User Feedback as a First Class Citizen in Information Integration Systems*

Khalid Belhajjame
School of Computer Science
University of Manchester
Manchester, UK
khalidb@cs.man.ac.uk

Norman W. Paton
School of Computer Science
University of Manchester
Manchester, UK
norm@cs.man.ac.uk

Alvaro A. A. Fernandes
School of Computer Science
University of Manchester
Manchester, UK
afernandes@cs.man.ac.uk

Cornelia Hedeler
School of Computer Science
University of Manchester
Manchester, UK
chedeler@cs.man.ac.uk

Suzanne M. Embury
School of Computer Science
University of Manchester
Manchester, UK
sembury@cs.man.ac.uk

ABSTRACT

User feedback is gaining momentum as a means of addressing the difficulties underlying information integration tasks. It can be used to assist users in building information integration systems and to improve the quality of existing systems, e.g., in dataspaces. Existing proposals in the area are confined to specific integration sub-problems considering a specific kind of feedback sought, in most cases, from a single user. We argue in this paper that, in order to maximize the benefits that can be drawn from user feedback, it should be considered and managed as a first class citizen. Accordingly, we present generic operations that underpin the management of feedback within information integration systems, and that are applicable to feedback of different kinds, potentially supplied by multiple users with different expectations. We present preliminary solutions that can be adopted for realizing such operations, and sketch a research agenda for the information integration community.

Keywords

User feedback, Feedback management, Information integration, Dataspaces

1. INTRODUCTION

Two decades of extensive research and real world experience suggest that building information integration systems is a difficult task [11]. Essentially, the difficulty lies in understanding user requirements as to what the relevant data sources are, and the way the contents of the sources are to be combined and structured in a form that is compatible with the user's conceptualization of the world.

To facilitate information integration, some database researchers have explored the use of feedback solicited from users. User feedback is a growing theme in information integration systems, as reflected in the number of recent proposals that utilise user feedback to guide the construction of information integration systems, and/or to improve the quality of the services they provide [1, 2, 4, 5, 13, 15, 17, 18]. For example, Talkudar *et al.* [17, 18] developed the `Q` system to assist users in creating integration queries. In doing so, the system solicits feedback on results of candidate integration queries. Feedback is also fundamental to the dataspaces vision [9], which aims to reduce the cost of setting up a data integration system, and to gradually improve the quality of the resulting system. The idea is to enlist users to provide feedback with a view to improving the quality of the services supplied by the data integration system. For example, Jeffery *et al.* [13] developed a decision-theoretic framework for specifying the order in which feedback can be solicited on a collection of schema mappings with the objective of providing the *most benefit* to a dataspace. Also, we have shown in previous work [2], that schema mappings can be selected, refined and annotated with metrics specifying their fitness to user requirements based on feedback commenting on the membership of tuples to relations in the integration schema. More recently, Doan *et al.* [7] proposed a taxonomy that classifies mass collaboration systems, namely systems that enlist a multitude of human users to help solve a given problem. In doing so, they identified issues with respect to user inputs (feedback), e.g., how to recruit and retain users, and how to evaluate their inputs.

While existing proposals showcase the key role user feedback can play within information integration systems, they are confined to the use of feedback given on a specific artifact, e.g., query [4], query result [2, 17], or mapping [1], to tackle a specific information integration sub-problem, e.g., designing a mapping [1], selecting a mapping [2], or specifying a query [18, 17]. In doing so, they do not explore the benefits that can be drawn from using feedback given on different kinds of artifacts to leverage multiple informa-

*The work reported in this paper was supported by a grant from the EPSRC.

5^{th} *Biennial Conference on Innovative Data Systems Research (CIDR '11)* January 9-12, 2011, Asilomar, California, USA.

tion integration tasks. Moreover, they make assumptions that do not necessarily hold in practice, and that if dropped may lead to costs and losses that outweigh the benefits that can be drawn from user feedback. For example, they assume that a data integration system either has a single user (e.g., [18]), or that all the users supplying feedback have the same requirements (e.g., [13]). In practice, different users may have different expectations, in which case, the quality of the services provided by the data integration system may be seen as deteriorating in response to user feedback, at least for some users, over time, due to conflicts in requirements. Also, existing proposals assume that user requirements do not change over time. This assumption may not hold since user needs may shift, in which case previously acquired feedback may prevent the improvement of the data integration system with a view to answering new user needs.

We argue that, in order to be able to exploit different kinds of feedback provided by multiple users to leverage different information integration tasks, user feedback should be considered and managed as a first class citizen. Accordingly, we present in this paper a set of feedback management research challenges that together aim to foster the semantic cohesion of the feedback provided by users, and to increase the value and the benefits that can be drawn from acquired feedback. We propose preliminary solutions to the research challenges identified, and sketch a research agenda for the information integration community.

The paper is structured as follows. We begin by presenting a general feedback model that caters for the specification of the different kinds of feedback that have been considered in the information integration literature in Section 2. We then present operations that can be used for fostering the cohesion of the feedback provided by users by detecting inconsistencies in feedback (Section 3), and by checking the validity of feedback over time (Section 4). We go on to present operations that aim to increase the value and the benefits derived from user feedback. Specifically, we present a clustering operation for grouping users with similar expectations (in Section 5), and an operation for learning new feedback using collaborative filtering techniques (in Section 6). We close the paper in Section 7 by underlining our main contributions.

