Entity Lifecycle Management for OKKAM[1]

Junaid Chaudhry[a], Themis Palpanas[a], Periklis Andritsos[a], Antonio Mana[b]

[a]University of Trento

[b]University of Malaga

Abstract. In this paper, we examine the special requirements of lifecycle management for entities in the context of an entity management system for the semantic web. We study the requirements with respect to creating and modifying these entities, as well as to managing their evolution over time. Furthermore, we present the issues arising from the access control models needed for the management of a large, distributed repository of entities. Finally, we discuss the research directions that can offer solutions to the above problems, and give a brief overview of techniques and methods relevant to these solution directions.

Keywords: semantic web, entity lifecycle management

1. Introduction

To date, the natural growth path for computer systems has been in supporting technologies such as data storage density, processing capability, and per-user network bandwidth. The usefulness of internet and intranet networks has fueled the growth of computing applications and in turn the complexity of their administration. The wealth of data that is currently found on the World Wide Web (WWW) is of limited use if it cannot be converted into meaningful information.

The Semantic Web (SW) is an evolving extension of the WWW, in which the meaning of data and services is defined by attaching semantic concepts to them, making it possible

[1] This work was partially supported by the FP7 EU Large-scale Integrating Project **OKKAM - Enabling a Web of Entities** (contract no. ICT-215032). For more details, visit http://www.okkam.org.

for applications and machines to make sense of the web content [2]. The SW is evolving towards a direction that leads in understanding, processing and utilizing the information on the web.

One of the major problems that have emerged through the semantic web effort is the problem of uniquely identifying entities[2] [3]. The entities play a major role for the SW since they represent the atomic objects of reference and reasoning. Nevertheless, we currently face the problem of identifying and referencing these entities, which prohibits us from moving to the next step towards the goal of the SW, that of reasoning about entities. The problem derives from the fact that different users, or systems, assign and use different identifiers for the same real-world entity. As a result, we cannot effectively reason about this entity, exactly because it is not consistently being assigned the same identifier.

We claim that the above problem is at the core of the semantic web effort. Along with the problem of assigning global identifiers to entities in the semantic web also come the problems of managing these identifiers throughout the entire lifetime of the entities. Giving solutions to the above issues is the goal of OKKAM [3], a web-scale system for assigning and managing unique, global identifiers to entities in the WWW.

In such a system, with which a large number of users is expected to interact, it is natural to have diversity among user queries about the same entity. This is due to the fact that their knowledge and viewpoints about the specific entity may differ from each other. Exploiting this innate property of system interaction, the entities in the system repository should respond to this diversity and evolve with time. This evolution can be triggered by nascent (i.e., new knowledge) or tacit knowledge discovery (i.e. discovering duplication, coexistence, resemblance, etc.). The aim of this study is to discuss requirements and propose solution directions for the problems of keeping the data in the OKKAM system that are related to the representation of entities consistent, up-to-date and unsullied. We will collectively refer to all the above problems as the problem of *entity lifecycle management*.

The rest of the paper is organized as follows. We review the relevant literature in Section 2, and give an overview of the OKKAM system in Section 3. In Section 4, we describe the requirements of the entity lifecycle management. Section 5 outlines the solution we envision and the research directions we are currently pursuing. Finally, we conclude in Section 6.

---

[2] In the rest of this paper, we will use the term *entity* to refer to individuals, particulars, and instances. This notion of entity is quite liberal, and includes things like products, organizations, associations, countries, events, publications, hotels, people, etc. It may also include fictional objects (e.g., Pegasus), objects from the past (e.g., Plato), or abstract objects (e.g., Gödel's Theorem).

2. Related Work

In this section, we review different approaches related to entity lifecycle management that have been proposed in the literature.

