

Adaptivity in Entity Subscription Services

George Giannakopoulos, Themis Palpanas

DISI

University of Trento

Trento, Italy

ggianna@disi.unitn.it, themis@disi.unitn.eu

Abstract—Real-word entities can be mapped to unique entity identifiers through an Entity Name System (ENS), to systematically support the re-use of these identifiers and disambiguate references to real world entities in the Web. An entity subscription service informs subscribed users of changes in the descriptive data of an entity, which is a set of attribute name-value pairs. We study the design, implementation and application of an adaptable push-policy subscription service, within a large-scale ENS. The subscription system aims to deliver ranked descriptions of the changes on entities, following user preferences through a feedback-driven adaptation process. The adaptation is based on both the content and the type of each entity change. We evaluate the learning curve of the system and the utility of the content-type discrimination. The experiments demonstrate good results, especially in the system’s content-aware adaptation aspect.

Keywords-Adaptive systems; Artificial intelligence; User modeling

I. INTRODUCTION

An Entity Name System (ENS) [1], [2], [3] is a system that aims to handle the process of assigning and managing identifiers for entities (e.g., people, locations) in the World Wide Web (WWW). These identifiers are global, with the purpose of consistently identifying a specific entity across system boundaries, regardless of the place in which references to this entity may appear.

The *ENS* has a repository for storing entity identifiers along with some small amount of descriptive information for each entity. Entities are described by a number of attribute-value pairs, where the attribute names and the potential values are user-defined (arbitrary) strings.

The *ENS* supports, through an *Access Services* layer, the search for an entity or the update of entities. The search allows the *ENS* clients to identify an entity based on cross-validated and updated data. Updates on the entities of the *ENS* repository can be performed either by inserting a new entity in the system or by changing some of the attributes of an existing entity. System administrators can even merge, split or delete entities.

The administration and checking of entity information is a difficult task in itself, considering the scalability of an *ENS*. To alleviate the burden of such a task and to handle the information need of non-administration clients, we propose the use of a change subscription system, that

helps clients follow the changes of entities in the *ENS* in the time they occur. The subscription system informs users through asynchronous messages (e.g., via e-mail) on sets of changes concerning entities the users have selected. Examples of subscription include subscribing to changes of the “current population” attribute of one’s home town, or of the “phone” attribute of a former colleague.

The system we propose ranks the information on changes in the sent message according to user preferences on what is important, and is able to adapt to the individual use-patterns of each client. This ranking is based on a user model, which is created through machine learning and a feedback-based user modeling methodology. The experimental results show that the proposed approach can successfully adapt to usage patterns, delivering to each user the most relevant and interesting change notifications first. Note that we use *ENS* as a concrete example for ease of exposition, but our techniques can be applied to any similar subscription service, where there are changes of various types and there is some string representation identifying the content of a change.

In the following sections, we define the problem (Section II) and elaborate on our proposed methodology (Section II-A). Then, we present the performed experiments (Section IV), followed by related work (Section V). We then conclude and present future work (Section VI).

II. ADAPTIVE SUBSCRIPTION SERVICE

The subscription service is a supplement to the usability of the *ENS*, tackling the problem of keeping consumers of information concerning specific entities updated. The system disseminates information to consumers¹ about changes on entities to which the consumers have subscribed.

It is clear that a subscription service for changes on entities is not very different from any other kind of subscription service. Nevertheless, the *attributes* of the information on changes partially define the adaptive approach we present within this work (see Section III-A). As we elaborate later in this article, changes can have various degrees of importance, based on their *type*. For example, a “delete” change can be more important than an “insert” change (also see Section

¹The terms *client*, *user* and *consumer* will be used interchangeably throughout the text.

III-A). Then, the *content* of a change is a completely different aspect from its *type*. We consider the content of the change to be the difference between the attributes of the original state of an entity and the attributes of the new state. We argue that this *content* can provide more information than the *type* of the change alone (also see Section IV).

A. Subscription Feedback and Adaptive Content Ranking

The problem we tackle within this work is bringing the information on changes to consumers, in such a way that will best suit individual consumers' needs. The main scenario we face wants a user-consumer to have subscribed to a set of entities from the ENS, in order to get informed about changes in these entities. However, each user may be interested in different kinds of changes. This problem needs an adaptable system to support the modeling and application of user needs and preferences. The system we propose is aware of user feedback in order to improve the users' experience and optimize the flow of information to each individual user.