2. WHAT IS FEEDBACK?

In essence, feedback can be seen as annotations that a user provides to comment on artifacts of an information integration system, be they matches, mappings, integration schema, queries or query results, with the objective of informing the construction of the system and/or improving the quality of the services it provides. We define a feedback instance by the tuple:

$$\langle \texttt{obj}, \texttt{t}, \texttt{u}, \texttt{k} \rangle$$

specifying that the user `u` annotates the artifact `obj` using the term `t`. The user can provide different kinds of feedback. `k` indicates the kind of feedback. The set of terms that can be used depends on the kind of feedback the user wishes to provide. They can belong to controlled vocabularies or domain ontologies. In what follows, we use `uf.obj`, `uf.annot`, `uf.user` and `uf.type` to refer to `obj`, `t`, `u` and `k`, respectively.

The above feedback model is general, in that it can be used to describe the different kinds of feedback defined in the information integration literature. Table 1 presents prominent proposals in the field that use feedback, and shows how the feedback used in each proposal can be described using the model presented above.

We present in what follows some examples of feedback defined in the literature and show how they can be mapped to the above model.

Example 2.1. Consider the `Q` system developed by Talkudar *et al.* [18]. This system assists users in creating integration queries that span multiple data sources. To learn user preference as to which query captures expectations, the user supplies feedback by ordering result tuples returned by different alternative queries. For example, consider two tuples $\texttt{t}_1$ and $\texttt{t}_2$ retrieved by the alternative queries $\texttt{q}_1$ and $\texttt{q}_2$, respectively. If the user judges that $\texttt{t}_1$ meets the expectations better than $\texttt{t}_2$, then $\texttt{t}_1$ is ranked before $\texttt{t}_2$. Using our model, such feedback can be specified as follows: $\langle\langle \texttt{t}_1, \texttt{t}_2 \rangle, \texttt{before}, \texttt{u}, \texttt{ranking} \rangle$. The vocabulary used for annotation, in this case, is composed of two terms `before` and `after`.

Example 2.2. McCann *et al.* [15] developed a system that informs schema matching by soliciting feedback from users. As an example, consider a binary match $\langle \texttt{r}_1, \texttt{r}_2 \rangle$ that associates the relations $\texttt{r}_1$ and $\texttt{r}_2$, and consider that the user `Anhai` specified that such a match is incorrect. Using our model, such feedback can be specified as follows: $\langle\langle \texttt{r}_1, \texttt{r}_2 \rangle, \texttt{fp}, \texttt{Anhai}, \texttt{match\_correctness} \rangle$. Using the same system, users can provide feedback that specifies (or confirms) the data type of a given attribute. For example, the same user can confirm that the attribute `fecha` is of type `date`, which can be described using our feedback model as follows: $\langle \texttt{fecha}, \texttt{date}, \texttt{Anhai}, \texttt{type\_correctness} \rangle$.

Example 2.3. We showed in a previous work [2] that feedback can be used to annotate schema mappings with estimates specifying the degree to which they meet user expectations. In doing so, a user annotates result tuples that are retrieved using candidate mappings to populate a given relation in the integration schema as true positives, false positives or false negatives. Consider a relation `r` in the integration schema, a mapping `m` used to populate `r` using data from the sources, and a tuple `t` of `r` that is retrieved using `m`. The following feedback instance, $\langle\langle \texttt{r}, \texttt{m}, \texttt{t} \rangle, \texttt{tp}, \texttt{Norman}, \texttt{tuple\_membership} \rangle$, specifies that `t` is a true positive, i.e., that `t` is a member of `r` according to the expectations of the user `Norman`. Using feedback instances of the above form, candidate mappings for populating the elements of an integration schema can be annotated, selected and refined [2].

Having illustrated how the model presented in this section can be used to describe different kinds of feedback, we now present operations that act on feedback instances of that model with the goal of determining the validity of the feedback instances provided by users, and to increase the benefits that can be derived from their use.

3. FEEDBACK INCONSISTENCY

Different users may have different requirements. Moreover, the requirements of the same user may change over time.

Table 1: Kinds of feedback considered in the information integration literature.

Proposal	Objects on which feedback is given	Set of terms used for annotating objects
Alexe *et al.* [1]	an instance of a given schema and the instance obtained by its transformation into another schema	{'yes', 'no'} // used to comment on schema transformation
Belhajjame *et al.* [2]	a result tuple	{'true positive', 'false positive', 'false negative'}
	an attribute and its value	{'true positive', 'false positive', 'false negative'}
Cao *et al.* [4]	a candidate query	{'true positive', 'false positive'}
	a pair of candidate queries	{'before', 'after'} // used for ordering queries
Chai *et al.* [5]	a view result tuple	{'insert', 'delete', 'update'}
Jeffery *et al.* [13]	a mapping	{'true positive', 'false positive'}
McCann *et al.* [15]	a relation attribute	set of attribute data types
	two attributes of a given relation	set of constraints
	a match	{'true positive', 'false positive'}
Talkudar *et al.* [18]	a result tuple	{'true positive', 'false positive'}
	a pair of result tuples	{'before', 'after'} // used for ordering results

In this section, we present an approach to detecting these phenomena, viz. feedback inconsistency. The basic idea is that if two users have different requirements, then this difference may give rise to conflicts between the feedback instances they supply. Similarly, if the requirements of a given user changes over time, then this change may give rise to inconsistency between feedback instances supplied at different points in time. Given two feedback instances $\texttt{uf}_1$ and $\texttt{uf}_2$, we present rules that can be used to check whether $\texttt{uf}_1$ and $\texttt{uf}_2$ are inconsistent.

We start by considering the case in which $\texttt{uf}_1$ and $\texttt{uf}_2$ annotate the same object. $\texttt{uf}_1$ and $\texttt{uf}_2$ are inconsistent if they are of the same kind, and annotate the same object using conflicting terms. Below is a rule that detects this type of inconsistency.

```
1 inconsistent(uf1, uf2) : −
2   - uf1 and uf2 are of the same kind
3   uf1.type = uf2.type,
4   - uf1 and uf2 label the same object
5   uf1.obj = uf2.obj,
6   - using inconsistent annotation
7    uf1.type.conflictAnnotations(uf1.annot, uf2.annot)
```

Notice that the notion of inconsistency depends on the kinds of feedback *(line 7)*; in this respect, the function `conflictAnnotations()` acts as an extensibility point. As a proof of concept, we present below two definitions of this function for two different kinds of feedback.