2.1 Data Storage

There has been lots of work on rule-based reasoning for data partitioning and placement techniques [6], but empirical evidence shows the need for reliance of the system on a human user. The placement of data based on recency is proposed in [7]. The most recently queried data are placed in the top tier of the memory, because it is pre-assumed that these data will be accessed again in the near future. Fred et al. [1] propose a storage infrastructure that effectively takes into account not only disk read and writes, but also data creation and deletion. The storage system they propose uses a mechanism based on relative values in order to decide which portions of the data to retain in the case of space shortage. Mirta et al. [8] propose and evaluate query-based partitioning, a novel approach for partitioning documents and indexes across the storage hierarchy, based on the insight that documents not present in the top-K results of a query are unlikely to be accessed through that query.

Various techniques that employ different strategies have been proposed for efficiently storing different versions of data objects [37][38]. Versioning has also been studied in the context of semi-structured documents [39], and efficient query answering algorithms have been proposed [40].

2.2 Data Lineage

When entities are created and modified, we are interested in keeping track of information related to the provenance of the entity data stored in the repository [34]. An important issue in data providence is its characterization. That is, to find the answers of questions like "*why is a piece of data in the output?*" and "*where is the piece of data copied from?*". Buneman et al. [35] target these issues and propose a framework for describing and understanding provenance using a special tree-like model, where the location of a piece of data can be uniquely described by a path from the root of this tree.

Sometimes the propagation of annotations is dependent on the syntax of the query. One may want to control the propagation of annotations in a schema. The custom propagation schemes allow the user to specify where to obtain annotations from. Bhagwat et al. [36] present propagation schemes that are essentially based on where data is copied from.

2.3 Entity Resolution

A lot of structural heterogeneity is expected in the OKKAM system (for example, representing a date as *year/month/day* in place *day/month/year*, or the location of a room as *room number-building-university* in place of *university-room number-building*). For such situations various *data cleaning* techniques are necessary [11].

Elmagarmid et al. [9] target the *record linkage* and *record matching* problems, and enumerate various techniques suitable for resolving lexical heterogeneity. They divide the duplicate record detection techniques into two broad categories: *ad hoc* and *probabilistic*. The ad hoc methods perform efficiently on existing relational databases. On the other hand, the probabilistic methods outperform ad hoc techniques in terms of accuracy. However, they are only efficient for relatively small datasets.

Jaro et al. [13] use the linear sum assignment model to pair related records together. They calculate a string similarity numerical value obtained by taking into account all the characters of the corresponding strings. Then, a weight adjustment strategy is applied, which leads to an efficient way for recognizing typographical errors. Using the Bayes decision rule for minimum cost, Dempster et al. [14] use the expectation maximization algorithm for parameter estimation from incomplete data. Winkler [15] introduces probabilistic methods of accounting for certain types of typographical variation (and hence, duplicate detection). An important point of this research is that no calibration datasets are needed to train the algorithm.

The ALIAS system [19] uses the "reject region" scheme to reduce the size of the dataset, but it needs human intervention when the level of uncertainty is high. Using rule-based approaches [27][28][29], it is easier to create a large number of training pairs that are either clearly non duplicates or clearly duplicates. Despite that, the rule-based approaches require user intervention in rule management scenarios. Recent approaches have also focused on the problem of how to efficiently support the duplicate identification operation in the context of relational database systems [22][24].

Duplicate detection through record linkage has also been studied [16][17][18][26]. These approaches are based on different flavors of clustering algorithms. McCallum et al. [12] discuss *anaphora resolution,* where the problem is to locate different mentions of the same entity in a document. They propose the use of *canopy like* structures for speeding up the duplicate detection process. The first step is to use a cheap comparison metric to group records into clusters. Then the records are compared in a pair wise fashion, using a more expensive similarity metric that leads to better qualitative results. Benjelloun et al. [10] propose three algorithms for solving the entity resolution problem, namely, G-Swoosh, R-Swoosh, and F-Swoosh. These algorithms take into account the characteristics of the match and merge functions, and can also provide approximate results.

3. Background

In this section, we give a brief overview of the OKKAM system (a more detailed presentation can be found elsewhere [3]). We will use OKKAM as the basis for our discussion on the requirements and solution directions for the entity lifecycle management, exposed in the rest of this study. Note however, that our discussion is not restricted to the OKKAM system, but is general, and relevant to any system for entity identification management.

