.trailmeme.com](http://www.trailmeme.com)), a site published by Xerox whose aim is to implement Bush's memexA.1 on a Web platform. Trailmeme users, referred on the site itself as "trailblazers", can link bookmarked pages creating paths that make sense. Links are not typed, but the user can have alternative links (with a default one) and short descriptions of pages to help with navigation.

Another popular curation site is Pearltrees, that allows the user to classify her bookmarks by mean of trees in which each non leaf node is a subtopic of its parents. According to the fact that each user can choose her own topic hierarchy Pearltrees offers a blend of Dewey and folksonomies approaches. Other features of the site include search for similar topics and possibility to create teams to edit collaboratively the same (sub)tree.

These are just a pair of sites offering curation capabilities focused on content linking done "by hand". Other services, such as paper.li (<http://paper.li>) took a completely different approach aggregating content from sources like Twitter accounts and presenting them in the form of a newspaper, with automatic classification.

## A.5 Real time web and beyond

A definition for the Web of the future is that of "Real-time Web". Thanks to mobile devices and sites that allow fast publishing of material on the Web this become Real-time in the sense that it is constantly kept up to date with fresh news posted just when they're happening. Geo-location services, for example, can be considered a real-time web-service, giving the user the possibility to share where he is "on-the-fly". Going further Internet is more and more an infrastructure for services other than content publishing and it is likely in a future not so far all of daily PC activity will happen with the contribute of some remote machine. This is roughly the concept of cloud computing, or "the Web as operative system" to use the words of technology visionary and entrepreneur Nova Spivack.[59]

# Appendix B

## Libraries required by Lastnews

Name	Maven coordinates		
	groupId	artifactId	version
<b>Lastnews</b>			
Elmo	org.openrdf.elmo	elmo-sesame	1.5
Simple logging façade for Java	org.slf4j	slf4j-api	1.6.1
Foaf concepts	org.openrdf.elmo	elmo-foaf	1.5
rdfs concepts	org.openrdf.elmo	elmo-rdfs	1.5
Sesame runtime	org.openrdf.sesame	sesame-runtime	2.2.4
TriG Support	org.openrdf.sesame	sesame-rio-trig	2.2.4
SPARQL support	org.openrdf.sesame	sesame- queryparser-sparql	2.2.4
Slf4j bridge to Java logging	org.slf4j	slf4j-jdk14	1.6.1
Slf4j extensions	org.slf4j	slf4j-ext	1.6.1
QtJambi libraries	net.sf.qtjambi	qtjambi	4.5.2_01
<b>Twitter Retriever</b>			
Lastnews	it.polimi.lastnews	lastnews	0.0.1-SNAPSHOT
Twitter libraries	org.twitter4j	twitter4j	2.2.2
Twitter core	org.twitter4j	twitter4j-core	2.2.2
<b>Bayesian Evaluator</b>			
Lastnews	it.polimi.lastnews	lastnews	0.0.1-SNAPSHOT
<b>Roulette Wheel Selector</b>			
Lastnews	it.polimi.lastnews	lastnews	0.0.1-SNAPSHOT



# Bibliography

- [1] J. Jacoby, D. Speller, and C. Kohn, "Brand choice behavior as a function of information load," *Journal of Marketing Research*, vol. 11, no. 1, pp. 63–69, 1974.
- [2] R. Bohn and J. Short, "How much information," *2009 report on American Consumers. December*, Dec. 2009.
- [3] M. Siegler, "Eric schmidt: Every 2 days we create as much information as we did up to 2003." <http://techcrunch.com/2010/08/04/schmidt-data/>, 2010. [Online; accessed 22-May-2011].
- [4] J. Alpert and N. Hajaj, "We knew the web was big." <http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html>, 2008. [Online; accessed 15-June-2011].
- [5] M. Arrington, "googles misleading blog post on the size of the web." <http://techcrunch.com/2008/07/25/googles-misleading-blog-post-on-the-size-of-the-web/>, 2008. [Online; accessed 15-June-2011].
- [6] pingdom, "number of domain names pass 183 million up 12 from last year." <http://royal.pingdom.com/2009/06/04/number-of-domain-names-pass-183-million-up-12-from-last-year/>, 2009. [Online; accessed 15-June-2011].
- [7] Verisign, "Domain name industry brief." <http://www.verisigninc.com/assets/domain-name-report-may2011.pdf>, 2008. [Online; accessed 15-June-2011].
- [8] P. Verna, "A spotlight on ugc participants." <http://www.emarketer.com/Article.aspx?R=1006914>, 2009. [Online; accessed 15-June-2011].