Example 3.1. Consider for example that feedback annotates query tuples as true positives or false positives (e.g., [2]). In this case, two terms are conflicting if they are different. That is:

```
1 conflictAnnotations(c1, c2) : −
2   - The terms c1 and c2 are different
3    c1 ≠ c2
```

Example 3.2. Consider now that the feedback provided by users uses terms taken from domain ontologies (e.g., [15]). In this case, two concepts are conflicting if they are disjoint. That is:

```
1 conflictAnnotations(c1, c2) : −
2   - The data types denoted by c1 and c2
3   - are disjoint
4    c1 ⊓ c2 = ⊥
```

The artifacts that constitute an information integration system can be dependent on each other. Such dependencies may give rise to constraints between feedback, more specifically between the terms used to annotate dependent artifacts. If such constraints are not satisfied, then this results in inconsistency between the feedback instances in question. This form of inconsistency can be detected using the following rule:

```
1 inconsistent(uf1, uf2) : −
2   - uf1 and uf2 are of the same kind
3   uf1.type = uf2.type,
4   - there is a pair of dependency and constraint
5   - ⟨dep_type, const_type⟩ that are associated with
6   - the kind of feedback uf1.type
7   ∃ ⟨dep_type, const_type⟩ ∈ uf1.type.getDepConstPair(),
8   - the objects annotated by uf1 and uf2 are
9   - related using the dep_type dependency
10  dependent(uf1.obj, uf2.obj, dep_type),
11  - the annotations assigned by the uf1 and uf2
12  - do not satisfy the const_type constraint
13  ¬constraint(uf1.annot, uf2.annot, const_type)
```

where getDepConstPair() is a function that return pairs of dependency type and constraint type that are associated with a given feedback type, $\texttt{dependent}(\texttt{uf}_1.\texttt{obj}, \texttt{uf}_2.\texttt{obj}, \texttt{dep}_{\texttt{type}})$ is a predicate that is true if the objects $\texttt{uf}_1.\texttt{obj}$ and $\texttt{uf}_2.\texttt{obj}$ are related by a dependency of type $\texttt{dep}_{\texttt{type}}$, and $\texttt{constraint}(\texttt{uf}_1.\texttt{annot}, \texttt{uf}_2.\texttt{annot}, \texttt{const}_{\texttt{type}})$ is a predicate that is true if the constraint of type $\texttt{const}_{\texttt{type}}$ holds between the annotations $\texttt{uf}_1.\texttt{annot}$ and $\texttt{uf}_2.\texttt{annot}$.

Example 3.3. As an example, consider the following feedback instances which annotate values of relation attributes as true positives or false positives:

- $\texttt{uf} = \langle\langle \texttt{r}.\texttt{att}, \texttt{v}\rangle, \texttt{tp}, \texttt{Norman}, \texttt{value\_membership}\rangle$ specifies that the value v of the attribute att of the r relation is a true positive, and
- $\texttt{uf}' = \langle\langle \texttt{r}'.\texttt{att}', \texttt{v}\rangle, \texttt{fp}, \texttt{Norman}, \texttt{value\_membership}\rangle$ specifies that the value v of the attribute $\texttt{att}'$ of the $\texttt{r}'$ relation is a false positive.

Additionally, consider that there is a foreign key that links the attribute att of r to the attribute $\texttt{att}'$ of $\texttt{r}'$. Given this, we can deduce that uf and $\texttt{uf}'$ are inconsistent. Indeed, if v is a true positive for the attribute att of r, then it should also be a true positive for the attribute $\texttt{att}'$ of $\texttt{r}'$ given the inclusion constraint implied by the foreign key. The predicates dependent and constraint used for detecting inconsistencies of the above form can be defined as follows:

- $\texttt{dependent}(\texttt{uf}_1.\texttt{obj}, \texttt{uf}_2.\texttt{obj}, \text{`}\texttt{foreign\_key}'\text{)}$ is true if the attribute in $\texttt{uf}_1.\texttt{obj}$ is linked to the attribute in $\texttt{uf}_2.\texttt{obj}$ using a foreign key, and the attribute values in $\texttt{uf}_1.\texttt{obj}$ and $\texttt{uf}_2.\texttt{obj}$ are the same, and is false otherwise.
- $\texttt{constraint}(\texttt{uf}_1.\texttt{annot}, \texttt{uf}_2.\texttt{annot}, \text{`}\texttt{tp\_inclusion}'\text{)}$ is false if $\texttt{uf}_1.\texttt{annot}$ takes the value tp and $\texttt{uf}_2.\texttt{annot}$ takes the value fp, and is true otherwise. That is:

```
1 constraint(annot1, annot1, 'tp_inclusion') : −
2   ¬((annot1 = tp), (annot2 = fp)),
```

4. FEEDBACK VALIDITY

User requirements may change over time in which case previously acquired feedback may become invalid. In this section, we specify the situations under which a feedback instance can be stated to be valid or invalid.

We use the event calculus [14] as a formalism to define the rules used to check the validity of user feedback. The event calculus is a logic-based formalism that can be used to deduce the state of a given domain based on the events (actions) that have taken place. In the case of our analysis, the main event is the fact that a user supplies a feedback instance uf. We denote this event by supplyFeedback(uf). In addition we consider the following predicates:

- exists(obj) is true if the object obj exists, and is false, otherwise.
- valid(uf) is true if the feedback instance uf is valid, and is false, otherwise.
- invalid(uf) is true if the feedback instance uf is invalid, and is false, otherwise.
- hasUnkownStatus(uf) is true if the status of uf is unknown, and is false if uf is either valid or invalid.