The problem can be formalized as follows:

Given

- a set of users, i.e., user models \mathbb{U}
- a set of change descriptions $\mathbb{D} = \{ \langle T, C \rangle, T \in \mathbb{T}, C \in \mathbb{C} \}$ of type T and content C , where \mathbb{T} is the set of possible types and \mathbb{C} the set of possible contents
- a set of feedback indications $\mathbb{F} = \{ \langle U, D, I \rangle, U \in \mathbb{U}, D \in \mathbb{D}, I \in \mathbb{I} \}$, where \mathbb{I} a set of importance indicators, either categorical (nominal-scale) or real

we need the system to optimize the estimation function of importance for a new description of a change D_n , $f(U, D_n | \mathbb{F}) : \mathbb{U} \times \mathbb{D} \rightarrow \mathbb{I}$, for our criterion J (described below) that aims at minimizing the following absolute difference: $|f(U, D_n | \mathbb{F}) - I_0(U, D_n)|$, where $I_0(U, D_n) \in \mathbb{I}$ is the user assigned interest value.

In this work the \mathbb{I} set will be considered to be the set of real numbers, i.e., $\mathbb{I} \equiv \mathbb{R}$. This way we can provide a partial ordering of descriptions $D_i \in \mathbb{D}$. Our criterion J is based on whether the ordering of the set of descriptions a user is entitled to get (due to a subscription) is the one *expected by the user*. We do not consider that there is a unique ordering that will suit the user's needs. Instead, we measure the performance of the system based on a devised measure, called Ratio of Acceptable Errors (RAE), defined in Section IV.

III. ADAPTIVE SUBSCRIPTION-BASED UPDATES

The architecture we have devised consists of the *Change Queue*, the *User Profile Database*, the *Subscription Information Broker* and the *Adaptive Information Control* (see Figure 1).

The *Change Queue* is a change log or repository of changes retaining information on the time stamp of a change.

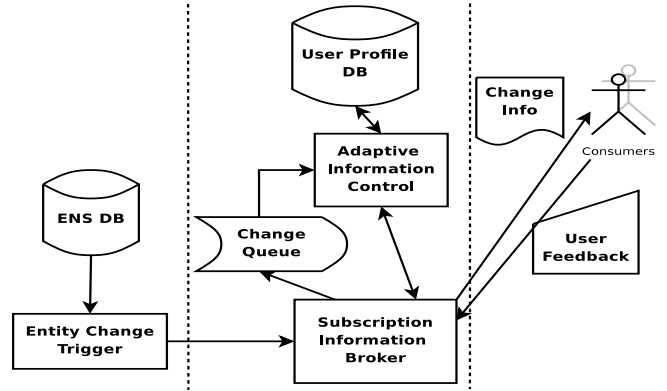


Figure 1. Schematic of the *Subscription Service* and its interactions.

The *User Profile Database*, matches a user-consumer to its profile and holds such information as the time stamp of the last update the user has received.

The *Subscription Information Broker* takes change information and stores them into the *Change Queue*. The broker is also the component that sends the information to the consumers. To do that, the broker requests from the *Adaptive Information Control* component the set of data that need to be sent to active users, on a per user basis. Furthermore, the *Subscription Information Broker* reports the consumer feedback to the *Adaptive Information Control* when the consumer uses the feedback mechanism, to allow for adaptation to consumer needs.

The *Adaptive Information Control* matches each consumer to a user profile, requests from the *Change Queue* the changes that have occurred since the user's last update and ranks and groups the information that will be sent to the user. This information is then passed on to the *Subscription Information Broker* to be disseminated. The *Adaptive Information Control* also uses user feedback provided by the *Subscription Information Broker* to recompute and update the user profiles in the *User Profile Database*.

The asynchronous nature of the overall process, as well as the fact that only *Subscription Information Broker* interacts with external messages, allows for offline and distributed processing, facilitating scalability. This is why no external access is granted to the subscription service database directly. Furthermore, the *Adaptive Information Control* module is a required medium between the feedback mechanism and the user model database, because of the required calculations for the update of the user model, especially as far as it concerns the content of a change (see Section III-A2).

The adaptation of the aforementioned architecture to user needs relies on two main aspects of the *Adaptive Information Control*: the first is how changes' descriptions are represented to form a user model and the second is how the system adapts to user feedback.

