ESTEEM

Emergent Semantics and cooperaTion in multi-knowledgE EnvironMents

http://www.dis.uniroma1.it/∼esteem/

Programma di Ricerca Cofinanziato dal MIUR (Esercizio 2005)

Stato dell'Arte

Carola Aiello, Roberto Baldoni, Silvia Bonomi, Tiziana Catarci, Alessia Milani, Diego Milano, Leonardo Querzoni, Adriano Rippa, Monica Scannapieco, Sirio Scipioni, Sara Tucci Piergiovanni

Sommario

Questo documento, che costituisce il deliverable D1.1 del progetto ESTEEM, ha lo scopo di fornire lo stato dell'arte nelle principali aree di ricerca legate agli obiettivi del progetto: (i) problematiche di qualità in sistemi d'integrazione dati, (ii) sistemi di reputazione, (iii) reti overlay P2P.

Data	31 luglio 2006
Tipo di prodotto	Rapporto tecnico
Numero di pagine	43
Unità responsabile	RM
Unità coinvolte	BS, MI, MIP, RM
Autore da contattare	Carola Aiello Dipartimento di Informatica e Sistemistica Università di Roma "La Sapienza" Via Salaria 113, 00198 Roma, Italia carola.aiello@dis.uniroma1.it

Stato dell'Arte

Carola Aiello, Roberto Baldoni, Silvia Bonomi, Tiziana Catarci, Alessia Milani, Diego Milano, Leonardo Querzoni, Adriano Rippa, Monica Scannapieco, Sirio Scipioni, Sara Tucci Piergiovanni

31 luglio 2006

Integrating and coordinating heterogeneous and distributed contents in open information systems over the Web is one of the crucial challenges at the current evolutionary stage of IT infrastructure. Semantic information sharing, rather than information processing or just syntactic exchange à la XML, is one of the information technology primary goals of these years. Semantics is particularly important because information has to be sharable in open environments where interacting nodes, which can be called sources, sites, agents, etc., and that we will simply call peers, do not necessarily share a common understanding of the world at hand. Lack of common background generates the need for explicit guidance in understanding the exact meaning of data, hence the growing importance of ontologies. Furthermore, meaningful exchanges between peers can only occur on the basis of dynamic trustful agreements on common interpretations within the context of a given task. This particular form of semantic interoperability is referred to as "emergent semantics". In this document, we will detail the SoA in the main areas that involve the emergent semantics research issues: (i) Data Quality Issues in Data Integration Systems, (ii) trust issues, (iii) P2P overlay networks, (iv) P2P for supporting trust.

1 Data Quality Issues in Data Integration Systems

1.1 Introduction

Integrating and coordinating heterogeneous and distributed contents in open information systems is one of the crucial challenges at the current evolutionary stage of IT infrastructure. Lack of common background generates the need for explicit guidance in understanding the exact meaning of data.

In distributed environments, data sources are typically characterized by various kinds of heterogeneities that can be generally classified into (i) technological heterogeneities, (ii) schema heterogeneities and (iii) instance level heterogeneities. *Technological heterogeneities* are due to the use of products by different vendors, employed at various layers of an information and communication infrastructure. An example of technological heterogeneity is the usage of two different relational database management systems like IBM's DB2 vs. Microsoft's SQLServer. *Schema heterogeneities* are principally caused by the use of (i) different data models, such as one source that adopts the relational data model and a different source that adopts the XML data model, and (ii) different data representations, such as one source that stores addresses as one single field and another source that stores addresses with separate fields for street, civic number, and city. *Instance-level heterogeneities* are caused by different, conflicting data values provided by distinct sources for the same objects. This type of heterogeneity can be caused by quality errors, such as accuracy, completeness, currency, and consistency errors; such errors may result, for instance, from independent processes that feed the different data sources.

Today, there are many examples of scenarios in which data residing at different sources must be accessed in a unified way, overcoming such heterogeneities. *Data integration* is a major research and business area that has the main purpose of allowing a user to access data stored by heterogeneous data sources through the presentation of a unified view of this data. Though data integration must face all the types of heterogeneities listed above, in this work we focus particularly on instance-level heterogeneities, where data quality issues become very significant. Indeed, instance-level heterogeneities can strongly affect query processing in data integration systems. Specifically, the query processing activity can be performed by considering that different data sources may exhibit different quality levels for the data. Hence, answering algorithms can be executed to provide the optimal quality results for the final user. Furthermore,when collecting data as answers to queries, possible conflicts must be solved, by means of a specific *instance-level conflict resolution* activity; otherwise, the whole integration process cannot be correctly terminated.

Quality-driven query processing and instance-level conflict resolution can be seen as two complementary approaches that deal with instance-level heterogeneities. Specifically, it is possible to consider

1. only-quality driven query processing (without conflict resolution);
2. only conflict resolution (without quality-driven query processing);
3. both approaches used complementarily.

Quality-driven query processing modifies the query answering semantics in order to take into account varying quality of source data. It can assume (case 1) that instance-level conflicts are not solved, but metadata are available in the system to return the best quality answer (see [55]). Instance-level conflict resolution can focus on solving conflicts between sources independently of the query processing (case 2), for example by operating not at query time but at a different phase of the data integration process, such as the population of a data warehouse (see [54]). Alternatively (case 3), conflicts resolution techniques can be performed at query-time, within the quality-driven query answering process itself (see [71]).

In this section, we provide an overview of existing proposals to deal with quality-driven query processing (Section 1.2) and with instance-level conflict resolution (Section 1.3).

1.2 Techniques for Quality-Driven Query Processing

In this section we provide an overview of several proposals to perform quality-driven query processing, which returns an answer to a global query, by explicitly taking into account the quality of data provided by local sources; however, several other techniques are present in the literature, e.g., [9, 8].

1.2.1 The QP-alg: Quality-Driven Query Planning

In this section, we describe the approach presented in [55], which we will refer to as QP-alg in the following.

The mapping between local sources and the global schema is specified by means of *query correspondence assertions* (QCAs) that have the general form

$$\texttt{MQ} \leftarrow \texttt{Si.vj} \leftarrow \texttt{WQ},$$

where (i) `MQ` is the mediator query and is a conjunctive query, (ii) `Si.vj` denotes an arbitrary view `vj` on the source `Si`, and (iii) `WQ` is the wrapper query. The mapping can be classified *global-local-as-view* (GLAV), as a query on the global schema is defined in terms of a query on the sources.

Three classes of data quality dimensions, called *information quality criteria* (IQ criteria), are defined:

- *Source-specific criteria*, defining the quality of a whole source. Examples of such criteria are *reputation* of the source, based on users' personal preferences, and *timeliness*, measured by the source update frequency.
- *QCA-specific criteria*, defining the quality of specific query correspondence assertions. An example of such criteria is *price*, i.e., the price to be paid for the query.
- *User-query specific criteria*, measuring the quality of the source with respect to the answer provided to a specific user query. An example of such criteria is *completeness*, based on the fullness of source relations.

Some IQ criteria metrics are predetermined, others are dynamically calculated, and the result is a set of IQ criteria vectors to be used to rank sources and plans. Note that, in a DBMS, given a query, query plans are constructed that are equivalent in terms of the query result provided; they are then ranked and selected on the basis of a cost model. Conversely, the plans built according to the QP-alg's approach produce different query results, though they are checked to be semantically correct. The phases of QP-alg are shown in Figure 1.

The first phase consists of a pruning of the source space by filtering out low quality sources on the basis of source-specific criteria. In order to classify sources on the basis of IQ criteria vectors, a multiattribute decision making method is used, namely, the data envelopment analysis [14].

The second phase creates plans by exploiting the fact that QCAs are actually views over the mediator schema, and thus basic data integration results for query answering using views can be exploited [44].

The third phase first evaluates the quality of QCAs (step 1 in plan selection in Figure 1). Specifically, QCA-specific criteria and user-query specific criteria are calculated for each QCA. Then, the quality of a plan is evaluated (step 2 in plan selection in Figure 1) by relying on a procedure similar to cost models for DBMSs. A tree is built for each plan, with QCAs as leaves and join operators as inner nodes. The IQ vector is recursively calculated for a node, starting from its children nodes. A set of "merge" functions for each quality criterion is defined in order to combine IQ vectors. As an example, the merge function for the price criterion is defined as the sum of both the right child and the left child of a given node, meaning that both queries must be made. In Figure 2, an example is shown, detailing how the price of a plan P_i is computed.

Then, plan ranking is performed by means of the simple additive weighting (SAW) method (step 3 in plan selection in Figure 1). Specifically, the final IQ score for a plan is computed as the weighted sum of scaled criteria, where weights represent the "importance" of each criterion to the user. Finally, the best plans, according to the performed ranking, are returned.