Notice that the above predicates, exists(obj), valid(uf), invalid(uf) and hasUnkownStatus(uf), are fluent: their value is subject to change over time. As well as the above predicates, we consider two predicates that are specific to the event calculus, viz. holdsAt and happens. holdsAt(fl, t) and happens(e, t) are used to check that a given fluent fl holds at a given time point t and to check that a given event e occurs at a given time point t, respectively. We will now specify the cases in which a feedback instance is valid, invalid or has unknown status.

We start by specifying the conditions under which a feedback instance $\texttt{uf}_1$ is invalid. $\texttt{uf}_1$ is invalid if there is a valid feedback instance $\texttt{uf}_2$ that is conflicting with and fresher than $\texttt{uf}_1$. This is because we consider that inconsistencies in feedback emerge from changes in requirements, in which case, $\texttt{uf}_1$ reflects past requirements that have changed, and as a result, is no longer valid. We formally define the rule for identifying invalid feedback instances as follows:

```
1 holdsAt(invalid(uf1), t) : −
2   - uf1 was supplied by the user at a time point
3   - that is equal to or before t
4   happens(supplyFeedback(uf1), t1), t1 ≤ t,
5   - There exists a feedback instance uf2 that was
6   - supplied by the user at a time point that
7   - is equal to or less than t, and greater than t1
8   happens(supplyFeedback(uf2), t2), t1 < t2 ≤ t,
9   - uf2 is inconsistent with uf1,
10  - and is known to be valid at t
11  inconsistent(uf1, uf2), holdsAt(valid(uf2), t)
```

We now specify the conditions under which a feedback instance $\texttt{uf}_1$ is valid. $\texttt{uf}_1$ is valid at the time point t if it was supplied by the user before t, there is no conflicting feedback instance $\texttt{uf}_2$ that is fresher than $\texttt{uf}_1$ and valid at t, and the object $\texttt{uf}_1.\texttt{obj}$ on which feedback is given exists at t. We do not consider feedback that was given on an object that is no longer available. This is because the non existence of the object in question may in certain cases mean that the annotation given by the feedback is no longer valid. To illustrate this, consider the following relational table car(model, manufacturer), and consider that the user gave a feedback instance uf annotating the tuple $\langle\texttt{Mini}, \texttt{Rover}\rangle$ as a true positive of the car relation. The Mini manufacturer changed later on, and the tuple was updated to $\texttt{car} = \langle\texttt{Mini}, \texttt{BMW}\rangle$. As a result of this update, the feedback instance uf given previously is no longer valid. We formally define the rule for identifying valid feedback instances as follows:

```
1 holdsAt(valid(uf1), t) : −
```

```
2   - uf1 was supplied by the user at a time point
3   - that is equal to or before t
4   happens(supplyFeedback(uf1), t1), t1 ≤ t,
5   - There is no valid feedback instance uf2 that
6   - is conflicting with and fresher than uf1
7   not (happens(supplyFeedback(uf2), t2),
8        inconsistent(uf1, uf2),
9        holdsAt(valid(uf2), t)), t1 < t2 ≤ t),
10  - The object that uf1 annotates exists at t
11  holdsAt(exists(uf1.obj), t).
```

Notice that the above rule is recursive. Specifically, the validity of the feedback instance $\texttt{uf}_1$ is preconditioned by the non existence of a valid feedback instance $\texttt{uf}_2$ that is conflicting with and fresher than $\texttt{uf}_1$.

There are situations in which we are unable to determine whether a feedback instance is valid or invalid. This is the case, for example, when the object on which feedback was given, is no longer available. This is also the case if the user supplies two inconsistent feedback instances at the same time. (The latter case may be indicative of malicious behaviour of the user [15].) The status of a feedback instance $\texttt{uf}_1$ is unknown if it is neither valid nor invalid. That is:

```
1 holdsAt(hasUnkownStatus(uf1), t) : −
2   - uf1 was supplied by the user before t
3   happens(supplyFeedback(uf1), t1), t1 ≤ t,
4   - uf1 is not known to be valid at t
5   not holdsAt(valid(uf1), t),
6   - uf1 is not known to be invalid at t
7   not holdsAt(invalid(uf1), t).
```

Note that the above analysis on validity of user feedback can be applied to feedback instances that are supplied by the same user, or by a group of users that are known to have the same requirements. This raises the question as to how to determine whether a group of users have the same requirements. In the following section, we show how clustering techniques can be used for this purpose.

5. CLUSTERING USERS BASED ON FEEDBACK

The users of an information integration system should have the same (or similar) requirements to ensure the cohesion of the feedback they provide. Checking the consistency of the feedback provided by different users can be used as a means for ensuring such cohesion. In this section, we explore another technique that can be used for this purpose, namely clustering.

Clustering is unsupervised classification [12]. We use this technique to group users according to their requirements. Specifically, given a set of users and their (partial) requirements, elicited through feedback, clustering is used to identify groups of users with similar requirements. Clustering presents the following potential benefits:

- The feedback provided by the users within a cluster can be used collectively, thereby improving the quality of the integration system.
- Clustering may reveal that the users of an existing information system have different requirements, and therefore point out the need to create multiple information integration systems to replace the existing one.
- Conversely, clustering may identify opportunities for grouping the users of two or more information integration systems into a single group, if it turns out that they have similar requirements.
- Clustering can also be used to identify outlier users within an information integration system, i.e., users that have different requirements from the rest of the users.

As a proof of concept, we present in what follows an example illustrating how users can be clustered according to their requirements.