The overall goal of OKKAM is to handle the process of assigning and managing unique identifiers for entities in the WWW. These identifiers are global, with the purpose of consistently identifying a specific entity across system boundaries, regardless of who and where references this entity. Figure 1 shows a high-level *conceptual* view of the OKKAM system and the interactions with its environment.

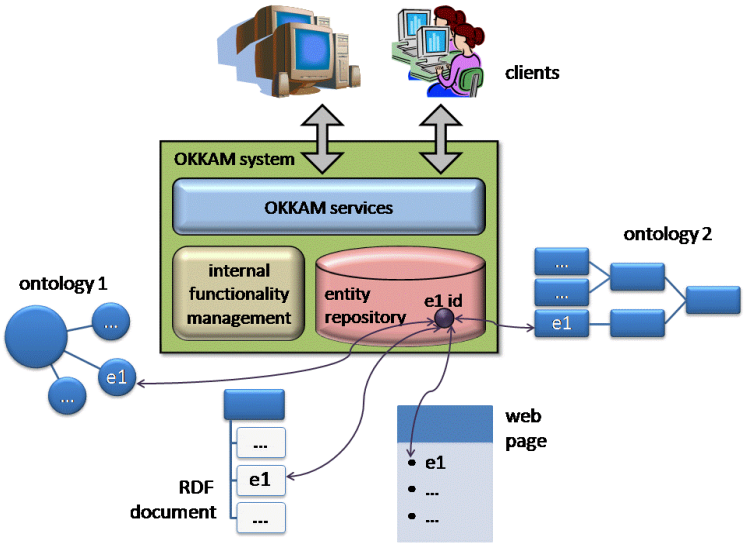

Figure 1: Schematic of OKKAM system and interactions.

The OKKAM system has a repository for storing the entity identifiers (note that in reality this repository will be distributed and replicated) along with some small amount of descriptive information for each entity. The purpose of storing this information is to use it for *discriminating* among entities, not exhaustively describing them. Note that there is no fixed schema for this kind of information. Entities are described by a number of attribute-value pairs, where the attribute names and the potential values are user-defined (arbitrary) strings. The exact number of attribute-value pairs used to represent an entity is not preset, but may vary according to the information provided for each specific entity.

Clients interact with the system through the OKKAM Services layer. Clients can be both human users and applications. There are two types of interaction. First, clients inquire about the identifier of an entity by providing a set of attributes that describes this entity. If the entity exists in the repository, the system returns its identifier. Second, clients may insert a new entity in the system. The system returns the newly assigned identifier.

As shown in Figure 1, the end result is that all instances of the same entity (i.e., mentioned in different systems, ontologies, web pages, etc.) are assigned the same OKKAM identifier. Therefore, entity identity resolution becomes trivial, and is done without any further interactions with OKKAM.

4. Requirements for Entity Lifecycle Management

In this section we discuss the requirements for entity lifecycle management, in the context of a large, distributed repository of entities for the semantic web. We start by examining the representation of entities. Then, we present the issues arising for the processes of creating, modifying, versioning, and merging entities, as well as keeping track of information relevant to provenance and access control.

4.1 Entity Representation

The OKKAM system is designed to store arbitrary entities, referring to very diverse domains, including (but not limited to) persons, buildings, documents, and products. As such, the representation of the entities in the system has to be flexible in order to accommodate the requirements of all the different domains. Note that in OKKAM we are merely interested in assigning and managing unique ids to entities, which means that we do not need to represent all the known information about an entity, but rather only a small amount of data that can help us discriminate this entity from all the rest. Nevertheless, the set of data that need to be stored can vary drastically among entities.