1.2.2 DaQuinCIS Query Processing

The DaQuinCIS system, described in [71], is a framework for dealing with data quality in cooperative information systems. A module of the system, the *data quality broker*, is a data integration system.

In this section we focus on the proposed query answering process, which is one of the functionalities of the Data Quality Broker.

The main idea of the DaQuinCIS approach is to make cooperating organizations export not only data that they intend to exchange with other organizations, but also metadata, that characterize their quality level. To this extent, a specific semistructured data model is proposed, called D^2Q. On the basis of such quality characterization of exported data, user queries are processed so that the "best quality" answer is returned as a result.

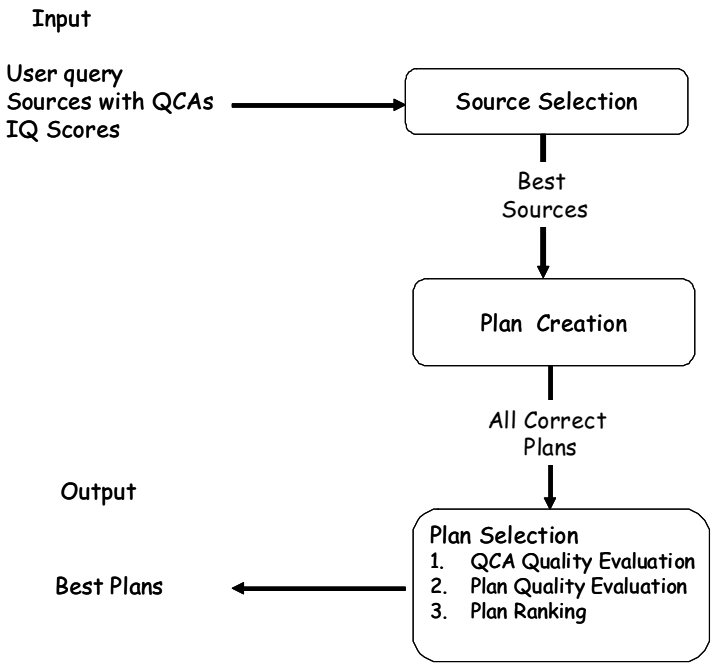

Figure 1: Phases of the QP-alg approach

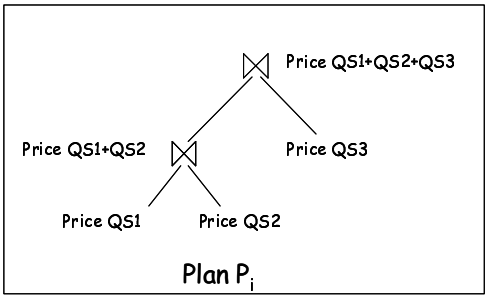

Figure 2: Example of price computation for the plan P_i

Queries on the global schema are processed according to the *global-as-view* (GAV) approach by unfolding , i.e., by replacing each atom of the original query with the corresponding view on local data sources. When defining the mapping between concepts of the global schema and concepts of the local schemas, while the extension of global-level concepts can be retrieved by multiple sources, the mapping is actually defined to retrieve the union of local source extensions. Such a mapping definition stems directly from the assumption that the same concept can have different extensions at a local source level due to data quality errors. Therefore, when retrieving data, they can be compared and a best quality copy can be either selected or constructed.

More specifically, query processing in DaQuinCIS is performed by the following sequence of steps:

1. *Query unfolding.* A global query $\mathcal{Q}$ is unfolded according to a static mapping that defines each concept of the global schema in terms of the local sources; this mapping is defined in order to retrieve all copies of the same data that are available, i.e., exported by the cooperating organizations according to the D^2Q model. Therefore, the query $\mathcal{Q}$ is decomposed into $\mathcal{Q}_1, \ldots, \mathcal{Q}_k$ queries to be posed over local sources. Such queries are then executed to return a set of results $\mathcal{R}_1, \ldots, \mathcal{R}_k$ (see Figure 3).

2. *Extensional checking.* In this step, a record matching algorithm is run on the set $\mathcal{R}_1 \cup \mathcal{R}_2 \cup \ldots \cup \mathcal{R}_k$. The result of the running of the record matching algorithm is the construction

of a set of clusters composed by records referring to the same real-world objects $\mathcal{C}_1, \ldots, \mathcal{C}_z$ (see Figure 3, middle).

3. *Result building.* The result to be returned is built by relying on a *best quality default semantics*. For each cluster, a best quality representative is either selected or constructed. Each record in the cluster is composed of couples in which a quality value `q` is associated with each field value `f`. The best quality record for each cluster is selected as the record having the best quality values in all fields, if such a record exists. Otherwise, a best quality record is constructed by composing the fields that have the highest quality from records within the same cluster. Once representatives for each cluster have been selected, the result $\mathcal{R}$ is constructed as the union of all cluster representatives (see Figure 3, right). Each quality value `q` is a vector of quality values corresponding to the different quality dimensions. For instance, `q` can include values for accuracy, completeness, consistency, and currency. These dimensions have potentially different scales; therefore, a scaling step is needed. Once scaled, those vectors need to be ranked. Therefore a ranking method must also be applied. Both scaling and ranking problems have well-known solutions, e.g., multi-attribute decision making methods, like AHP [65].

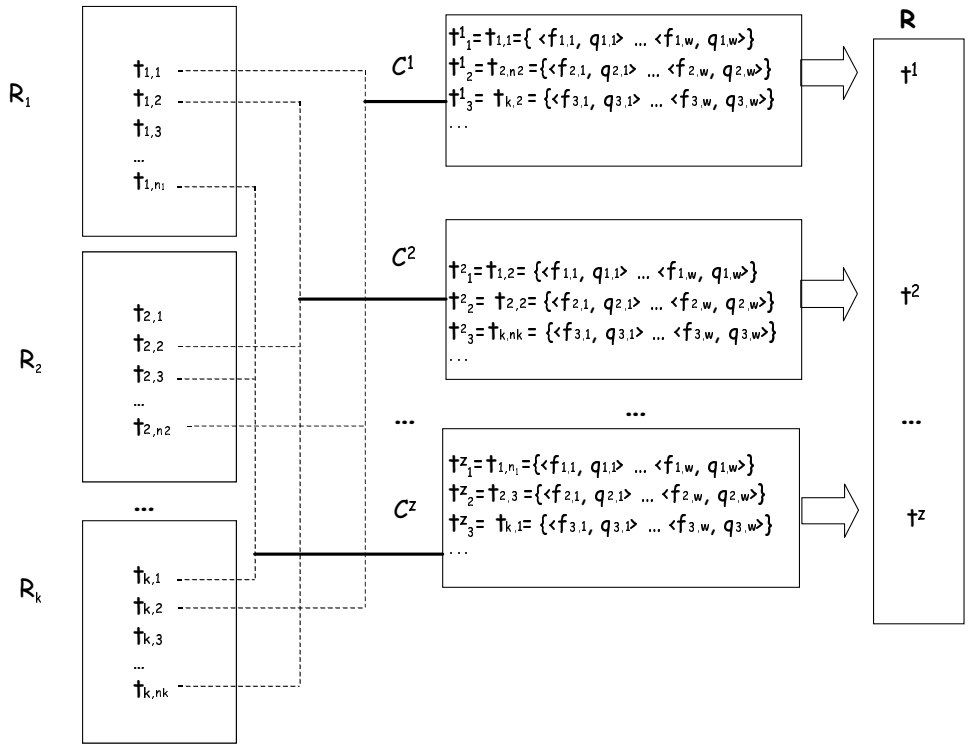

Figure 3: The query result construction in DaQuinCIS

1.2.3 Fusionplex Query Processing

Fusionplex[49] models a data integration system by (i) a relational global schema `D`; (ii) a set of relational local sources (D_i, d_i), where d_i is the instance of the local schema D_i; and (iii) a set of schema mappings (D, D_i). The mapping definition is GLAV, i.e., views on the global schema are put in correspondence with views on schema of the local sources. In FusionPlex, it

is assumed that the *schema consistency assumption* holds, meaning that there are no modeling errors at the local sources, but only modeling differences. Instead, it is assumed that the *instance inconsistency assumption* holds, meaning that the same instance of the real world can be represented differently in the various local sources due to errors. In order to deal with such instance-level inconsistencies, Fusionplex introduces a set of metadata, called *features*, about the sources to be integrated.