Example 5.1. Consider a set of users who would like to have information about available proteins. Specifically, there is a `Protein` relation in the integration schema that users wish to query. Not all users have the same requirements, e.g., they may be interested in proteins belonging to different species or with different structures, motifs, etc. Clustering can be used to identify groups of users with similar requirements as to which tuples should populate the `Protein` relation. There are many algorithms for clustering that are readily available in the literature [12]. To make use of such algorithms, however, we need to choose a pattern representation for specifying user requirements, and a metric for measuring the distance between user requirements. In what follows, we present a pattern representation and a distance metric that are suitable for the above clustering problem.

Pattern Representation. We specify a user's requirements with respect to the extent of the `Protein` relation using a pair $\langle \texttt{E}, \texttt{UE} \rangle$, where $\texttt{E}$ is the set of tuples that are expected by the user, i.e., the tuples that the user annotated as true positives or false negatives, and $\texttt{UE}$ is the set of tuples that are not expected by the user, i.e., the tuples that the user annotated as false positives.

Distance metric. To define the distance between requirements, we start by defining the notion of similarity of requirements. Consider two users $\texttt{u}_1$ and $\texttt{u}_2$. The larger the overlap between the expected tuples of $\texttt{u}_1$ and $\texttt{u}_2$, i.e., $\texttt{E}_1$ and $\texttt{E}_2$, and the larger the overlap between their unexpected tuples, $\texttt{UE}_1$ and $\texttt{UE}_2$, the more similar are their requirements are. A similarity measure that captures this property can be defined as follows:

$$\texttt{sim}(\texttt{u}_1, \texttt{u}_2) = \texttt{w}_\texttt{e} \times \frac{|\texttt{E}_1 \cap \texttt{E}_2|}{|\texttt{E}_1 \cup \texttt{E}_2|} + \texttt{w}_\texttt{ue} \times \frac{|\texttt{UE}_1 \cap \texttt{UE}_2|}{|\texttt{UE}_1 \cup \texttt{UE}_2|}$$

where $\texttt{w}_\texttt{e}$ and $\texttt{w}_\texttt{ue}$ are weights such that $\texttt{w}_\texttt{e} + \texttt{w}_\texttt{ue} = 1$. The ratios $\frac{|E_1 \cap E_2|}{|E_1 \cup E_2|}$ and $\frac{|UE_1 \cap UE_2|}{|UE_1 \cup UE_2|}$ are known as the Jaccard

coefficient [3]. The above similarity measure takes values between 0 and 1, and the corresponding distance metric can be defined as follows: $\mathtt{dis}(u_1, u_2) = 1 - \mathtt{sim}(u_1, u_2)$.

Using the representation and distance function defined above, we can group users of the Protein relation using existing clustering algorithms [3]. In what follows, we present preliminary empirical results with synthetic data show how such users can be clustered into groups using hierarchical agglomerative clustering. Specifically, we consider the following setting. A set of 20 users, each of which provides feedback instances identifying expected and unexpected tuples of the Protein relation. Regarding the distance metric, we consider that a feedback instance specifying an expected tuple has the same weight as a feedback instance specifying an unexpected tuple, i.e., $w_e = w_{ue} = 0.5$. To cluster users, we use the *hclust* algorithm provided by the R statistical framework[1] for performing hierarchical clustering. Specifically, we run the following procedure:

```
1 For each user u_i
2    Generate a set of feedback instances UF_{u_i}
3 For each pair of users ⟨u_i, u_j⟩
4    Compute the distance between them
5    Log that distance in the distance matrix DM
6 Invoke the hclust program using as input DM
```

DM is a two-dimensional array that is used to log the distances, taken pairwise, of a set of users.

We run the above procedure by varying the size of overlap in feedback instances between users. Specifically, we considered the following three scenarios:

1. The overlap in feedback instances between users is small: less than 5 percent of the feedback instances supplied by a given user overlap with the feedback instances supplied by another user.
2. The overlap in feedback instances between users is significant: 10 to 20 percent of the feedback instances that are provided by a given user overlap with the feedback instances supplied by another user.
3. The overlap in feedback instances between users is large: 20 to 50 percent of the feedback instances that are provided by a given user overlap with the feedback instances supplied by another user.

Figures 1, 2 and 3 visualize the clustering results obtained in each of the above scenarios using dendrograms. The users, which are identified by integers, are listed along the x-axis in an order that is convenient for showing the cluster structure. The y-axis measures inter-cluster distance. Consider, for example, the users identified by the integers 13 and 17 in Figure 3. They are joined into the same cluster. The distance between them is 0.1, and no other users are separated by a distance smaller than that value. Consider Figures 1, 2 and 3 together. As expected, the comparison of three dendrograms shows that the smaller the overlap between users in terms of feedback, the poorer the quality of the clustering. Indeed, the intra-cluster distance is large when overlap in feedback is small (Figure 1). A cluster, therefore, may contain users with requirements that are significantly different. On the other hand, the quality of clustering is better when overlap in feedback is large. The intra-cluster distance is reduced (Figure 3), thereby yielding clusters that join users with similar requirements.