A. Modeling Change and Information Importance

We define the description D of a change in a dual way, concerning the *type* and the *content* of the change. For each of these aspects we use a different kind of representation to address their individuality.

1) *Type of a Change*: The *type* T of a description can be viewed as a point in a multi-dimensional space, where the dimensions are:

- One dimension for each alternative of *entity change*: *deletion*, *splitting* of an an entity into two, *merging* of two entities in one, and *entity update*. The value for each dimension can be either 1.0, indicating the corresponding type of change, or 0.0 otherwise.
- One dimension for each alternative of *attribute change*, which is the result of an *entity update* change: *deletion*, *insertion*, *update*.
- The normality of a type of change should indicate, in a quantitative manner, whether a given update appears to be expected. This normality can be judged by such processes as type checking for new values, or by whether a value holds similar qualities to other values for the same attribute. For example a grammar model for a given attribute can indicate whether the new value is normal or not. We use normality as a real value, normalized between 0.0 and 1.0, where 0.0 indicates maximum abnormality and 1.0 perfect normality.

We can now represent a type T of change in a vector space, but the content C of the change is a completely different challenge. In an ENS system, the state of an entity can be described as a set of attribute name-value pairs. Thus, C is determined as the difference between the two states.

Ideally, we want to be able to identify such preferences as “I am interested in changes that have to do with my entities’ name or telephone number”, or “I am interested about when the *inProduction* field of proteins has a value of *true*”. To identify such preferences one must act on the string level and create a model for attribute names and attribute values that are of interest for the consumer. The string representation of the content is the attribute-value pair that was changed, in its changed version if such a pair is applicable to the change. Otherwise, the representation is an empty string.

2) *Content of a Change*: To model interesting attribute name-value pairs, we use the paradigm of character n-gram graphs[4], which can take into account substring matches and offer a set of operators that allow for an updatable model [5].

A *character* n-gram S^n contained in a text T can be any substring of length n of the original text. The *n-gram graph* is a graph $G = \{V^G, E^G, L, W\}$, where V^G is the set of vertexes, E^G is the set of edges, L is a function assigning a label to each vertex and edge, and W is a function assigning a weight to every edge. N-grams label the vertexes $v^G \in V^G$ of the graph. The (directed) edges are labeled by the

concatenation of the labels of the vertexes they connect in the direction of the connection.

The graph is simply constructed by a running window over a given string, that analyzes the string into overlapping character n-grams and records information about which n-grams are neighbours within the window. The edges $e^G \in E^G$ (the superscript G will be omitted where easily assumed) connecting the n-grams indicate proximity of these n-grams in the text within a given window D_{win} of the original text [4]. The edges are weighted by measuring the number of co-occurrences of the vertexes’ n-grams within D_{win} .

We use the n-gram graphs within this work due to several of their traits like language neutrality and the fact that, when used for matching between strings, they offer a graded normalized indication of similarity. We also use the updatability of n-gram graphs, when applied as language models [5]. In other words, if we judge a set of attribute names-values as indicative of importance, we can create an n-gram graph that models the whole set and, thus, avoid keeping all the attribute names-values for matching. Furthermore, the model offers fuzzy matching and substring matching which helps in open domains of attribute names and values, as is the case of an ENS.

To model the content C of changes a user is interested in, we create for each training instance C_i , given by the feedback process, a corresponding n-gram graph G_{C_i} . The graph is based on the string representation of the change.

The model graph construction process for each set of changes (e.g., of the uninteresting/interesting/critical classes of changes) comprises the initialization of a corresponding graph with the first string representation of content, and the subsequent update of this initial graph with the graphs of the other content instances in the class.

Specifically, given two graphs, G_1 and G_2 , the first representing the training set of changes and the second a new instance, we create a single graph that represents the updated model graph G_1 with the graph of new evidence G_2 : $\text{update}(G_1, G_2) \equiv G^u = (E^u, V^u, L, W^u)$, such that $E^u = E_1^G \cup E_2^G$, where E_1^G, E_2^G are the edge sets of G_1, G_2 , respectively. In our implementation two edges are equal $e_1 = e_2$ when they have the same label, i.e., $L(e_1) = L(e_2)$.