- [9] S. B. F. van Harmelen Jim Hendler Ian Horrocks Deborah L. McGuinness Peter F. Patel-Schneider Lynn Andrea Stein, "Owl web ontology language reference." <http://www.w3.org/TR/owl-ref/#Sublanguages>, 2004. [Online; accessed 15-June-2011].
- [10] I. H. Peter F. Patel-Schneider, "Owl 1.1 web ontology language reference." <http://www.w3.org/Submission/owl11-overview/>, 2006. [Online; accessed 15-June-2011].
- [11] "Owl 2 web ontology language profiles." <http://www.w3.org/TR/2009/REC-owl2-profiles-20091027/>, 2009. [Online; accessed 17-June-2011].
- [12] F. Baader and C. Lutz, "Pushing the el envelope further," 2008.
- [13] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, "Tractable reasoning and efficient query answering in description logics: The dl-lite family.," *J. Autom. Reasoning*, vol. 39, no. 3, pp. 385–429, 2007.
- [14] B. Grosz, I. Horrocks, R. Volz, and S. Decker, "Description logic programs: Combining logic programming with description logic," in *Proceedings of the WWW2003 Conference, Budapest, Hungary* (Y.-F. R. Chen, L. Kovács, and S. Lawrence, eds.), (New York), ACM, 2003.
- [15] D. Beckett and T. B. Lee, "Turtle - terse rdf triple language." <http://www.w3.org/TeamSubmission/turtle/>, 2011. [Online; accessed 17-July-2011].
- [16] "Rdf/xml syntax specification (revised)." <http://www.w3.org/TR/REC-rdf-syntax>, 2004. [Online; accessed 2-August-2011].
- [17] E. Prud'hommeaux and A. Seaborne, "Sparql query language for rdf." <http://www.w3.org/TR/rdf-sparql-query/>, 2008. [Online; accessed 17-July-2011].
- [18] U. Shardanand and P. Maes, "Social information filtering: Algorithms for automating "word of mouth"," in *Proceedings of Conference of Human Factors in Computing Systems*, pp. 210–217, ACM Press, 1995.
- [19] W. Hill, L. Stead, M. Rosenstein, and G. Furnas, "Recommending and evaluating choices in a virtual community of use," in *Proceedings of the SIGCHI conference on Human factors in computing systems, CHI '95*, (New York, NY, USA), pp. 194–201, ACM Press/Addison-Wesley Publishing Co., 1995.

- [20] F. Lousame and E. Sánchez, “A taxonomy of collaborative-based recommender systems,” *Web Personalization in Intelligent Environments*, pp. 81–117, 2009.
- [21] W. W. Cohen, R. E. Schapire, and Y. Singer, “Learning to order things,” *CoRR*, vol. abs/1105.5464, 2011. informal publication.
- [22] J. L. D. L. R. M. Montaner, “A taxonomy of recommender agents on the internet,” *Artificial Intelligence Review*, vol. 19, pp. 285–330, 2003.
- [23] T. M. Mitchell, *Machine learning*. New York: McGraw Hill, 1997.
- [24] L. Ungar and D. Foster, “Clustering methods for collaborative filtering,” in *Proceedings of the Workshop on Recommendation Systems*, AAAI Press, Menlo Park California, 1998.
- [25] J. Han and M. Kamber, *Data Mining. Concepts and Techniques*. Morgan Kaufmann, 2nd ed. ed., 2006.
- [26] C. Buckley, “Implementation of the smart information retrieval system,” Technical Report TR85-686, Cornell University, 1985.
- [27] J. S. Breese, D. Heckerman, and C. Kadie, “Empirical analysis of predictive algorithms for collaborative filtering,” in *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, (San Francisco), pp. 43–52, Morgan Kaufmann, 1998.
- [28] E. Rich, “User modeling via stereotypes\*,” *Cognitive science*, vol. 3, no. 4, pp. 329–354, 1979.
- [29] M. J. Pazzani, “A framework for collaborative, content-based and demographic filtering.,” *Artif. Intell. Rev.*, vol. 13, no. 5-6, pp. 393–408, 1999.
- [30] I. Soboroff and C. Nicholas, “Combining content and collaboration in text filtering,” in *Proceedings of the IJCAI-99 Workshop on Machine Learning for Information Filtering*, 1999.
- [31] S. E. Middleton, N. R. Shadbolt, and D. C. D. Roure, “Ontological user profiling in recommender systems,” *ACM Trans. Inf. Syst.*, vol. 22, no. 1, pp. 54–88, 2004.