[1] http://www.r-project.org

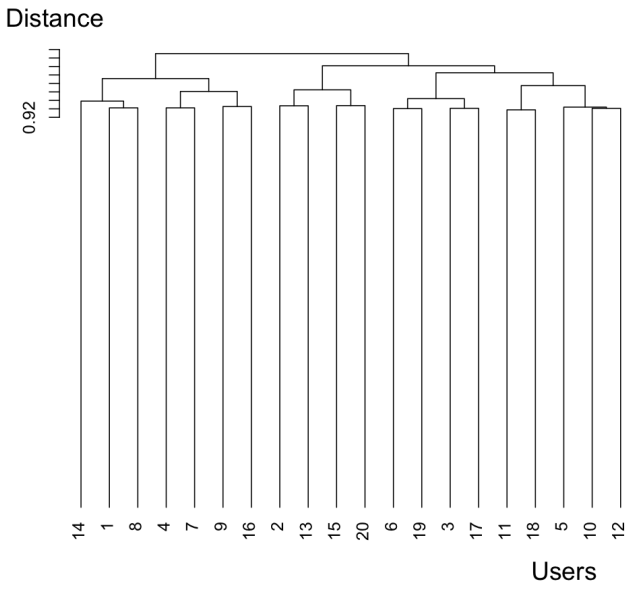

Figure 1: Clustering results when overlap in user feedback is small

The above example considers feedback instances of the same kind. In practice, different kinds of feedback instances given on different kinds of artifacts will be used to elicit user requirements. We can use feedback propagation as a mechanism to deal with the issue. The basic idea can be formulated as follows: given a feedback instance uf annotating an object of a given type, e.g., a mapping m, derive a feedback instance $\mathtt{uf}'$ annotating an object of another type, e.g., a tuple t that is produced using the mapping m. Propagating feedback may in certain situations be used for aggregating feedback instances of different types.

Aggregation is also another means that can be explored when feedback instances are of different kinds. As an example, assume that we want to cluster users based on feedback instances of different kinds. Using the approach described above, we obtain a set of clusterings $\{C_1, \ldots, C_n\}$; C_i being the clustering obtained using feedback instances of the ith type. The obtained clusterings can then be aggregated to produce a single clustering C that agrees as much as possible with the n clusterings [10]. We can do so, for example, by creating a graph G, the vertices of which represent users, and where the edges are weighted using the information provided by the clusterings $C_1, \ldots, C_n$. Specifically, the weight of the edge $\langle u_1, u_2 \rangle$ linking the users u_1 and u_2 is the fraction

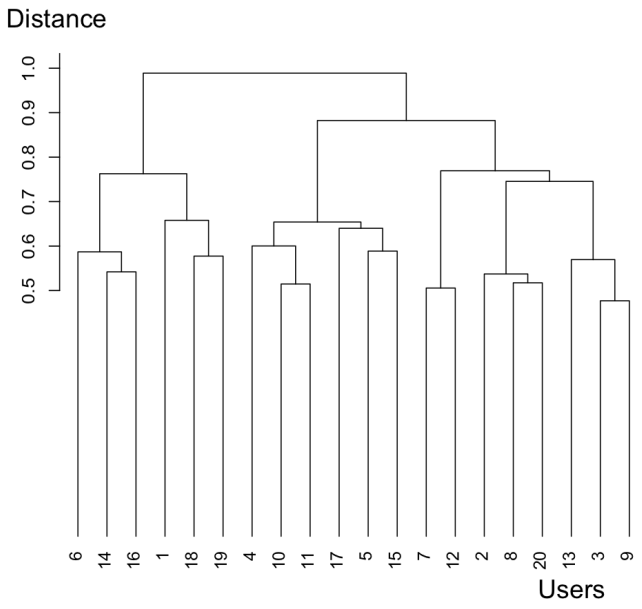

Figure 2: Clustering results when overlap in user feedback is important

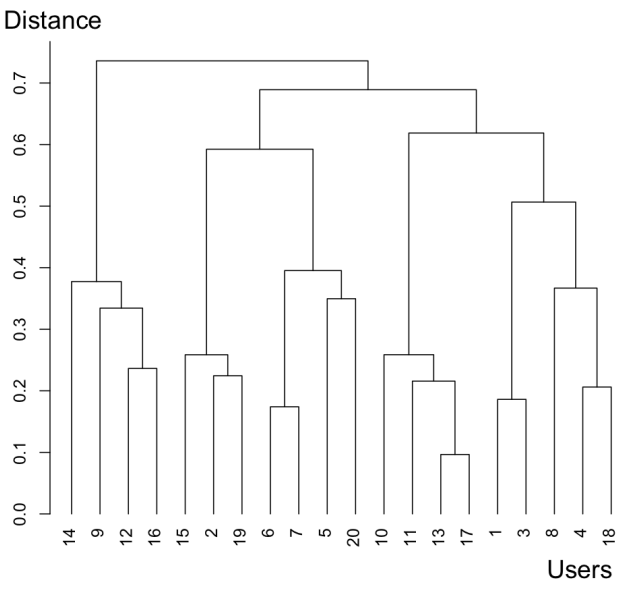

Figure 3: Clustering results when overlap in user feedback is large

of clusterings in which the two users are placed in different clusters. The cluster `C` is then obtained using a partitioning of the graph `G` that cuts as few as possible of the edges with small weight, e.g., less than `0.5`, and cuts as many as possible of the edges with large weight, e.g., more than `0.5`. Multidimensional clustering algotithms, e.g., PCA, is another means that can be explored for grouping users based on feedback instances of different types [3], e.g., each dimension can be used to encode the user requirements that are derived from feedback instances of a given kind.

As mentioned at the beginning of this section, clustering can be used as a means for reducing the workload of individual users in terms of the feedback they provide. We present in the following section a technique that can be used for automatically generating user feedback.

6. LEARNING FEEDBACK USING COLLABORATIVE FILTERING

There are situations in which it is desirable to learn the requirements of a given user. For example, a user may have only partial knowledge of requirements, in which case, s/he may not be in a position to provide feedback on given artifacts. Also, if a new user joins an information integration system, then we may want to accelerate the requirement acquisition process. In this section, we explore the use of collaborative filtering [16], a technology that is used for learning user preferences, to learn user feedback. Specifically, the problem that we tackle can be formulated as follows. Given a user `u` and an object `obj` of an information integration system, predict the feedback `uf` that `u` would provide to annotate `obj` based on other users' feedback. The underlying assumption to our approach is that if two users have similar requirements, then they are likely to provide the same feedback on a given artifact.