The weights of the resulting graph’s edges are calculated as follows: $W^i(e) = W^1(e) + (W^2(e) - W^1(e)) \times l$. The factor $l \in [0, 1]$ is called the learning factor: the higher the value of learning factor, the higher the impact of the second graph to the first graph. As we need the model of a class to hold the average weights of all the individual graphs contributing to this model, the i -th graph that updates the class graph (model) uses a learning factor of $l = (1 - \frac{i-1}{i}), i > 1$. This creates a class model that acts as a representative graph for the class content instances.

As already indicated, we need to create more than one model graphs: one per feedback alternative of the user. Within this work we give the user three alternatives for

feedback: unimportant information, important information and critical information. Therefore, we create three corresponding model graphs. These graphs represent their corresponding content instances in the user model.

In our system the set of \mathbb{I} , which stands for importance, is the set of real numbers \mathbb{R} . We set three qualitatively mapped thresholds of importance: -1.0, which indicates unnecessary information, 1.0, which indicates useful information and 2.0, which indicates critical information. Of course, the set of importance alternatives could have as many elements as desired, keeping in mind that, if $\mathbb{I} \equiv \mathbb{R}$, using higher values for higher importance will probably provide better distinction. Given this kind of mapping, we need to be able to judge the importance $i \in \mathbb{I}$ of a new instance of changes, for a particular user.

B. Ranking Using User Feedback

Having reported on the representation of changes, we can now describe the methodology for assigning importance values to entity changes, based on a user model.

The algorithm we use to learn the user model is actually that of Support Vector Machine Regression (SVR). SVMs have already been successfully used in a variety of applications and settings [6]. Within this work we use the LibSVM library[7] and especially its ϵ -SVR implementation of the algorithm found in [8].

The basic idea behind the ϵ -SVR is that, given a set of training data $\{(x_1, y_1), \dots, (x_l, y_l)\} \in X \times R$, where X is the space of the input patterns we need to “find a function $f(x)$ that has at most” ϵ “deviation from the actually obtained targets y_i for all the training data, and at the same time is as flat as possible. In other words, we do not care about errors as long as they are less than” ϵ , ” but will not accept any deviation larger than this” [9]. In our case, of change types T , $X \equiv \mathbb{R}^d$ which is the vector space we defined for the representation of types $T \in \mathbb{T}$, and $y_i \in \mathbb{I}$.

For the content C of changes, on the other hand, we first calculate the *size-normalized value similarity*[5] between the n-gram graph G_C of a judged C , with respect to each of the n-gram graphs of the user model $G_U^{\text{important}}, G_U^{\text{unimportant}}, G_U^{\text{critical}}$. This similarity value, which lies between 0.0 and 1.0, indicates what part of the graph of C can be found in the corresponding graphs of the model of the user. This set of similarities $\mathbb{S} = \{S_{\text{unimportant}}, S_{\text{important}}, S_{\text{critical}}\}$ is the second constituent of the representation of a description $D = \langle T, C \rangle$ of a change with respect to a user model. To use this set of similarities, we integrate them within the vectors of the type as new dimensions-features. Therefore, when n-gram graphs are used, the overall importance of a change is estimated based on the *combined vector* for type *and* content in an extended input vector space X' .

Table I
ENS USER BEHAVIOUR: PROBABILITY DISTRIBUTION OF CHANGE EMISSION

User type (Prob.)	Change type	Probability
Benevolent (0.95)	Attribute change (normal)	0.60
	Attribute insertion	0.30
	Attribute deletion	0.10
Sys.admin.(0.03)	Entity merge	0.45
	Entity split	0.45
	Entity deletion	0.10
Malevolent (0.02)	Attribute change (abnormal)	0.70
	Attribute deletion	0.30

IV. EXPERIMENTS AND EVALUATION

In order to evaluate our methodology, we had to generate a synthetic set of entity changes, much like other works on adaptive systems (e.g., [10], [11]). The information concerning user behaviour is twofold in our case. We try to replicate the behaviour of the users of the ENS who change entity data. Then, we create profiles for the behaviour of subscribers using the adaptive subscription service.

As behaviour of the users that change the entities’ data, we generate a number of changes’ descriptions D — 10000 instances split into sets of 1000 instances to provide for 10-fold validation. To generate this kind of dataset, we randomly create changes based on a selection from the following user behaviours: benevolent user changes, system administration changes and malevolent user changes. The probabilities of emission per user profile and change type are elaborated on in Table I.