4.2 Creating and Modifying Entities

The creation of new entities can be initiated through one of the following two ways: an automatic OKKAM-ization process, or a manual interface-based method. When a document is parsed, entities are identified using an automatic *entity identification* process that is part of the OKKAM-ization process. When new entities are created, we have to check if these entities already exist in the system. This process can be of two kinds: (a) Offline Check: the entities are added first and checked for duplication later, and (b) Online check: the entities are checked for duplication before they are actually added to the database.

If the entity is unique, it is assigned a unique identifier, and stored in the system. After the entity has been added to the system, it is subject to updates. New attributes may be

added to the description of the entity, or the values of existing attributes may change (e.g., when the information is outdated). If the description of an entity changes, we once again have to check if this entity is a duplicate of another entity stored in the repository of the system.

4.3 Data Lineage

When entities are created and modified, we are interested in keeping track of information related to the provenance of the entity data stored in the repository [34]. This includes information related to the source of the corresponding data, the owner and creation and modification time of an entities schema.

The above information can potentially be very useful for other algorithms operating on the entities in the repository, such as matching and merging. For example, this kind of information can allow the matching algorithm to differentiate between attributes modified by humans (i.e., manually) and by computers (i.e., automatically), as well as between attributes that were changed in the recent past (i.e., up-to-date) and the distant past (out-of-date). Obviously, the knowledge of how and when each of the data representing an entity has been edited may lead to different strategies for performing entity matching.

The information on data lineage can refer to each entity as a whole, or be more fine-grained, and refer to each individual attribute of every entity in the repository. The latter alternative results in a much more detailed view of how all the entity data was inserted in the repository, but also leads to higher space requirements and management cost. The proposed solution will have to take into account this tradeoff between flexibility and implementation cost.

4.4 Versioning of Entities

Regular updates of the attributes describing an entity lead to the creation of different versions of the same entity. In several cases, it is beneficial to store old information for future reference. For example, queries may ask for entities based on old attributes/attribute values.

When a query is performed, the results returned to the user are from the latest information that has been collected after the update procedures. In many cases, once the information is updated, the old information is deleted. Of course we have to define a limit (by date, or by version) with respect to the storage of old records. The user should be given this facility to query across different versions of the entity or in some cases, search the changes that have been performed on an entity over a certain period of time.

The possibility of having different versions of the same entity raises some efficiency questions. The OKKAM system will be deemed efficient if it entails the following properties with respect to efficiency.

- Storage: it is important to define the storage schemes and physical representation for versions and the *up-to-date* information.
- Query partitioning and redirection: Defining whether the user wants to search into the *up-to-date* records or the version, and appropriately redirecting the query.
- Search and matching: the searching algorithms that are going to fetch the query results for the user.

4.5 Merging Entities

Merging of entities may take place when one or more entities exceed some threshold of similarity. Before making this decision, one has to carefully take into account the nature of the considered entities and the tools used to compute this similarity.

As we have mentioned, entities are described by a set of attributes. The values of these attributes can be either numerical (e.g., year, age) or categorical (e.g., name, address). In these cases, it is evident that the same proximity measure cannot be used. Therefore, suitable distance measures will need to be devised, which will also take into account the different characteristics of the various types of attributes.

As more and more entities are added in the system, it may be the case that a single entity is represented by multiple instances in the repository. In such cases, we would like to employ techniques in order to detect such situations, and merge the duplicated entities. These techniques will be monitoring the evolution of entities and their attributes and/or corresponding values and will take the necessary actions either automatically or after interacting with the curator of the OKKAM repository.

Ideally, we would like to be able to identify duplicates at the time when new entities are inserted in the system, using on-line algorithms. This assumes that (a) summaries exist to store the necessary information from existing entities, and (b) a proper threshold is used to guide the decision of whether the entities can be merged or not. We envision the use of techniques used in clustering to build attribute and attribute value summaries, and use them to decide whether a new or existing entity can be merged with one or more entities in the repository.