Example 6.1. Assume, for example, that we would like to know whether a given tuple `t` can be used to populate a relation `r` of the integration schema in the conceptualization of the user `u`. We can learn the feedback the user `u` would provide to annotate `t` by using the feedback instances that other users provided to annotate that tuple. Assume that the users $\mathtt{u}_1, \ldots, \mathtt{u}_n$ provided feedback instances annotating `t`. The following formula computes a score, $\mathtt{expected}(\mathtt{t}, \mathtt{u})$, estimating the likelihood that `t` is expected and a score, $\mathtt{unexpected}(\mathtt{t}, \mathtt{u})$, estimating the likelihood that `t` is unexpected for `u`.

$$\mathtt{expected}(\mathtt{t},\mathtt{u}) = \frac{\sum_{\mathtt{i}=1}^{\mathtt{i}=\mathtt{n}}(\mathtt{sim}(\mathtt{u},\mathtt{u_i}) \times \mathtt{isExp}(\mathtt{u_i},\mathtt{t}))}{n}$$

$$\mathtt{unexpected}(\mathtt{t},\mathtt{u}) = \frac{\sum_{\mathtt{i}=1}^{\mathtt{i}=\mathtt{n}}(\mathtt{sim}(\mathtt{u},\mathtt{u_i}) \times \mathtt{isUnexp}(\mathtt{u_i},\mathtt{t}))}{n}$$

$\mathtt{expected}(\mathtt{t},\mathtt{u})$ and $\mathtt{unexpected}(\mathtt{t},\mathtt{u})$ are calculated as weighted averages of the feedback given by other users where the weights are the similarity scores between user requirements. `sim`() is the similarity measure defined in the previous section. $\mathtt{isExp}(\mathtt{u_i},\mathtt{t})$ (resp. $\mathtt{isUnexp}(\mathtt{u_i},\mathtt{t})$) is a boolean predicate that is true if the tuple `t` is expected (resp. unexpected) in the conceptualization of the user $\mathtt{u_i}$.

The operations that we presented for detecting inconsistencies in feedback, clustering users and learning feedback operate on feedback given on common objects. Therefore, the results they produce may not be trustworthy when the set of objects annotated by more than one feedback instance is small. To improve the quality of the results produced by the operations presented in this paper, we can take inspiration from model-based collaborative filtering algorithms [6]. Rather than making predictions based on items that are

commonly annotated by users, model-based collaborative filtering algorithms employ user annotations to derive, for each user, a model characterizing his/her preferences. Predictions are then made by matching those models. We can adopt a similar approach. The issue here is, of course, which model to adopt for synthesizing user requirements elicited through feedback.

As an example, in our previous work [2], users provide feedback commenting on the membership of tuples to a relation in the integration schema. If the set of tuples annotated by more than one user is small then the operation we presented for learning feedback may not be able to generate meaningful feedback.

Feedback that comments on tuple membership was used in [2] to annotate schema mappings with metrics estimating their precision and recall. Specifically, given $\mathtt{R}$, a relational table, and $\mathtt{m}$, a mapping that is a candidate to populate $\mathtt{R}$, the precision of $\mathtt{m}$ given the feedback $\mathtt{UF}_{\mathtt{u_i}}$ supplied by the user $\mathtt{u_i}$ is defined as the ratio of the number of true positive tuples returned by $\mathtt{m}$ to the sum of the number of true positive tuples and the number of false positive tuples of $\mathtt{m}$ given the feedback instances in $\mathtt{UF}_{\mathtt{u_i}}$. Similarly, the recall of $\mathtt{m}$ is defined as the ratio of the number of true positive tuples returned by $\mathtt{m}$ to the sum of the number of true positive tuples and the number of false negative tuples of $\mathtt{m}$ given $\mathtt{UF}_{\mathtt{u_i}}$. That is:

$$\mathtt{Precision}(\mathtt{m}, \mathtt{UF}_{\mathtt{u_i}}) = \frac{|\mathtt{tp}(\mathtt{m}, \mathtt{UF}_{\mathtt{u_i}})|}{|\mathtt{tp}(\mathtt{m}, \mathtt{UF}_{\mathtt{u_i}})| + |\mathtt{fp}(\mathtt{m}, \mathtt{UF}_{\mathtt{u_i}})|}$$

$$\mathtt{Recall}(\mathtt{m}, \mathtt{UF}_{\mathtt{u_i}}) = \frac{|\mathtt{tp}(\mathtt{m}, \mathtt{UF}_{\mathtt{u_i}})|}{|\mathtt{tp}(\mathtt{m}, \mathtt{UF}_{\mathtt{u_i}})| + |\mathtt{fn}(\mathtt{m}, \mathtt{UF}_{\mathtt{u_i}})|}$$

where $|\mathtt{s}|$ denotes the magnitude of the set s, and $\mathtt{tp}(\mathtt{m}, \mathtt{UF}_{\mathtt{u_i}})$, $\mathtt{fp}(\mathtt{m}, \mathtt{UF}_{\mathtt{u_i}})$ and $\mathtt{tp}(\mathtt{m}, \mathtt{UF}_{\mathtt{u_i}})$ denote, respectively, the sets of true positives, false positives and false negatives of $\mathtt{m}$ given the feedback instances in $\mathtt{UF}_{\mathtt{u_i}}$.

When overlaps between the sets of feedback instances provided by different users are small, the approach we described for learning feedback is likely to be ineffective: predictions will be made using little or no evidence, and as such are likely to be inaccurate. To overcome this problem, we can make use of the precision and recall estimates defined above to learn feedback. Indeed, such estimates synthesize user requirements as to the quality of the mapping used to populate a given relation. Therefore, they can be used to compare users' requirements. Users with similar requirements are likely to be associated with close estimates, and, conversely, users with different requirements are likely to be associated with disparate estimates. Using such estimates, we can therefore learn the feedback instance a given user $\mathtt{u_i}$ would provide to annotate a given tuple $\mathtt{t}$ by matching the precision and recall estimates computed based on the feedback supplied by $\mathtt{u_i}$ with the precision and recall estimates computed for other users.