Source features include time stamp,availability, and accuracy. The data integration framework definition presented above is extended by including features into the definition of schema mappings. Specifically, the mappings are triples consisting of a global schema view $\mathtt{D}$, a local schema view $\mathtt{D}_i$, and, in addition the features associated with the local view. Fusionplex includes an extension of the relational algebra that takes into account the association of a set of features $F = \{F_1 \ldots F_n\}$ with source relations. For instance, the extended cartesian product concatenates the database values of the participating relations, but fuses their feature values. The fusion method depends on the particular feature. Therefore, the availability value of the new tuple is the product of the availability values of the input tuples; the time stamp is the minimum of the input time stamps; and so on. In this setting, query processing is performed in several steps:

1. Given a query Q, the set of *contributing views* is identified. First, the sets of attributes of the query and each contributing view are intersected. If the intersection is empty, the contributing view is not relevant. Next, the selection predicates of the query and the contributing view are joined. If the resulting predicate is true, then the contributing view is considered relevant to the query.

2. Once relevant contributing views are identified, *query fragments* are derived as the unit of information suitable for populating the answer to the query. A query fragment results from the removal from the contributing view of all tuples and attributes that are not requested in the query, and from the addition of null values for the query attributes that are missing from the contributing view. As an example, in Figure 4, two contributing views, $\mathtt{C}_1$ and $\mathtt{C}_2$, are shown, and the corresponding query fragments, $\mathtt{QF}_1$ and $\mathtt{QF}_2$.

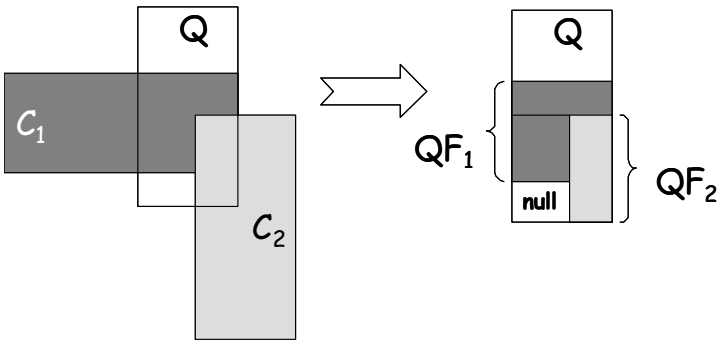

Figure 4: Example of query fragment construction from contributing views

3. From each relevant contributing view, a single query fragment is constructed, where some of these fragments may be empty. The union of all nonempty query fragments is termed a *polyinstance* of the query. Intuitively, a polyinstance includes all the information derived from the data sources in response to a user query.

In order to provide a unique answer to the query Q for the user, instance-level conflicts present in the polyinstance must be solved. Once polyinstances have been constructed a strategy for conflict detection and resolution is applied, as described in Section 1.3.2.

1.2.4 Comparison of Quality-Driven Query Processing Techniques

In Figure 5, a comparison of the query processing techniques described is shown. The techniques are compared according to the following features:

- *Quality metadata*, showing that each technique is based on a set of metadata that support the query processing activity.
- *Granularity of the quality model* that represents data elements quality metadata can be associated with. QP-alg associates quality metadata not only with sources but also with query correspondence assertions and user queries. DaQuinCIS exploits the flexibility of a semistructured data model for quality association at various granularity levels. Fusionplex allows association only at a source level.
- *Type of mapping*, showing that both QP-alg and Fusionplex have a GLAV approach to the mapping definition, while DaQuinCIS has a GAV approach.
- *Support to quality algebra*, meaning that quality values associated with local source data need to be "combined" by means of specific algebraic operators. There are some research proposals in this direction, but it is still an open problem. Some attempts toward the algebraic manipulation of quality values are present in the merge functions of QP-alg and in the extension of the relational operators of Fusionplex.

Techniques	Quality Metadata	Granularity of Quality Characterization	Type of Mapping	Quality Algebra Support
QP-alg	YES	Source, Query Correspondences Assertions, User Queries	GLAV	Preliminary
DaQuinCIS Query Processing	YES	Each data element of a semistructured data model	GAV	No
FusionPlex Query Processing	YES	Source	GLAV	Preliminary

Figure 5: Comparison of quality-driven query processing techniques

1.3 Instance-level Conflict Resolution

Instance-level conflict resolution is a major activity in data integration systems. No data integration system can return answers to user queries if these types of conflicts are not solved. As data integration typically deals with heterogeneous and autonomous sources, instance-level conflicts are very common and frequent. Unfortunately, most of the existing data integration solutions have simplifying assumptions regarding conflicts on data values.

In this section, after a classification of these conflicts (Section 1.3.1), we describe some of the existing proposals of instance-level conflict resolution techniques (Section 1.3.2), and we conclude with a comparison between techniques (Section 1.3.3).

1.3.1 Classification of Instance-Level Conflicts

As already mentioned, in order to integrate data coming from distinct data sources, problems caused by technological, schema, and instance-level heterogeneities need to be solved. In the following section, we briefly describe conflict originating from schema heterogeneities, called

schema-level conflicts, while the latter part is devoted to the description of conflicts due to instance-level heterogeneities, called *instance-level conflicts*.

Schema-level conflicts have been extensively studied (see [41]), and include

- *heterogeneity conflicts*, occurring when different data models are used;
- *semantic conflicts*, regarding the relationship between model element extensions. For instance, a `Person` entity may have different extensions in different sources that may be disjoint, partially overlapping, including one into another, or completely overlapping;
- *description conflicts*, concerning the description of concepts with different attributes. These conflicts include different formats, different attribute types, and different scaling. These conflicts are on the boundary between schema level conflicts and instance-level conflicts; for instance, in [25], such conflicts are classified as data value conflicts. We prefer to consider description conflicts at a schema-level because they are actually caused by different design choices of data schemas, though such choices certainly have an impact on values to be integrated;
- *structural conflicts*, regarding different design choices within the same model. For instance, such conflicts may occur if one source represents an `Address` as an entity and another source represents it as an attribute.

In contrast with schema-level conflicts, instance-level conflicts have received much less attention, and only recently has the importance of these types of conflicts increased, due to the primary role they play in the data integration processes. Instance-level conflicts are due to poor quality of data; they occur because of errors in the data collection process or the data entry process or because sources are not updated.

According to the granularity of the model element, instance-level conflicts can be distinguished into *attribute conflicts* and *key conflicts*, also called *entity* or *tuple conflicts*. Some works, e.g., [45], also consider *relationship conflicts* that are particularly meaningful at a conceptual level. In the following, we will focus on attribute and key conflicts, as they are the principal conflict types involved in data integration processes.

Let us consider two relational tables, $\mathtt{S}_1(\mathtt{A}_1, \ldots, \mathtt{A}_k, \mathtt{A}_{k+1}, \ldots, \mathtt{A}_n)$ and $\mathtt{S}_2(\mathtt{B}_1, \ldots, \mathtt{B}_k, \mathtt{B}_{k+1}, \ldots, \mathtt{B}_m)$, where $\mathtt{A}_1 = \mathtt{B}_1 \ldots \mathtt{A}_k = \mathtt{B}_k$. Let the same real world entity be represented by the tuple $\mathtt{t}_1$ in $\mathtt{S}_1$ and by the tuple $\mathtt{t}_2$ in $\mathtt{S}_2$, and let $\mathtt{A}_i = \mathtt{B}_i$; the following conflicts can be defined:

- An *attribute conflict* occurs iff
$$\mathtt{t}_1.\mathtt{A}_i \neq \mathtt{t}_2.\mathtt{B}_i.$$
- Let us further suppose that $\mathtt{A}_i$ is a primary key for $\mathtt{S}_1$ and $\mathtt{B}_i$ is a primary key for $\mathtt{S}_2$. A *key conflict* occurs iff
$$\mathtt{t}_1.\mathtt{A}_i \neq \mathtt{t}_2.\mathtt{B}_i \text{ and } \mathtt{t}_1.\mathtt{A}_j = \mathtt{t}_2.\mathtt{B}_j,$$
for all $\mathtt{j}$ ranging from 1 to $\mathtt{k}$, and $\mathtt{i} \neq \mathtt{j}$.

In Figure 6, several examples of both attribute and key conflicts are shown. In the figure, two relations, `EmployeeS1` and `EmployeeS2`, represent information about employees of a company. Notice that we assume there is no schema-level conflict, i.e., the two relations have exactly the same attributes and the same extension. Nevertheless, they present instance-level conflicts. Two attribute value conflicts are shown, concerning the `Salary` of the employee `arpa78` and the `Surname` of the employee `ghjk09` in the two relations. A key-level conflict is also shown between the employee `Marianne Collins`, as identified in the relation `EmployeeS1` and as identified in relation `EmployeeS2`, assuming that the two tuples represent the same real-world object.