4.6 Access Control

An important issue that we have not discussed so far is the access control (AC) model. This is probably the most important aspect to take into account regarding the security of the OKKAM infrastructure from the users' point of view. The huge amount and the heterogeneity of the information stored require very scalable and flexible access control mechanisms in order to adequately protect each individual piece of information. Additionally, the information stored in OKKAM repositories must respect different (and sometimes very strong) privacy requirements. Furthermore, the information to be

protected is not only the one returned in the queries to the repositories, but also the information that can be derived (for instance, by means of data mining techniques) from the data stored.

Solving this situation requires the ability to define restrictions on the data that can be used for searching, and data that can be returned as a query result. For the access control scheme to be able to suit the needs of OKKAM we have identified the following sub-requirements: Flexibility; scalability; manageability; and provision of advanced features such as controlled delegation, owner-retained control, dynamic self-adaptability (i.e. content-based and context-aware access), anonymous access and provisional authorization. All these characteristics pose important challenges in the protection and access control mechanisms.

Furthermore, we believe that the importance of automation and ease of management must not be underestimated: due to the large-scale and distributed nature of OKKAM it is essential that security management is highly automated and does not require a lot of administration, as otherwise the system would rapidly become vulnerable due to the human limitations of the administrators and the high complexity of the system. In fact, this aspect, which is usually overlooked when designing a security infrastructure, is the most important source of vulnerabilities and attacks.

5. Proposed Approach

In what follows, we briefly outline the directions that we will pursue in order to address the issues related to entity lifecycle management we identified in the previous section.

5.1 Entity Representation Conceptual Model

In OKKAM, we represent an entity E as a tuple $<\boldsymbol{P}, \boldsymbol{M}>$. We call the set $\boldsymbol{P}$ the *profile* of the entity, because it stores the unique identifier of the entity, and contains information that specifies the (semantic) type of an entity, as well as the relationship of the entity with other entities in OKKAM (if we know that it is identical to another entity stored in the system), or outside OKKAM (if there exists another id assigned to the entity by another system). The entity profile is also composed of a set of arbitrary, used-defined attributes that describe the entity. For example, if the entity is a person, possible attributes are *name*, *date of birth*, and *nationality*. Note that this set of attributes can be different for every entity, even for entities in the same domain.

The set $\boldsymbol{M}$ refers to the metadata of the entity. These metadata are used to support complex algorithms for the other functionalities offered by OKKAM, such as entity matching that based on the attributes in $\boldsymbol{P}$ identifies if two entities are the same or not. Examples of the information that these metadata may carry are the creation time and usage patterns of the entity, and of each individual attribute in $\boldsymbol{P}$. The information in $\boldsymbol{M}$ can also be used to store some simple information regarding the version history of the

entity. Nevertheless, specialized data structures are needed in order to keep track of the history of changes in ***P***.

5.2 Processing of Usage Patterns

The way the users access the system and interact with it may determine several aspects of the entity lifecycle management. Consider the following example. Assume that many users search for an entity with attributes A_1 and A_2, and always select entity E_1, which is the only entity in the repository that contains attribute A_1 in its profile. If E_1 does not contain A_2 as well, we may choose to add it to the profile of E_1, because many users refer to E_1 using A_2.

Alternatively, assume that the query for entities with attributes A_1 and A_2 returns n entities, E_1, E_2, … , E_n, that satisfy the search conditions, but the interested users always select entity E_k, $1 \leq k \leq n$. In this case, we may choose to increase the importance of entity E_k, so that it ranks first for the particular query.

In both the above situations, we are interested in monitoring the usage patterns of the system, in order to obtain some knowledge that can help the system perform better. By monitoring and analyzing the way users interact with OKKAM, we can determine which entities, or profile attributes, are relevant to specific queries or to certain contexts. This information can subsequently be used for updating the profile or the metadata of the entities, and ultimately for producing more relevant search results. The above kind of processing can be done automatically, in an online fashion, and be flexible enough to allow effective and efficient data analysis [5][41], as well as time-decaying representations that evolve with the arrival of new data [4]. Some of the results of the above processing will be part of the entity representation (stored as metadata). If necessary, additional data structures will be created as well, in order to fully exploit the knowledge hidden in the usage patterns.