The second part of the evaluation dataset consists of subscriber feedback. We consider a few representative cases of subscribers to minimize the evaluation overhead, while providing useful insight on the adaptivity of the system. A more detailed description of what each user finds interesting and critical can be found in Table II. All changes not noted within a profile are considered uninteresting for the profile. The profiles have been chosen so that they are require different kinds of information to be determined, concerning either the type or the content of the change.

Before the actual evaluation, we use the learning methodology and produce corresponding results, also supplying feedback for every step. This process is reiterated for every subscriber scenario for the whole set of change data. The information supposedly sent to the user is annotated with the iteration number, simulating the timestamp of the change.

To evaluate the system learning effectiveness, as well as whether the addition of content-related features is useful, we experiment on different aspects of the system response.

We determine how quickly the system learns, by “emitting” changes to the supposed user in groups of ten and measuring how well the systems adapts to the feedback. We consider that the user feeds back the system after every new emission, by indicating the importance of all the items in the last group. The performance of the system for every emission

Table II

PROFILE DESCRIPTIONS. *Note:* ALL CHANGES NOT NOTED WITHIN A PROFILE ARE CONSIDERED UNINTERESTING FOR THE PROFILE.

Subscriber	Importance	Description
Type-based	Critical	Attribute deletion.
	Interesting	Entity deletion.
Attribute name-based	Critical	Any change concerning an attribute that contains the string "name".
	Interesting	(None)
Attribute name-value pair-based	Critical	Attribute change or insertion on "isDeceased" attribute, with a new value of "true".
	Interesting	Attribute change or insertion on "isDeceased" attribute, with a new value of "false".
Complex	Critical	Default attribute (some attributes in the ENS are considered default — e.g., the name of a person entity — while all the others non-default) update or insertion with an abnormal value.
	Interesting	Default attribute deletion or normal update.

Table III

CORRELATION BETWEEN EMISSION-ITERATION NUMBER AND MEAN ABSOLUTE ERROR PER SUBSCRIBER PROFILE AND METHOD. *Note:* HIGH STATISTICAL CONFIDENCE IS INDICATED USING **BOLD WRITING**.

Subscriber	Graphs	Correlation (p -value)
Type-based	✓	-0.3398559 ($< 10^{-2}$)
		-0.2993715 ($< 10^{-2}$)
Attribute name-based	✓	-0.3734062 ($< 10^{-2}$)
		-0.03564642 (0.2601)
Attribute name-value pair-based	✓	-0.08581718 ($< 10^{-2}$)
		-0.02968072 (0.3484)
Complex	✓	-0.5989662 ($< 10^{-2}$)
		-0.03393356 (0.2837)

is based on the mean absolute error of the importance estimation. To judge the learning curve of the system, we study the magnitude of the mean absolute error as a function of the current number of emissions.

To determine whether the content aspect of the system is a valuable resource, we try two alternatives. One uses the graph similarities as a feature, while the other does not. When we use content, we expect that the system will perform better for subscriber profiles that indeed use content criteria. For other profiles, we expect no loss in performance.

Table III indicates the Pearson correlation [12] between the number of emission, i.e., the training iteration, and the mean absolute error. A negative correlation indicates that, after performing more training the error is diminished. We see that in all cases, when using graphs, the system learns, with statistical support. In all the cases where we expected that the content should be used, indeed the system cannot learn when the content-sensitive methodology is not applied.

To further understand how quickly the system learns, we define the system performance for a given change emission to be the number of times a ranking error has exceeded 0.5. Given our \mathbb{I} values, errors beyond this 0.5 margin *may* cause an error. Errors below this margin cannot cause an error by themselves. So the performance is *the percentage of the importance estimation in a given set that have their absolute error below 0.5*. Thus, a value of 1.0 in