As well as collaborative learning techniques, feedback can be inferred by exploiting available domain knowledge. To illustrate this, consider the two following relational tables $\mathtt{Protein}(\mathtt{name}, \mathtt{accession}, \mathtt{gene}, \mathtt{pfam})$ and $\mathtt{ProteinFamily}(\mathtt{id}, \mathtt{desc})$ and consider a referential integrity constraint specifying that a value of the attribute `pfam` of the `Protein` relation is also a value of the attribute `id` of the `ProteinFamily` relation. Consider now that the user supplies a feedback instance specifying that `PF05387` is an expected value of the `pfam` attribute. Given the above integrity constraint, we can derive a feedback instance specifying that the value `PF05387` is an expected value of the `id` attribute. Likewise, if the user specifies that a given instance is an unexpected value for the `id` attribute, then we can derive a feedback instance specifying that such a value is unexpected for the `pfam` attribute.

It is also worth mentioning that feedback provides partial information about user requirements. We cannot expect users to provide feedback on every artifact. Instead, in most cases, users will be willing to provide feedback on a small subset of the objects they are presented with. Because of the scarcity of the feedback, the results of the operations presented for verifying the validity of feedback, clustering users and learning feedback may be wrong or misleading, e.g., an invalid feedback instance may be found to be valid, or two users with different requirements can be grouped into the same cluster. This raises the question as to how to manage uncertainty in the results produced by the operations given the feedback used as input. A possible solution that can be explored consists in devising metrics, such as those adopted in [8, 2], to estimate the degree to which the results of an operation are correct based on the proportion of objects that are annotated by feedback.

7. CONCLUSIONS

While it is recognised that user feedback can play a central role in dealing with the difficulties underlying the construction and maintenance of information integration systems, existing proposals, e.g., [2, 17, 4, 11], address specific integration sub-problems considering a specific kind of feedback sought, in most cases from a single user, to annotate a specific kind of artifacts. We have shown in this paper that treating feedback as a first class citizen presents several advantages. Specifically, we presented a straightforward model for describing different kinds of feedback that have been considered in the information integration literature. We proposed operations that can underpin the management of feedback within information integration systems, and that are applicable to feedback instances of the model we proposed. We illustrated through examples how such operations can be used to maximize the benefits that can be derived from feedback compared with existing proposals. In particular, the operations presented can be applied to feedback instances of different kinds supplied by users with different and changing requirements. We have presented preliminary solutions that can be adopted in the realization of such operations. Furthermore, we identified issues (and therefore research opportunities) that underlie feedback management in information integration systems.

8. REFERENCES

[1] B. Alexe, L. Chiticariu, R. J. Miller, and W. C. Tan. Muse: Mapping understanding and design by example. In *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, México*, pages 10–19. IEEE, 2008.

[2] K. Belhajjame, N. W. Paton, S. M. Embury, A. A. A. Fernandes, and C. Hedeler. Feedback-based annotation, selection and refinement of schema mappings for dataspaces. In *Proceedings of the 13th International Conference on Extending Database Technology, EDBT 2010, Lausanne, Switzerland*, pages 573–584. ACM, 2010.

[3] P. Berkhin. A survey of clustering data mining techniques. In J. Kogan, C. Nicholas, and M. Teboulle, editors, *Grouping Multidimensional Data: Recent Advances in Clustering*, pages 25–71. Springer, 2006.

[4] H. Cao, Y. Qi, K. S. Candan, and M. L. Sapino. Feedback-driven result ranking and query refinement for exploring semi-structured data collections. In *Proceedings of the 13th International Conference on Extending Database Technology, EDBT 2010, Lausanne, Switzerland*, pages 3–14. ACM, 2010.

[5] X. Chai, B.-Q. Vuong, A. Doan, and J. F. Naughton. Efficiently incorporating user feedback into information extraction and integration programs. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA*, pages 87–100. ACM, 2009.

[6] A. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada*, pages 271–280. ACM, 2007.

[7] A. Doan, R. Ramakrishnan, and A. Y. Halevy. Mass collaboration systems on the world-wide web. *Commun. ACM*. To appear.

[8] X. L. Dong, A. Y. Halevy, and C. Yu. Data integration with uncertainty. *VLDB J.*, 18(2):469–500, 2009.

[9] M. J. Franklin, A. Y. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *SIGMOD Record*, 34(4):27–33, 2005.

[10] A. Gionis, H. Mannila, and P. Tsaparas. Clustering aggregation. *TKDD*, 1(1), 2007.

[11] A. Y. Halevy, A. Rajaraman, and J. J. Ordille. Data integration: The teenage years. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea*, pages 9–16. ACM, 2006.

[12] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, 1999.

[13] S. R. Jeffery, M. J. Franklin, and A. Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada*.

[14] R. A. Kowalski and M. J. Sergot. A logic-based calculus of events. *New Generation Comput.*, 4(1):67–95, 1986.

[15] R. McCann, W. Shen, and A. Doan. Matching schemas in online communities: A web 2.0 approach. In *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, Cancún, México*, pages 110–119. IEEE, 2008.

[16] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:2–2, 2009.

[17] P. P. Talukdar, Z. G. Ives, and F. Pereira. Automatically incorporating new sources in keyword search-based data integration. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA*.

[18] P. P. Talukdar, M. Jacob, M. S. Mehmood, K. Crammer, Z. G. Ives, F. Pereira, and S. Guha. Learning to create data-integrating queries. *PVLDB*, 1(1):785–796, 2008.