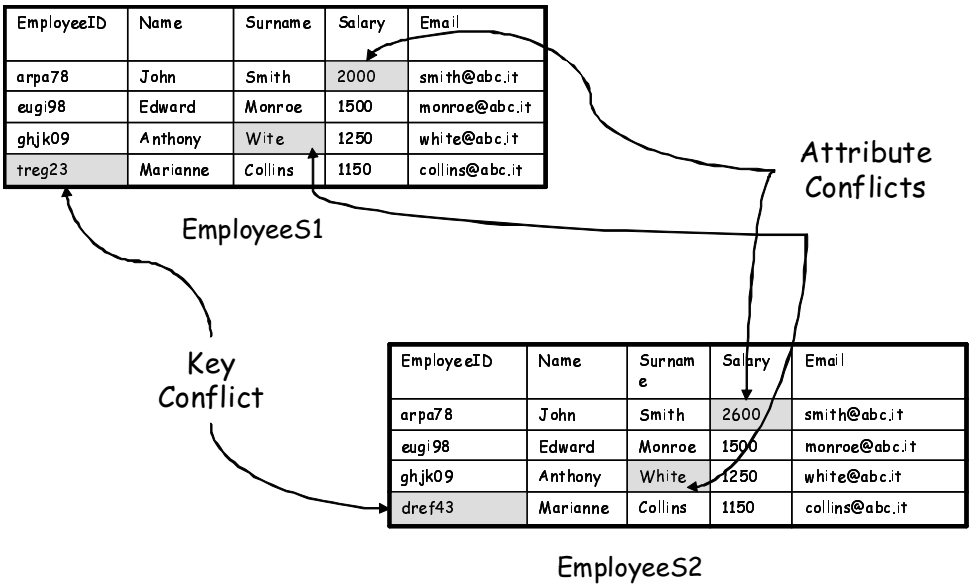

Figure 6: An example of key- and attribute-level conflicts

Instance-level conflicts can be present in both virtual and materialized integration. In virtual data integration, a theoretical formulation of the problem has been proposed. Specifically, the cited key and attribute conflicts have been formally specified as a violation of integrity constraints expressed over the global schema representing the integrated view.

In the next section, we will describe several techniques proposed in order to solve instance-level conflicts.

1.3.2 Overview of Techniques

Techniques that deal with instance-level conflicts can be applied in two different phases of the life cycle of a data integration system, namely, at *design time* and at *query time*. In both cases, the actual conflicts occur at query time; however, the design time approaches decide the strategy to follow for fixing conflicts before queries are processed, i.e., at the design stage of the data integration system. The techniques operating at query time incorporate the specification of the strategy to follow within query formulation.

A proposal for solving conflicts at design time can be found in [19]. The main idea is to resolve attribute conflicts by means of aggregation functions to be specified for each attribute that may involve conflicts during query execution time.

Design time techniques have a major optimization problem, as outlined in [80]. Let us consider the example shown in Figure 6, and suppose that it is specified at design time, for the `Salary` attribute, that in the case of conflicts the minimum salary must be chosen. Given a global schema, `Employee(EmployeeID, Name, Surname, Salary, Email)`, let us consider the following query:

```
SELECT EmployeeID, Email
FROM   Employee
WHERE  Salary < 2000
```

Since the `Salary` attribute is involved in the query, all employees must be retrieved in order to compute the minimum salary, not only employees with `Salary < 2000`, even if no conflicts on salary occur. Therefore, conflict resolution at design time may be very inefficient.

Query time conflict resolution techniques have been proposed to overcome such performance inefficiencies. Furthermore, query time techniques are characterized by greater flexibility, since, as we will see, they allow those who formulate the query to indicate a specific strategy to adopt for conflict resolution. Given a user query posed on the global schema, query time techniques deal with key and/or attribute conflicts that may occur on the data retrieved as results.

Key conflicts require the application of object identification techniques that are not described in this work. With reference to the example shown in Figure 6, object identification techniques will match the tuple `treg23` from `EmployeeS1` with the tuple `dref43` from `EmployeeS2` by comparing the attribute values of the two tuples in order to determine whether the "Marianne Collins" represented in the two sources is the same person. After a positive matching decision, the tuples referring to "Marianne Collins" will be considered a single tuple, and a unique key will be chosen to identify the tuple, thereby solving the key conflict. If the matching decision is negative, no key conflict occurs.

With respect to attribute conflicts, several techniques for solving them have been proposed:

- SQL-based conflict resolution [54];
- Aurora [80];
- Fusionplex [49];
- DaQuinCIS [70];
- FraSQL-based conflict resolution [72]; and
- OO_{RA} [45].

In the following we describe the details of such techniques; however, several other proposals are present in the literature, including [26, 57]. Before providing the detailed description, we illustrate which are the "abstract" steps to be followed for solving attribute-level conflicts.

Let us consider again the example in Figure 6, and let us suppose the following query is formulated over the global schema `Employee(EmployeeID, Name, Surname, Salary, Email)`:

```
SELECT Salary
FROM   Employee
WHERE  Name = "John" AND Surname = "Smith"
```

In order to return a result to this type of query, the attribute conflict between the two values for John Smith's salary stored in the relations `EmployeeS1` and `EmployeeS2` must be solved.

A solution to this problem is to *declaratively* specify how to deal with such conflicts. A declarative specification consists of

- a set of conflict resolution functions that, on the basis of the specific attributes involved in the conflict, can select the most appropriate value;
- a set of strategies to deal with conflicts, corresponding to different tolerance degrees; and
- a query model that can take into account possible conflicts directly, i.e., with specific extensions, such as adhoc functions dealing with conflicts, or indirectly, i.e., without specific extensions.

A *resolution function* takes two (or more) conflicting values of an attribute as input and outputs a value that must be returned as the result to the posed query. Common resolution functions are `MIN` and `MAX`. To these, resolution functions that are specific to some attribute types can be added. For instance, for numerical attribute types, `SUM` and `AVG` can be used.

For non numerical attributes, further resolution functions can be identified, such as `CONCAT`. In [54], a resolution function `MAXIQ` is proposed. Assuming the presence of a data quality model that associates quality values to model elements (e.g., attributes), the resolution function `MAXIQ` returns the value with the highest quality. In Figure 7, conflict resolution functions are summarized, as proposed in [54]. Some functions are the usual aggregation functions; others serve the specific purpose of resolving conflicts.

Function	Attribute Type	Description
COUNT	any	Counts number of conflicting values
MIN	any	Minimum value
MAX	any	Maximum value
RANDOM	any	Random non null value
CHOOSE(Source)	any	Chooses most reliable source for the particular attribute
MAXIQ	any	Value of highest information quality
GROUP	any	Groups all conflicting values
SUM	numerical	Sums all values
MEDIAN	numerical	Median value, namely having the same number of higher and lower values
AVG	numerical	Arithmetic mean of all values
VAR	numerical	Variance of values
STDDEV	numerical	Standard Deviation of values
SHORTEST	non-numerical	Minimum length value, ignoring spaces
LONGEST	non-numerical	Maximum length value, ignoring spaces
CONCAT	non-numerical	Concatenation of values
ANNCONCAT	non-numerical	Annotated concatenation of values, whose purpose is to specify the source, before the actual returned value

Figure 7: Resolution functions as proposed in [54]

The *tolerance strategies* allow the user to define the degree of conflict permitted. For example, it is possible to specify that on a specific attribute no conflicts are admitted. This means that all values returned by the sources on that attribute must be aligned. As another example, it may be possible to specify that in the case of conflicts, a randomly chosen value among the conflicting ones be proposed as the result. As another tolerance strategy, a threshold value may be specified for distinguishing tolerable conflicts from intolerable ones. For instance, a conflict on two values for the `Name` attribute such as `Michael` and `Maichael`, that have a reciprocal edit distance of one character, can be tolerated since it is very easy to transform `Maichael` into `Michael`, by deleting simply one character. In contrast, for a numerical attribute like `Salary`, even a one digit distance may be intolerable.

With respect to the *query model*, it is possible to appropriately use SQL to specify how to solve conflicts [54], or to use adhoc extensions such as the ones proposed in [45] and [80].

The next sections will describe several techniques for conflict resolution that instantiate the abstract steps presented.

SQL-Based Conflict Resolution The approach proposes formulating queries in SQL, exploiting the capabilities of current database systems. Three possible strategies are discussed, based on three SQL operations:

- *Group*, where by using the `Group by` SQL statement, a query is specified that groups

tuples on the basis of one or more group attributes. Then, an aggregated function is specified to select conflicting values appropriately. For instance,

```
SELECT   EmployeeId, min(Salary)
FROM     Employee
GROUP BY EmployeeId
```

The main disadvantage of this approach is that only the aggregation functions supported by SQL can be used.