- [32] R. Burke, "Knowledge-based recommender systems," in *Encyclopedia of Library and Information Systems*, vol. 69, 2000.
- [33] J. Leigh. <http://www.openrdf.org/doc/elmo/1.5/user-guide.html>, 2009. [Online; accessed 11-November-2011].
- [34] E. Zolin, "Description logic complexity navigator." <http://www.cs.man.ac.uk/~ezolin/dl/>, 2011. [Online; accessed 22-May-2011].
- [35] E. Miller and D. Brickley, "Foaf vocabulary specification 0.98." <http://xmlns.com/foaf/spec/>, 2010. [Online; accessed 22-May-2011].
- [36] E. Foundation. <http://www.eclipse.org/>, 2011. [Online; accessed 16-November-2011].
- [37] A. S. Foundation. <http://maven.apache.org>, 2011. [Online; accessed 16-November-2011].
- [38] B. Bentmann, I. Fedorenko, J. V. Zyl, and M. K. et al. <http://eclipse.org/m2e/>, 2011. [Online; accessed 16-November-2011].
- [39] Collabnet. <http://subclipse.tigris.org/>, 2011. [Online; accessed 16-November-2011].
- [40] Soyatec. <http://www.soyatec.com/euml2/>, 2011. [Online; accessed 16-November-2011].
- [41] J. Lerner, T. Self, A. Perez-Lopez, and al. <http://semwebcentral.org/projects/owl-eclipse/>, 2005. [Online; accessed 16-November-2011].
- [42] S. C. for Biomedical Informatics Research (BMIR). <http://protege.stanford.edu>, 2011. [Online; accessed 16-November-2011].
- [43] K. Clark, B. Parsia, and al. <http://clarkparsia.com/pellet/>, 2011. [Online; accessed 16-November-2011].
- [44] E. Hammer-Lahav, "Rfc 5849: The oauth 1.0 protocol," *Internet Engineering Task Force (IETF)*, 2010.
- [45] J. Bennett, S. Lanning, and N. Netflix, "The netflix prize," in *In KDD Cup and Workshop in conjunction with KDD*, 2007.