5.3 Repository Adaptation

The results of the usage patterns monitoring techniques that we discussed in the previous paragraphs are also relevant to the repository evolution process. One of the important aspects of this process is the entity merging operation, which takes place when we discover that two entities in the repository represent the same real-world entity.

As we already mentioned, when merging entities, it is important to consider the type of values at hand, i.e., numerical vs. categorical with their corresponding measures. When the merging of the entities takes place off-line, we plan to employ standard techniques from clustering in order to group similar entities together. Although time may not be an issue in this case, we seek to perform the merging in an as efficient fashion as possible. Adopting techniques like BIRCH [32] for numerical data, and LIMBO for categorical data

[33], we may perform the assessment of similarity among entities very efficiently as these algorithms promise linear complexity as the size of the input increases.

The greatest advantage of employing techniques as the ones mentioned above is that we get to summarize the input data set, i.e. the OKKAM entities, in summaries that retain as much of the initial information as possible. We plan to extend the construction of summaries so that a) they can handle both numerical and categorical data at the same time, and b) can be used effectively with streaming data. The big challenge in this case is the maintenance of the summaries in that we will employ policies, either automatic or user-driven with the use of which we can "forget" summaries [4] of entities in the verge of extinction, or change the group (cluster) in which entities belong as their properties (attributes and attribute values) evolve in time.

5.4 Management of Access Control Policies

In current access control models, the complexity of management grows exponentially with the size and complexity of the contents. This is due to (i) the existence of different related artifacts such as roles, groups, which capture semantics in a "hidden" way; (ii) the possibility of creating conflicting policies, which in turn introduces the need for conflict resolution strategies; (iii) the direct relation between the location where contents are stored and the policies; (iv) the lack of dynamism and context-awareness in the AC model; and (v) the lack of policy validation tools. In this situation, when the size, dynamism and heterogeneity of the content set increases, it becomes impossible for administrators to understand the result of the policies they define, which inevitably leads to errors.

The OKKAM security model, and in particular the access control model, is based on the use of semantic modeling of the contents, policies and users. Semantic modeling brings important advances to access control because it facilitates the creation of easy to understand policies that can be automatically assigned to contents. Additionally, it facilitates interoperability of users' credentials and the development of mechanisms providing advanced access control features such as controlled delegation and user-retained control.

Among the traditional access control models, Role-Based Access Control (RBAC) is the most popular and has received much attention from researchers, but despite such effort, there are situations that are not well-handled by this model. In RBAC, a role must be defined for each different set of access criteria required by the group of resources controlled. This means that when the number of resources and the heterogeneity of access conditions are very high, the administration of RBAC systems becomes very complex.

A more general approach is needed for these new environments, and in particular for OKKAM repositories. The Semantic Access Control (SAC) model [30] provides the foundations for an appropriate solution to the aforementioned problems. Moreover, the flexibility of the SAC model allows it to easily simulate other models such as MAC, DAC

or RBAC. Although it represents a good foundation for OKKAM, SAC lacks several features that we need. We are currently working on the extension of the SAC model to support controlled delegation, provisional access and to incorporate privacy-preserving mechanisms like those provided by the Interactive Access Control model (iAccess) [31].

6. Conclusions

The web is quickly moving towards the direction of adding semantics to the online information, and using these semantics for enabling a vastly richer range of applications and user-experiences. A major step, necessary for achieving this goal, is to have a way for uniquely identifying entities in the emerging semantic web.

In this paper, we argue for an entity naming and management system, and we discuss the various problems that arise when considering the entity lifecycle management in this context. We examine the special requirements of representing entities, creating new and modifying existing entities, detecting duplicates, merging and versioning entities, and finally controlling the access to entities. Evidently, all the above functionalities are interrelated and affect each other. This work identifies these relationships, and presents the directions we are currently pursuing for realizing the functionalities outlined above.