- *Join*, which considers the union of two sources, and partitions it into three sets: the intersection of the two sources, the tuples only in the first source, and the tuples only in the second source. Then, the merging query is expressed on each of these parts, and, finally, the results are merged. The first query is expressed on the intersection:

```
SELECT EmployeeID, min(Employee1.Salary, Employee2.Salary)
FROM   Employee1, Employee2
WHERE  Employee1.EmployeeId = Employee2.EmployeeId
```

This query has the advantage that the resolution is no longer an aggregate function, but a scalar one. This extends the possibility of using user-defined functions, thereby enlarging the spectrum of possible resolution functions while continuing to be compliant with most database systems that allow user-defined scalar functions. The following query selects the tuples of the first source that are not in the second:

```
SELECT EmployeeId, Price
FROM   Employee1
WHERE  Employee1.EmployeeId NOT IN
       (SELECT EmployeeID
        FROM Employee2)
```

The query to select the tuples of the second source that are not in the first source is similar to the above one. The query to merge is simply the combination of the results of all queries through the `UNION` operator. The main disadvantage of this approach is the complexity of the queries, because the number of partitions increases exponentially with the number of sources. The length and complexity of queries may become prohibitive.

- *Nested Join*, an improvement over the previous method, which can be performed when resolution functions are associative. Given N sources to be merged, the idea is to first merge two, then merge this with a third, and so on. With this approach, queries grow linearly, but still remain complex.

Aurora Aurora is a mediation-based data integration system. The approach proposes a conflict-tolerant query model for conflict resolution at a desired degree. The conflict-tolerant query model has the following features:

- Two operators, for attribute conflict resolution, called *resolve attribute-level conflict* (RAC) and for tuple conflict resolution, called *resolve tuple-level conflict*(RTC). The operators take a resolution function as parameter. For example, consider the global population of the relation `Employee`, shown in Figure 8, which represents the global instance resulting from the integration of the two relations `EmployeeS1` and `EmployeeS2` shown in Figure 6.

An example of how the operator RAC works is reported in Figure 9, where the specified resolution functions are `MIN` for `Salary`, `LONGEST` for `Surname`, and `ANY` for `EmployeeID`. An example for the RTC operator is shown in Figure 10, where the resolution function is `ANY`, and the tuple conflicts are solved choosing tuple `dref43`.

- Three strategies for conflict resolution, namely, `HighConfidence, RandomEvidence`, and `PossibleAtAll`. These strategies allow the user to define the degree of conflicts permitted, and are used in conjunction with the previously described operators when formulating queries. `HighConfidence` allows us to specify that no conflicts on a specific attribute are admitted. This means that all values returned by the sources on that attribute must be aligned. `RandomEvidence` specifies that in the case of conflicts, a runtime function has to select a value to be returned. `PossibleAtAll` returns all values that correctly answer the query, independently of conflicts.

TupleID	EmployeeID	Name	Surname	Salary	Email
t_1	arpa78	John	Smith	2000	smith@abc.it
t_2	eugi98	Edward	Monroe	1500	monroe@abc.it
t_3	ghjk09	Anthony	Wite	1250	white@abc.it
t_4	treg23	Marianne	Collins	1150	collins@abc.it
t_5	arpa78	John	Smith	2600	smith@abc.it
t_6	eugi98	Edward	Monroe	1500	monroe@abc.it
t_7	ghjk09	Anthony	White	1250	white@abc.it
t_8	dref43	Marianne	Collins	1150	collins@abc.it

Figure 8: Instance of the global relation `Employee`

TupleID	EmployeeID	Name	Surname	Salary	Email
t_1	arpa78	John	Smith	2000	smith@abc.it
t_2	eugi98	Edward	Monroe	1500	monroe@abc.it
t_3	ghjk09	Anthony	White	1250	white@abc.it
t_4	treg23	Marianne	Collins	1150	collins@abc.it

RAC(Employee,Salary(MIN), Surname(Longest), EmployeeID(Any))

Figure 9: Resolution of attribute conflicts

TupleID	EmployeeID	Name	Surname	Salary	Email
t_1	arpa78	John	Smith	2600	smith@abc.it
t_2	eugi98	Edward	Monroe	1500	monroe@abc.it
t_3	ghjk09	Anthony	Wite	1250	white@abc.it
t_4	dref43	Marianne	Collins	1150	collins@abc.it

RTC(Employee,ANY)

Figure 10: Resolution of tuple conflicts

The conflict-tolerant query model is built on tuple-level conflicts only, but the user is allowed to specify attribute-level conflict resolution. Some examples of conflict-tolerant queries are as follows:

- Q1: SELECT EmployeeID, Name (ANY), Salary[MIN]
  FROM Employee
  WHERE Salary>1800
  WITH HighConfidence

- Q2: SELECT [ANY]EmployeeID, Name, Salary
  FROM Employee
  WHERE Salary>1800
  WITH RandomEvidence

Both queries select employees with Salary greater than 1800 euros. If there is a conflict, Q1 selects employees whose Salary value is greater than 1800 in all sources. Therefore, based on Figure 8, the tuples t_1 and t_5 are selected. Then, applying the resolution function MIN on Salary, the returned tuple will have the Salary value of t_1, namely, 2000. Q2 selects a random Salary value, and, if it is greater than 1800, it is returned as a result. Then, the ANY tuple resolution function is applied as specified in the selection clause. Based on Figure 8, a random value between the Salary value of t_1 and t_5 is returned.

Fusionplex and DaQuinCIS The two approaches to conflict resolution adopted in the Fusionplex and DaQuinCIS systems are similar. They both resolve attribute conflicts on the basis of metadata associated with data of local sources.

Fusionplex proposes the following metadata, called *features* :

- *time stamp*, representing the time the source was validated in the system;
- *cost*, which can be transmission time over the network, or money to be paid for information or both;
- *accuracy*, evaluated according to a probabilistic approach;
- *availability*, probability that the information is randomly available; and
- *clearance*, corresponding to the clearance level needed to access the information.

In Fusionplex, the features are associated with sources as a whole, with the restrictive assumption that data in sources are homogeneous with respect to a specific feature.

DaQuinCIS proposes the following metadata, referred to as *dimensions*:

- *accuracy*, concerning the syntactical accuracy of data values;
- *currency*, considering the degree of up-to-dateness of values;
- *consistency*, measuring intrasource integrity constraints; and
- *completeness*, counting the number of null values.

The D^2Q data model is semistructured, and permits the association of metadata with data elements of different granularity, and therefore, with single values, as well as with attributes and all other model elements.

An example of the extended SQL statements that can be defined in Fusionplex is

```
SELECT EmployeeID, Salary
FROM   EmployeeS1, EmployeeS2
WHERE  EmployeeS1.EmployeeID=EmployeeS2.EmployeeID
USING  cost>0.6
WITH   timestamp as 0.5
```

Considering an XML-based representation of the two relations `EmployeeS1` and `EmployeeS2`, an example of a DaQuinCIS query, expressed in XQuery [11], is

```
FOR    $i in input()//EmployeeS1
FOR    $j in input()//EmployeeS2
WHERE  ($i/EmployeeID=$j/EmployeeID) and
       quality($i/Salary)>0.7 and quality($j/salary)>0.7
RETURN ($i/Name,$i/Salary)
```

As described, attribute conflict resolution is based on metadata in both Fusionplex and DaQuinCIS. Also, both systems have a step in which, upon issuing a user query, all the significant instances answering the query are collected and grouped into clusters of different copies of the same objects. Then, in both systems, a resolution policy is applied in order to produce selected tuples to be included in the result.

The two systems differ in the process for building the final result. In Fusionplex, as described in Section 1.2.3, the phase in which results are collected from local sources terminates with the construction of a polyinstance, upon which a conflict resolution strategy is applied. Conflict resolution is performed in two phases: in the first phase a utility function is used to take user preferences into account, while in the second phase the actual fusion is performed.

With reference to the first phase, users can specify the importance they assign to each feature. Then, an overall utility function consisting of the weighted sum of the feature values of a source is calculated, and a first pruning of sources is done on the basis of a fixed utility threshold.

With respect to the second phase, resolution of inconsistencies can be done either on the basis of their features, called *feature-based resolution*, or on the basis of the data, called *content-based resolution*.

A resolution policy consists of the sequential selection of

- *elimination functions*, which can be feature-based or selection-based. Examples of elimination functions are `MIN` and `MAX`; `MAX(timestamp)` and `MIN(cost)` are examples of feature-based elimination functions, while `MAX(Salary)` is an example of a content-based elimination function.

- *fusion functions*. Fusion functions are always content-based; examples are `ANY` and `AVERAGE`.