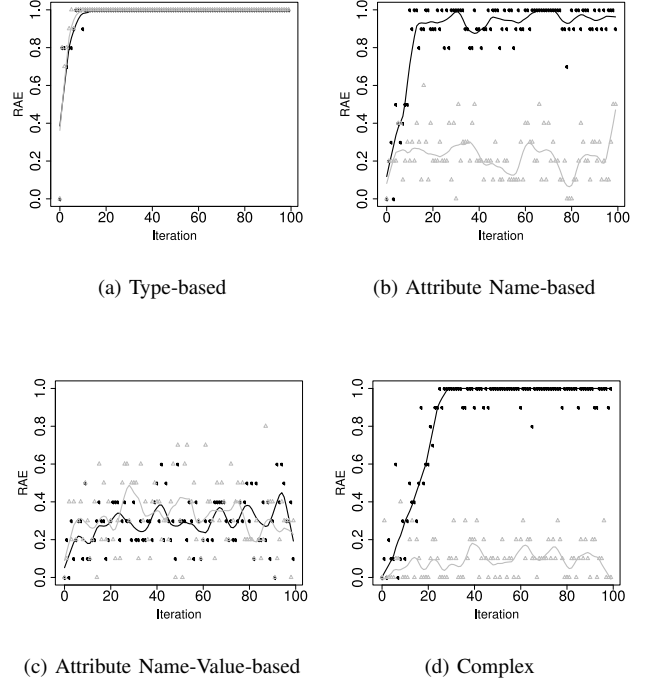


Figure 2. The learning curve for different profiles.

performance indicates a ranking that is ideal, while a value of 0.0 indicates a ranking that will have several errors. We call this measure *Ratio of Acceptable Errors (RAE)*. The formula, for a given set \mathbb{D}_0 of descriptions, their corresponding sequence of importance estimations $\tilde{\mathbb{I}}_0 = \{y_i | y_i = f(D_i, U|\mathbb{F}), D_i \in \mathbb{D}_0\}$ and actual importance values \mathbb{I}_0 , is $RAE(\tilde{\mathbb{I}}, \mathbb{I}_0) = 1.0 - \frac{\sum_{i \in \{1, \dots, |\mathbb{I}|\}} [\min(|\tilde{\mathbb{I}}(i) - \mathbb{I}(i)|, 0.5) + 0.5]}{|\mathbb{I}|}$, where $\tilde{\mathbb{I}}(i)$, $\mathbb{I}(i)$ is the i -th element of the corresponding sequence, $\lfloor x \rfloor$ is the floor operator, $|\mathbb{X}|$, gives the number of elements of a sequence \mathbb{X} , $|x|$ is the absolute value of a number x and $\min(x, y)$ is the minimum function.

RAE as a function of time, as illustrated in Figure 2, indicates the learning curve of the system. The points in the graphs indicate the RAE for a given iteration. We note that there can be several RAE for a given iteration, since we have applied 10-fold validation of the results. The lines represent an approximation (LOWESS smoothing [13]) of RAE over iterations. Dark lines refer to the RAE when using graphs, while gray lines represent the performance without them.

It appears that, when the content methodology is used, all profiles are feasible to learn. The most difficult case appears to be the attribute name-value-based one, probably because we represent the name-value pair as a single string and, therefore, generate features of similarity for them in common, inducing noise in the n -gram graph pattern matching. In other cases very few iterations (< 10) are

enough for the profile to be learned. Even the complex profile is learned in about 20 iterations. This shows that the presented methodology is very promising in learning subscriber profiles on changes.

V. RELATED WORK

We have proposed an adaptive system, related to the definition of adaptive user interfaces: “an adaptive user interface is a software artifact that improves its ability to interact with a user by constructing a user model based on partial experience with that user” [14]. Our system is based on individual user needs and not on “canonical” user needs [15], i.e., group or community needs.

Within this system we use a feedback mechanism, much like the feedback mechanisms proposed in information retrieval since the 1970’s [16]. These feedback mechanisms have relied both on the vector space model [14] or Bayesian modeling [17] to represent user needs. The adaptivity provided is, however, mostly applied on *filtering* tasks [18], [19] or recommendations [20] even through active learning [21].

In this work, the information retrieved is defined a priori and is always sent to the user without any filtering. We use a hybrid approach for the representation of the user model, discriminating between *type* and *content* of change. The *type* of change is identified by vector features engineered for entity change subscription, while an n-gram graph representation is used for the *content* of changes. A function of similarity then allows to add the *content* as a set of new dimensions to the feature space describing a change.

VI. CONCLUSION AND FUTURE WORK

We propose an adaptive subscription service architecture, concerning the update of clients of an ENS with information on entity changes. The subscription service uses information from user feedback to model user needs, taking into account *both the type and the content* of changes. The system appears to learn even complex user preferences in a low number of feedback iterations and offers very promising results in the way it ranks changes for the user.

In the future, we need to study the representation of attribute names and attribute values into different graphs. We further need to determine how the system will be able to cope with interest shifts of a client.