References

[1] Fred Douglis, John Palmer, Elizabeth S. Richards, David Tao, William H. Tetzlaff, John M. Tracey, and Jian Yin, "Position: Short Object Lifetimes Require a Delete-Optimized Storage System," 11th ACM SIGOPS European Workshop, September 2004.

[2] Nigel Shadbolt, Tim Berners-Lee, Wendy Hall: The Semantic Web Revisited. IEEE Intelligent Systems 21(3): 96-101 (2006).

[3] Paolo Bouquet, Heiko Stoermer, and Daniel Giacomuzzi. OKKAM:. In Proceedings of the WWW2007 Workshop i3: Identity, Identifiers and Identification, Banff, Canada, May 8 2007, CEUR Workshop Proceedings, ISSN 1613-0073, May 2007.

[4] Themis Palpanas, Michail Vlachos, Eamonn J. Keogh, Dimitrios Gunopulos, Wagner Truppel: Online Amnesic Approximation of Streaming Time Series. ICDE 2004: 338-349.

[5] Themis Palpanas, Vana Kalogeraki, Dimitrios Gunopulos: Online Distribution Estimation for Streaming Data: Framework and Applications. SEBD 2007: 430-438.

[6] E. Pierre. Introduction to ILM: A tutorial. http://www.snia.org/, 2004.

[7] C. Johnson. ILM Case Study: Complete Data Lifecycle Management Solution. http://www.snia.org/, 2004.

[8] Soumyadeb Mitra, Marianne Winslett and Windsor Hsu: Query-based Partitioning of Documents and Indexes for Information Lifecycle Management. SIGMOD 2008.

[9] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, Vassilios S. Verykios: Duplicate Record Detection: A Survey. IEEE Trans. Knowl. Data Eng. 19(1): 1-16 (2007).

[10] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S.E. Whang, and J. Widom. Swoosh: A Generic Approach to Entity Resolution. To appear in VLDB Journal, 2008.

[11] S. Sarawagi, ed., special issue on data cleaning, IEEE Data Eng. Bull., vol. 23, no. 4, Dec. 2000.

[12] A. McCallum, Information Extraction: Distilling Structured Data from Unstructured Text, ACM Queue, vol. 3, no. 9, pp. 48-57, 2005.

[13] M.A. Jaro, "Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida," J. Am. Statistical Assoc., vol. 84, no. 406, pp. 414-420, June 1989.

[14] A.P. Dempster, N.M. Laird, and D.B. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," J. Royal Statistical Soc., vol. B, no. 39, pp. 1-38, 1977.

[15] W.E. Winkler, "Improved Decision Rules in the Felligi-Sunter Model of Record Linkage," Technical Report Statistical Research Report Series RR93/12, US Bureau of the Census, Washington, D.C., 1993.

[16] N. Bansal, A. Blum, and S. Chawla, "Correlation Clustering," Machine Learning, vol. 56, nos. 1-3, pp. 89-113, 2004.

[17] W.W. Cohen and J. Richman, "Learning to Match and Cluster Large High-Dimensional Data Sets for Data Integration," Proc. Eighth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '02), 2002.

[18] P. Singla and P. Domingos, "Multi-Relational Record Linkage," Proc. KDD-2004 Workshop Multi-Relational Data Mining, pp. 31-48, 2004.

[19] S. Sarawagi and A. Bhamidipaty, "Interactive Deduplication Using Active Learning," Proc. Eighth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '02), pp. 269-278, 2002.

[20] S. Tejada, C.A. Knoblock, and S. Minton, "Learning Domain- Independent String Transformation Weights for High Accuracy Object Identification," Proc. Eighth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '02), 2002.

[21] W.W. Cohen, “Data Integration Using Similarity Joins and a Word-Based Information Representation Language,” ACM Trans. Information Systems, vol. 18, no. 3, pp. 288-321, 2000.

[22] N. Koudas, A. Marathe, and D. Srivastava, “Flexible String Matching against Large Databases in Practice,” Proc. 30th Int’l Conf. Very Large Databases (VLDB ’04), pp. 1078-1086, 2004.