Note that the resolution policy is completely specified by users according to their specific requirements. Moreover, Fusionplex admits three tolerance levels: no resolution, pruning of polytuples, and selective attribute resolution. The no resolution policy aloows an answer with conflicts to be returned to the user. The pruning of polytuple policy removes tuples that either do not satisfy the feature selection predicate or are below the utility threshold. The selective attribute resolution forces resolution on some attributes only.

In the DaQuinCIS system, the reconciled result is produced according to the process described in Section 1.2.2, and it is completely based on quality values associated with data on the basis of the D^2Q model.

FraSQL-Based Conflict Resolution The approach proposes an extension of a multidatabase query language, called *FraSQL*, which provides operations for transformation and integration of heterogeneous data. The main idea is to use grouping for duplicate elimination and aggregation for conflict resolution. For conflict resolution, FraSQL provides both *user-defined aggregation* and *user-defined grouping*. User defined aggregation is useful for conflict resolution, allowing for the selection of a representative value from a group of values corresponding to the same real-world object. The grouping of values is performed by means of user-defined grouping. User defined grouping can be of two types: (i) *context free* and (ii) *context aware*. Context free grouping is the usual approach, as in SQL standards, with, in addition, the possibility of using

external functions. The following query shows the usage of a context free user-defined grouping [72]:

```
SELECT   avg (Temperature),rc
FROM     Weather
GROUP BY regionCode(Longitude,Latitude) AS rc
```

where `regionCode` is an external function that computes the region from its geographical position.

Context-aware grouping is proposed in order to overcome some limitations of the current SQL standardized group by operator. Indeed, SQL standardized group by operator works one tuple at a time, not considering possible relationships between grouping tuples. Therefore, in order to have a more flexible grouping, similarity criteria can be introduced that split or merge the group conveniently. As an example, consider the query:

```
SELECT EmployeeID,Salary
FROM   EmployeeS1
GROUP  maximumDifference(Salary,diff=150)
BY CONTEXT
```

The query considers the relation EmployeeS1 shown in Figure 6 and groups the tuples as shown in Figure 11, generating three sets corresponding to tuples for which the `Salary` values differ by at most 150.

EmployeeID	Salary
arpa78	2000

EmployeeID	Salary
eugi98	1500

EmployeeID	Salary
ghjk09	1250
treg23	1150

Figure 11: Result of the context-aware query as applied to the table EmployeeS1 of Figure 6

OO$_{RA}$ Though in the following we focus only on attribute-level conflicts, the model also considers key conflicts and relationship conflicts (see [45] for more details on these two conflict types). The approach distinguishes two types of attribute conflicts, namely, *tolerable conflicts* which can be automatically solved, and *intolerable conflicts*, which have to be solved with human intervention. The two types of conflicts are separated by means of a threshold. An extended object-oriented data model, called OO$_{RA}$, is proposed to handle attribute-level conflicts. The main features of the model with respect to attribute conflict resolution are

- the possibility of specifying thresholds and resolution functions for attribute-level conflict resolution; and
- the representation of original and resolved attribute values.

With respect to the threshold specification and resolution functions, the following three different combinations are considered for a given attribute: (i) threshold predicate and resolution

function both unspecified; (ii) specified threshold predicate and unspecified resolution function; and (iii) threshold predicate and resolution function both specified. In case (i), no conflict is tolerated, so if a conflict arises, the resolved attribute value is null. In case (ii), a conflict can arise and can be acceptable, but if it arises, the returned value is `NULL`. In case (iii), there can be tolerable conflicts, and the returned value is computed by the resolution function.

With respect to conflicting values representation, the OO_{RA} approach for every non-identifier attribute represents a triple: original value, resolved value, and conflict type. Conflict type is `NULL` if there is no conflict, `RESOLVABLE` if there is an intolerable conflict, and `ACCEPTABLE` if there is a tolerable conflict. For example, let us consider the following threshold predicate and resolution function applied to the global relation described in Figure 8:

```
DEFINE Salary.threshold@EMPLOYEE(s1,s2) = (abs(s1-s2)<=1000)
DEFINE Salary.resolution@EMPLOYEE(s1,s2) = MIN(s1,s2)
```

In this case, the conflict between t_1 and t_5 is tolerable, as the differences between the two values for salary are within the specified threshold. The conflict is solved by choosing the value for salary present in tuple t_1.

As another example, let us consider the following threshold predicate and resolution function, also applied to the relation in Figure 8:

```
DEFINE Surname.threshold@EMPLOYEE(s1,s2) = (editDistance(s1,s2)<=1)
DEFINE Surname.resolution@EMPLOYEE(s1,s2) = LONGEST(s1,s2)
```

Still, the conflict between t_3 and t_7 is tolerable, and the value for `Surname` stored by tuple t_7 is returned as a result. In contrast, supposing that the `Surname` value for t_3 were `Wie`, and the edit distance between t_3.`Salary` and t_7.`Salary` greater than 1, an intolerable conflict would have occurred.

1.3.3 Comparison of Instance-level Conflict Resolution Techniques

In Figure 12, the different declarative techniques for the resolution of inconsistencies are compared with respect to permitted tolerance strategies and query models. Reviewing the tolerance strategies column, Aurora, Fusionplex, and OO_{RA} propose a degree of flexibility that can be selected once conflicts occur. We recall that the three degrees of flexibility proposed by Aurora are (i) high confidence, meaning that no conflict is tolerated, (ii) random evidence meaning that in the case of conflicts a runtime function will select the value to be returned, and (iii) possible at all, meaning that all values that correctly answer the query must be returned. Similarly to Aurora, Fusionplex admits three tolerance levels: no resolution, pruning of polytuples, and selective attribute resolution. The no resolution policy corresponds to PossibleAtAll; in both approaches, the answer with conflicts is returned to the user. The pruning of polytuple policy, which removes tuples not satisfying the feature selection predicate or the utility threshold, is a more specific case of the RandomEvidence policy and it shares the threshold concept with OO_{RA}. The selective attribute resolution involves leaving some (or all) attributes unresolved; it is a specific case of the no resolution policy with higher granularity.

Reviewing the query model column, we see that the SQL-based conflict resolution can rely on SQL. However, it has inefficiencies due to the fact that resolution functions were not considered for the native SQL. Therefore, computing aggregation and expressing SQL statements for them can become very onerous. Both DaQuinCIS and OO_{RA} deal with models that are different from the relational model, namely, with the XML data model and the object-oriented data model, respectively.

Techniques	Tolerance Strategies	Query Model
SQL-Based Conflict Resolution	NO	SQL
Aurora	High Confidence, RandomEvidence, PossibleAtAll	Ad-hoc Conflict Tolerant Query Model
Fusionplex	No resolution strategy, selective attribute resolution	Extended SQL
DaQuinCIS	NO	Extended XML
FraQL-Based Conflict Resolution	NO	Ad-hoc FraQL
OO_{RA}	Thresholds for tolerable and intolerable conflicts	Ad hoc Object Oriented Extension (OO_{RA})

Figure 12: Conflict resolution techniques

1.4 Summary

Data integration and data quality are two interrelated concepts. On the one hand, data integration can benefit from data quality. Quality-driven query processing techniques have the purpose of selecting and accessing data of the highest quality, thus deriving the maximum benefits from a context with multiple sources with varying quality of their data assets. In open contexts, such as P2P systems, these techniques are becoming increasingly more important. On the other hand, it is intuitive that most data quality problems become evident when data in one source are compared with similar data stored in a different source. Once they are detected, there is the need for appropriate mechanisms that allow a data integration system to perform the query processing function. These techniques are the conflict resolution techniques, which play the significant role of supporting query processing in virtual data integration systems. Note that the choice of solving conflicts at query time is an alternative to the more expensive choice of cleaning data sources *before* they are actually integrated. This would indeed require a data quality improvement activity performed independently by each source, and hence the complexity and the cost would grow.

In materialized data integration, e.g., in data warehouses, a cleaning activity is performed when populating the global schema. As instances gathered by disparate sources typically present instance-level conflicts, conflict resolution techniques can be also effectively applied for the purpose of producing a consistent materialized global instance.

2 Trust Issues

2.1 Introduction

Trust and Reputation Systems represent a significant trend in decision support for Internet mediated service provision. The basic idea is to let parties rate each other, for example after the completion of a transaction, and use the aggregated ratings about a given party to derive a trust or reputation score, which can assist other parties in deciding whether or not to transact with that party in the future.