ACKNOWLEDGMENT

This work is partially supported by the by the FP7 EU Large-scale Integrating Project OKKAM “Enabling a Web of Entities” (contract no. ICT-215032). See <http://www.okkam.org>.

REFERENCES

- [1] P. Bouquet, H. Stoermer, and B. Bazzanella, “An entity name system (ENS) for the semantic web,” in *ESWC*, 2008, pp. 258–272.
- [2] T. Palpanas, J. A. Chaudhry, P. Andritsos, and Y. Velegarakis, “Entity data management in OKKAM,” in *DEXA Workshops*, 2008, pp. 729–733.

- [3] B. Bazzanella, J. A. Chaudhry, T. Palpanas, and H. Stoermer, “Towards a general entity representation model,” in *SWAP*, 2008.
- [4] G. Giannakopoulos, V. Karkaletsis, G. Vouros, and P. Stamatopoulos, “Summarization system evaluation revisited: N-gram graphs,” *ACM Trans. Speech Lang. Process.*, vol. 5, no. 3, pp. 1–39, 2008.
- [5] G. Giannakopoulos, “Automatic summarization from multiple documents,” Ph.D. dissertation, Department of Information and Communication Systems Engineering, University of the Aegean, Samos, Greece, <http://www.iit.demokritos.gr/~ggianna/thesis.pdf>, April 2009.
- [6] M. Hearst, S. Dumais, E. Osman, J. Platt, and B. Scholkopf, “Support vector machines,” *IEEE Intelligent systems*, vol. 13, no. 4, pp. 18–28, 1998.
- [7] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, 2001, software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [8] V. Vapnik, “Structure of statistical learning theory,” *Computational Learning and Probabilistic Reasoning*, p. 3, 1998.
- [9] A. Smola and B. Schölkopf, “A tutorial on support vector regression,” *Statistics and Computing*, vol. 14, no. 3, pp. 199–222, 2004.
- [10] U. Cetintemel, M. Franklin, and C. Giles, “Self-adaptive user profiles for large-scale data delivery,” in *Data Engineering, 2000. Proc. of 16th Int. Conf. on*, 2000, pp. 622–633.
- [11] J. Allan, “Incremental relevance feedback for information filtering,” in *Proc. of the 19th annual Int. ACM SIGIR Conf. on Research and development in information retrieval*. ACM New York, NY, USA, 1996, pp. 270–278.
- [12] M. Hollander and D. Wolfe, “Nonparametric statistical methods,” *New York*, p. 518, 1973.
- [13] W. Cleveland, “LOWESS: A program for smoothing scatterplots by robust locally weighted regression,” *American Statistician*, pp. 54–54, 1981.
- [14] P. Langley, “Machine learning for adaptive user interfaces,” *Lecture notes in computer science*, pp. 53–62, 1997.
- [15] M. McTear, “User modelling for adaptive computer systems: a survey of recent developments,” *Artificial intelligence review*, vol. 7, no. 3, pp. 157–184, 1993.
- [16] J. Rocchio *et al.*, “Relevance feedback in information retrieval,” *The SMART retrieval system: experiments in automatic document processing*, pp. 313–323, 1971.
- [17] P. Zigoris and Y. Zhang, “Bayesian adaptive user profiling with explicit & implicit feedback,” in *Proc. of the 15th ACM Int. Conf. on Information and knowledge management*. ACM New York, NY, USA, 2006, pp. 397–404.
- [18] M. Shepherd, C. Watters, and A. Marath, “Adaptive user modeling for filtering electronic news,” in *System Sciences, 2002. HICSS. Proc. of the 35th Annual Hawaii Int. Conf. on*, 2002, pp. 1180–1188.
- [19] X. Zhou and T. Huang, “Relevance feedback in image retrieval: A comprehensive review,” *Multimedia systems*, vol. 8, no. 6, pp. 536–544, 2003.
- [20] D. Bonnefoy, M. Bouzid, N. Lhuillier, and K. Mercer, ““More like this” or “Not for me”: Delivering personalised recommendations in multi-user environments,” in *User Modeling 2007*, 2007, pp. 87–96.
- [21] M.-F. Balcan, A. Beygelzimer, and J. Langford, “Agnostic active learning,” *Journal of Computer and System Sciences*, vol. 75, no. 1, pp. 78 – 89, 2009.