[23] D. Dey, S. Sarkar, and P. De, “Entity Matching in Heterogeneous Databases: A Distance Based Decision Model,” Proc. 31st Ann. Hawaii Int’l Conf. System Sciences (HICSS ’98), pp. 305-313, 1998.

[24] S. Guha, N. Koudas, A. Marathe, and D. Srivastava, “Merging the Results of Approximate Match Operations,” Proc. 30th Int’l Conf. Very Large Databases (VLDB ’04), pp. 636-647, 2004.

[25] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk, “Mining Database Structure; or, How to Build a Data Quality Browser,” Proc. 2002 ACM SIGMOD Int’l Conf. Management of Data (SIGMOD ’02), pp. 240-251, 2002.

[26] Dong, X., Halevy, A., and Madhavan, J. 2005. Reference reconciliation in complex information spaces. In Proceedings of the 2005 ACM SIGMOD international Conference on Management of Data (Baltimore, Maryland, June 14 - 16, 2005).

[27] E.-P. Lim, J. Srivastava, S. Prabhakar, and J. Richardson, “Entity Identification in Database Integration,” Proc. Ninth IEEE Int’l Conf. Data Eng. (ICDE ’93), pp. 294-301, 1993.

[28] M.A. Herna´ndez and S.J. Stolfo, “Real-World Data Is Dirty: Data Cleansing and the Merge/Purge Problem,” Data Mining and Knowledge Discovery, vol. 2, no. 1, pp. 9-37, Jan. 1998.

[29] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita, “Declarative Data Cleaning: Language, Model, and Algorithms,” Proc. 27th Int’l Conf. Very Large Databases (VLDB ’01), pp. 371-380, 2001.

[30] Yagüe, M.I., Maña, A., López, J., Troya, J.M.:Applying the Semantic Web Layers to Access Control. Proc. Int. Workshop on Web Semantics, Dexa 2003. IEEE Computer Society Press. September 2003.

[31] Koshutanski H. and Massacci F. A Negotiation Scheme for Access Rights Establishment in Autonomic Communication. Journal of Network and System Management (JNSM), Springer-Verlag press (to appear).

[32] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In SIGMOD, Montreal, QB, Canada, 4-6 June 1996.

[33] Periklis Andritsos, Panayiotis Tsaparas, Renee J. Miller, and Kenneth C. Sevcik. LIMBO: Scalable Clustering of Categorical Data. In EDBT, Heraklion, Greece, 14-18 March 2004.

[34] Wang Chiew Tan: Provenance in Databases: Past, Current, and Future. IEEE Data Eng. Bull. 30(4): 3-12, 2007.

[35] P. Buneman, S. Khanna,On Propagation of Deletions and Annotations through Views. T. PODS 2002.

[36] D. Bhagwat, L. Chiticariu, W. Tan, G. Vijayvargiya, An Annotation Management System for Relational Databases. VLDB Journal Vol. 14, No. 4, Nov. 2005.

[37] Douglas S. Santry, Michael J. Feeley, Norman C. Hutchinson, Alistair C. Veitch, Ross W. Carton, Jacob Ofir: Deciding when to forget in the Elephant file system. SOSP 1999: 110-123.

[38] Mallik Mahalingam, Chunqiang Tang, Zhichen Xu: Towards a Semantic, Deep Archival File System. FTDCS 2003: 115-121.

[39] Shu-Yao Chien, Vassilis J. Tsotras, Carlo Zaniolo: Efficient schemes for managing multiversionXML documents. VLDB J. 11(4): 332-353 (2002).

[40] Shu-Yao Chien, Vassilis J. Tsotras, Carlo Zaniolo, Donghui Zhang: Supporting complex queries on multiversion XML documents. ACM Trans. Internet Techn. 6(1): 53-84 (2006).

[41] Ferry Irawan Tantono, Nishad Manerikar, Themis Palpanas: Efficiently Discovering Recent Frequent Items In Data Streams. SSDBM (2008).