Let's Consider as example online exchange, i.e. data sharing, goods sale or service provision, that commonly takes place between parties who have never transacted with each other before, in an environment where the client often has insufficient information about the provider, and about the goods/data and services offered. This forces the consumer to accept the "risk of

prior performance", i.e. to pay for services and goods before receiving them, which can leave him in a vulnerable position. The consumer generally has no opportunity to check products before the transaction. The provider, on the other hand, knows exactly what he supplies and often knows what he gets: i.e. when he is paid in money. The ineffciencies resulting from this information asymmetry can be mitigated through trust and reputation. The idea is that even if the consumer can not verify the product/data or service in advance, he can be confident that it will be what he expects as long as he trusts the provider. A trusted provider therefore has a significant advantage in case the product quality can not be verified in advance.

It turns out obvious that trust plays a crucial role in computer mediated transactions and processes. As a result, the topic of trust in open computer networks is receiving considerable attention in the academic community and e-commerce industry.

2.2 The notions of Trust and Reputations

Manifestations of trust are easy to recognise because we experience and rely on it every day, but at the same time trust is quite challenging to define because it manifests itself in many different forms. The literature on trust can also be quite confusing because the term is being used with a variety of meanings [59]. Two common definitions of trust which we will call *reliability trust* and *decision trust* respectively will be used in this study. As the name suggests, reliability trust can be interpreted as the reliability of something or somebody. The definition by Gambetta (1988) [27] provides an example of how this can be formulated:

Definition 1 (Reliability Trust) *Trust is the subjective probability by which an individual, A, expects that another individual,B, performs a given action on which its welfare depends.*

This definition includes the concept of *dependence* on the trusted party, and the *reliability* (probability) of the trusted party, as seen by the trusting party.

However, trust can be more complex than Gambetta's definition indicates. For example, Falcone & Castelfranchi (2001) [24] recognise that having high (reliability) trust in a person in general is not enough to decide to enter into a situation of dependence on that person. In [24] they write: *"For example it is possible that the value of the damage per se (in case of failure) is too high to choose a given decision branch, and this independently either from the probability of the failure (even if it is very low) or from the possible payoff (even if it is very high). In other words, that danger might seem to the agent an intolerable risk"*. In order to capture this broad concept of trust, the following definition inspired by McKnight & Chervany (1996) [59] can be used.

Definition 2 (Decision Trust) *Trust is the extent to which one party is willing to depend on something or somebody in a given situation with a feeling of relative security, even though negative consequences are possible.*

The relative vagueness of this definition is useful because it makes it more general. It explicitly and implicitly includes aspects of a broad notion of trust which are *dependence* on the trusted entity or party, the *reliability* of the trusted entity or party, *utility* in the sense that positive utility will result from a positive outcome, and negative utility will result from a negative outcome, and finally a certain *risk attitude* in the sense that the trusting party is willing to accept the situational risk resulting from the previous elements.

The main differences between trust and reputation systems can be described as follows: Trust systems produce a score that reflects the relying party's subjective view of an entity's trustworthiness, whereas reputation systems produce an entity's (public) reputation score as seen by the whole community. There can of course be trust systems that incorporate elements of reputation systems and vice versa, so that it is not always clear how a given systems should be classified. The descriptions of the various trust and reputation systems below must therefore be seen in light of this.

2.3 Reputation Systems

The rich literature growing around trust and reputation systems for Internet transactions, as well as the implementation of reputation systems in successful commercial applications, give a strong indication that this is an important technology. The commercial and live implementation seems to have settled around relatively simple schemes, whereas a multitude of different systems with advanced features are being proposed by the academic community. In this section we give an overview of existing both kind of models: (i)Trus Models proposed by the accademic community and (ii)the Reputations Systems used by some popular website.

2.3.1 Reputations Computation Engines

Seen from the relying party's point of view, trust and reputation scores can be computed based on own experience, on second hand referrals, or on a combination of both. Reputation systems are typically based on public information in order to reflect the community's opinion in general. A party who relies on the reputation score of some remote party, is in fact trusting that party through *trust transitivity* [38]. Some systems take both public and private information as input. Private information, e.g. resulting from personal experience, is normally considered more reliable than public information, such as ratings from third parties. This section describes various principles for computing reputation and trust measures. Some of the principles are used in commercial applications, whereas others have been proposed by the academic community.

Simple Summation or Average of Ratings The simplest form of computing reputation scores is simply to sum the number of positive ratings and negative ratings separately, and to keep a total score as the positive score minus the negative score. This is the principle used in eBay's reputation forum which is described in detail in [61]. The advantage is that anyone can understand the principle behind the reputation score, the disadvantage that it is primitive and therefore gives a poor picture participants' reputation score although this is also due to the way rating is provided.

A slightly more advanced scheme proposed in e.g. [77] is to compute the reputation score as the average of all ratings. This principle is used in the reputation systems of numerous commercial web sites, such as Epinions, and Amazon.(Section 2.3.2)

Advanced models in this category compute a weighted average of all the ratings, where the rating weight can be determined by factors such as rater trustworthiness/ reputation, age of the rating, distance between rating and current score etc.

Bayesian Systems Bayesian systems take binary ratings as input (i.e. positive or negative), and are based on computing reputation scores by statistical updating of beta probability density functions (PDF). The *a posteriori* (i.e. the updated) reputation score is computed by combining the *a priori* (i.e. previous) reputation score with the new rating [35, 37, 50, 51, 79]. The reputation score can be represented in the form of the beta PDF parameter tuple (α, β) (where α and β represent the amount of positive and negative ratings respectively), or in the form of the probability expectation value of the beta PDF, and optionally accompanied with the variance or a confidence parameter. The advantage of Bayesian systems is that they provide a theoretically sound basis for computing reputation scores, and the only disadvantage is that it might be too complex for average persons to understand.

Discrete Trust Models Humans are often better able to rate performance in the form of discrete verbal statements, than continuous measures. This is also valid for determining trust measures, and some authors, including [1, 12, 13, 47], have proposed discrete trust models. For example, in the model of Abdul-Rahman & Hailes [1] trustworthiness of an agent x can be referred as *Very Trustworthy, Trustworthy, Untrustworthy* and *Very Untrustworthy.* The relying

party can then apply his or her own perception about the trustworthiness of the referring agent before taking the referral into account. Look-up tables, with entries for referred trust and referring party downgrade/ upgrade, are used to determine derived trust in x. Whenever the relying party has had personal experience with x, this can be used to determine referring party trustworthiness. The assumption is that personal experience reflects x's real trustworthiness and that referrals about x that differ from the personal experience will indicate whether the referring party underrates or overrates. Referrals from a referring party who is found to overrate will be downgraded, and vice versa.

The disadvantage of discrete measures is that they do not easily lend themselves to sound computational principles. Instead, heuristic mechanisms like look-up tables must be used.

Belief Models Belief theory is a framework related to probability theory, but where the sum of probabilities over all possible outcomes not necessarily add up to 1, and the remaining probability is interpreted as uncertainty.

Jsang [35, 36] has proposed a belief/trust metric called *opinion* denoted by $\omega_x^A = (b, d, u, a)$ which expresses the relying party A's belief in the truth of statement x. Here b,d, and u represent belief, disbelief and uncertainty respectively where $b, d, u \in [0, 1]$ and $b + d + u = 1$. The parameter $a \in [0, 1]$, which is called the relative atomicity, represents the base rate probability in the absence of evidence, and is used for computing an opinion's probability expectation value $E(\omega_x^A) = b + au$, meaning that a determines how uncertainty shall contribute to $E(\omega_x^A)$. When the statement x for example says "David is honest and reliable", then the opinion can be interpreted as reliability trust in David. As an example, let us assume that Alice needs to get her car serviced, and that she asks Bob to recommend a good car mechanic. When Bob recommends David, Alice would like to get a second opinion, so she asks Claire for her opinion about David. When trust and trust referrals are expressed as opinions, each transitive trust path Alice→Bob→David, and Alice→Claire→David can be computed with the *discounting operator*, where the idea is that the referrals from Bob and Claire are discounted as a function Alice's trust in Bob and Claire respectively. Finally the two paths can be combined using the *consensus operator*. These two operators form part of *Subjective Logic* [30], and semantic constraints must be satisfied in order for the transitive trust derivation to be meaningful [34]. Opinions can be uniquely mapped to beta PDFs, and in this sense the consensus operator is equivalent to the Bayesian updating. This model is thus both belief-based and Bayesian.

Fuzzy Models Trust and reputation can be represented as linguistically fuzzy concepts, where membership functions describe to what degree an agent can be described as e.g. trustworthy or not trustworthy. Fuzzy logic provides rules for reasoning with fuzzy measures of this type. The scheme proposed by Manchala [47] as well as the REGRET reputation system proposed by Sabater & Sierra [66, 68, 67] fall in this category. In Sabater & Sierra's scheme, what they call *individual reputation* is derived from private information about a given agent, what they call *social reputation* is derived from public information about an agent, and what they call *context dependent reputation* is derived from contextual information.