- [46] OCLC, "Ddc 22 summaries." <http://www.oclc.org/dewey/resources/summaries/default.htm>, 06 2003. [Online; accessed 13-April-2011].
- [47] M. K. Buckland, "Emanuel goldberg, electronic document retrieval, and vannevar bush's memex," *Journal of The American Society for Information Science*, vol. 43, pp. 284–294, May 1992.
- [48] V. Bush, "As We May Think," *Atlantic Monthly*, vol. 176, pp. 641–649, March 1945.
- [49] T. Nelson, "Complex information processing: a file structure for the complex, the changing and the indeterminate," in *Proceedings of the 20th ACM National Conference*, (Cleveland), pp. 84–100, ACM, 1965.
- [50] A. van Dam, "Hypertext '87: Keynote address.," *Commun. ACM*, vol. 31, no. 7, pp. 887–895, 1988.
- [51] Wikipedia, "On line system — wikipedia, l'enciclopedia libera." [http://it.wikipedia.org/w/index.php?title=ON\\_Line\\_System&oldid=39905902](http://it.wikipedia.org/w/index.php?title=ON_Line_System&oldid=39905902), 2011. [Online; accessed 22-May-2011].
- [52] T. Berners-Lee, "Information management: A proposal," tech. rep., 1989.
- [53] Wikipedia, "History of the world wide web — wikipedia, the free encyclopedia." [http://en.wikipedia.org/w/index.php?title=History\\_of\\_the\\_World\\_Wide\\_Web&oldid=417922219](http://en.wikipedia.org/w/index.php?title=History_of_the_World_Wide_Web&oldid=417922219), 2011. [Online; accessed 13-April-2011].
- [54] Wikipedia, "Pagerank — wikipedia, the free encyclopedia." <http://en.wikipedia.org/w/index.php?title=PageRank&oldid=422496124>, 2011. [Online; accessed 5-April-2011].
- [55] M. Lawson, "Berners-lee on the read/write web." <http://news.bbc.co.uk/2/hi/technology/4132752.stm>, 2005. [Online; accessed 13-April-2011].
- [56] Wikipedia, "Prosumer — wikipedia, the free encyclopedia." <http://en.wikipedia.org/w/index.php?title=Prosumer&oldid=420725664>, 2011. [Online; accessed 13-April-2011].

- [57] D. Zarrella, “Is 22 tweets-per-day the optimum?.” <http://blog.hubspot.com/blog/tabid/6307/bid/4594/Is-22-Tweets-Per-Day-the-Optimum.aspx>, 2009. [Online; accessed 18-April-2011].
- [58] “Twitter application page.” <https://market.android.com/details?id=com.twitter.android>, 2011. [Online; accessed 18-April-2011].
- [59] N. Spivack, “The evolution of the web: past, present, future.” <http://www.novaspivack.com/uncategorized/the-evolution-of-the-web-past-present-future>, 2009. [Online; accessed 17-July-2011].
- [60] L. Page, S. Brin, R. Motwani, and T. Winograd, “The PageRank citation ranking: Bringing order to the Web,” tech. rep., Stanford Digital Library Technologies Project, 1998. 17 p.
- [61] D. Diderot, *Encyclopédie ou Dictionnaire raisonné des Sciences, des Arts et des Métiers...* Chez Briasson, 1756.
- [62] “Sioc core ontology specification.” <http://rdfs.org/sioc/spec/>, 2010. [Online; accessed 17-June-2011].
- [63] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.
- [64] G. Adomavicius and A. Tuzhilin, “Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions,” *IEEE transactions on knowledge and data engineering*, pp. 734–749, 2005.
- [65] J. Breese, D. Heckerman, C. Kadie, *et al.*, “Empirical analysis of predictive algorithms for collaborative filtering,” in *Proceedings of the 14th conference on Uncertainty in Artificial Intelligence*, pp. 43–52, Madison: Morgan Kaufmann, 1998.
- [66] G. Linden, B. Smith, and J. York, “Amazon.com recommendations: Item-to-item collaborative filtering,” *IEEE Internet Computing*, vol. 7, no. 1, pp. 76–80, 2003.

# List of Figures

1.1	Daily data consumption of a US citizen . . . . .	10
2.1	Semantic Web stack as described by the W3C . . . . .	13
4.1	Data flow in Lastnews application . . . . .	46
4.2	Item status flow in Lastnews . . . . .	49
5.1	Main concepts of SIOC ontology . . . . .	52
5.2	StausContainer Hierarchy . . . . .	58
5.3	Plugin hierarchy of concepts . . . . .	61
6.1	Plugin classes diagram . . . . .	72
6.2	Lastnews "incoming" tab showing items in Timeline . . . . .	79



# List of Tables

1.1	US User-generated Content creators by content type . . . . .	11
2.1	Main DLs that can be represented by OWL . . . . .	20
3.1	A summary of techniques used for recommender systems . . . . .	39
3.2	Some of the measures used in evaluation of recommender systems	41
5.1	Concept reproduced from NAO ontology . . . . .	54
5.2	Item classification according to SIOC-TYPES . . . . .	56