Flow Models Systems that compute trust or reputation by transitive iteration through looped or arbitrarily long chains can be called flow models. Some flow models assume a constant trust/reputation weight for the whole community, and this weight can be distributed among the members of the community. Participants can only increase their trust/reputation at the cost of others. Google's PageRank [56], the Appleseed algorithm [82] and Advogato's reputation scheme [42] described in Section 2.3.2 belong to this category. In general, a participant's reputation increases as a function of incoming flow, and decreases as a function of outgoing flow. In the case of Google, many hyperlinks to a web page contribute to increased PageRank whereas many hyperlinks from a web page contribute to decreased PageRank for that web page. Flow models

do not always require the sum of the reputation/trust scores to be constant. One such example is the EigenTrust model [39] which computes agent trust scores in P2P networks through repeated and iterative multiplication and aggregation of trust scores along transitive chains until the trust scores for all agent members of the P2P community converge to stable values.

2.3.2 Commercial and Live Reputations Systems

This section describes the most well known applications of online reputation systems. All analysed systems have a centralised network architecture. The computation is mostly based on the summation or average of ratings, but two systems use the flow model.

eBay's Feedback Forum eBay is a popular auction site that allows sellers to list items for sale, and buyers to bid for those items. The so-called Feedback Forum on eBay gives buyer and seller the opportunity to rate each other (provide feedback in the eBay jargon) as positive, negative, or neutral (i.e. 1, -1, 0) after completion of a transaction. Buyers and sellers also have the possibility to leave comments like *"Smooth transaction, thank you!"* which are typical in positive case or *"Buyers beware!"* in the rare negative case. The Feedback Forum is a centralised reputation system, where eBay collects all the ratings and computes the scores. The running total reputation score of each participant is the sum of positive ratings (from unique users) minus the sum of negative ratings (from unique users). In order to provide information about a participant's more recent behaviour, the total of positive, negative and neutral ratings for the three different time windows i) past six months, ii) past month, and iii) past 7 days are also displayed.

There are many empirical studies of eBay's reputation system, see Resnick et al. [62] for an overview. In general the observed ratings on eBay are surprisingly positive. Buyers provide ratings about sellers 51.7% of the time, and sellers provide ratings about buyers 60.6% of the time [55]. Of all ratings provided, less than 1% is negative, less than 0.5% is neutral and about 99% is positive. It was also found that there is a high correlation between buyer and seller ratings, suggesting that there is a degree of reciprocation of positive ratings and retaliation of negative ratings. This is problematic if obtaining honest and fair ratings is a goal, and a possible remedy could be not to let sellers rate buyers.

The problem of ballot stuffing, i.e. that ratings can be repeated many times, e.g. to unfairly boost somebody's reputation score, seems to be a minor problem on eBay because participants are only allowed to rate each other after the completion of a transaction, which is monitored by eBay. It is of course possible to create fake transactions, but because eBay charges a fee for listing items, there is a cost associated with this practice. However, unfair ratings for genuine transactions can not be avoided.

The eBay reputation system is very primitive and can be quite misleading. With so few negative ratings, a participant with 100 positive and 10 negative ratings should intuitively appear much less reputable than a participant with 90 positive and no negatives, but on eBay they would have the same total reputation score. Despite its drawbacks and primitive nature, the eBay reputation system seems to have a strong positive impact on eBay as a marketplace. Any system that facilitates interaction between humans depend on how they respond to it, and people appear to respond well to the eBay system and its reputation component.

Expert Sites Expert sites have a pool of individuals that are willing to answer questions in their areas of expertise, and the reputation systems on those sites are there to rate the experts. Depending on the quality of a reply, the person who asked the question can rate the expert on various aspects of the reply such as clarity and timeliness.

AllExperts provides a free expert service for the public on the Internet with a business model based on advertising. The reputation system on AllExperts uses the aspects: *Knowledgeable,*

Clarity of Response, Timeliness and Politeness where ratings can be given in the interval [1,10]. The score in each aspect is simply the numerical average of ratings received. The number of questions an expert has received is also displayed in addition to a *General Prestige* score that is simply the sum of all average ratings an expert has received. Most experts receive ratings close to 10 on all aspects, so the General Prestige is usually close to 10 $\times$ the number of questions received. It is also possible to view charts of ratings over periods from 2 months to 1 year.

AskMe is an expert site for a closed user group of companies and their employees, and the business model is based on charging a fee for participating in the AskMe network. AskMe does not publicly provide any details of how the system works.

Advogato is a community of open-source programmers. Members rank each other according to how skilled they perceive each other to be, using Advogato's trust scheme, which in essence is a centralised reputation system based on a flow model. The reputation engine of Advogato computes the reputation flow through a network where members constitute the nodes and the edges constitute referrals between nodes. Each member node is assigned a capacity between 800 and 1 depending on the distance from the source node that is owned by Raph Levien who is the creator of Advogato. The source node has a capacity of 800 and the further away from the source node is, the smaller the capacity is. Members can refer each other with the status of *Apprentice* (lowest), *Journeyer* (medium), *Master* (highest). A separate flow graph is computed for each type of referral. A member will get the highest status for which there is a positive flow to his or her node. For example if the flow graph of Master referrals and the flow graph of Apprentice referrals both reach member l then that member will have Master status, but if only the flow graph of Apprentice referrals reaches member x then that member will have Apprentice status. The Advogato reputation systems does not have any direct purpose other than to boost the ego of members, and to be an incentive for social and professional networking within the Advogato community.

Product Review Sites Product review sites have a pool of individual reviewers who provide information for consumers for the purpose of making better purchase decisions. The reputation systems on those sites apply to products as well as to the reviewers themselves.

Epinions founded in 1999 is a product and shop review site with a business model mainly based on so-called cost-per-click online marketing, which means that Epinions charges product manufacturers and online shops by the number of clicks consumers generate as a result of reading about their products on Epinions' web site. Epinions also provides product reviews and ratings to other web sites for a fee. Epinions has a pool of members who write product and shop reviews. Anybody from the public can become a member simply by signing up. The product and shop reviews written by members consist of prose text and quantitative ratings from 1 to 5 stars for a set of aspects such as *Ease of Use*, *Battery Life* etc. in case of products, and *Ease of Ordering*, *Customer Service*, *On-Time Delivery* and *Selection* in case of shops. Other members can rate reviews as *Not Helpful*, *Somewhat Helpful*, *Helpful*, and *Very Helpful*, and thereby contribute to determine how prominently the review will be placed, as well as to give the reviewer a higher status. A member can obtain the status *Advisor*, *Top Reviewer* or *Category Lead* (highest) as a function of the accumulated ratings on all his or her reviews over a period. It takes considerable reviewing effort to obtain a status above member, and most members don't have any status. Category Leads are selected at the discretion of Epinions staff each quarter based on nominations from members. Top Reviewers are automatically selected every month based on how well their reviews are rated, as well as on the EpinionsWeb of Trust (see below), where a member can *Trust* or *Block* another member. Advisors are selected in the same way as Top Reviewers, but with a lower threshold for review ratings. Epinions does not publish the exact thresholds for becoming Top Reviewer or Advisor, in order to discourage members from trying to manipulate the selection process. The Epinions Web of Trust is a simple scheme, whereby members can decide to either *trust* or *block* another member. A member's list of trusted members represents

that member's personal Web of Trust. As already mentioned, the Web of Trust influences the automated selection of Top Reviewers and Advisors. The number of members (and their status) who trust a given member, will contribute to that member getting a higher status. The number of members (and their status) who block another member will have a negative impact on that member getting a higher status.

Epinions has an incentive systems for reviewers called the *Income Share Program*, whereby members can earn money. Income Share is automatically determined based on general use of reviews by consumers. Reviewers can potentially earn as much for helping someone make a buying decision with a positive review, as for helping someone avoid a purchase with a negative review. This is important in order not to give an incentive to write biased reviews just for profit. As stated on the Epinions FAQ pages: *"Epinions wants you to be brutally honest in your reviews, even if it means saying negative things"*. The Income Share pool is a portion of Epinions' income. The pool is split among all members based on the utility of their reviews. Authors of more useful reviews earn more than authors of less useful reviews. The Income Share formula is not specified in detail in order to discourage attempts to defraud the system. Highly rated reviews will generate more revenue than poorly rated reviews, because the former are more prominently placed so that they are more likely to be read and used by others. Category Leads will normally earn more than Top Reviewers who in turn will normally earn more than Advisors, because their reviews per definition are rated and listed in that order. Providing high quality reviews is Epinions core value proposition to consumers, and the reputation system is instrumental in achieving that. The reputation system can be characterised as highly sophisticated because of the revenue based incentive mechanism. Where other reputation systems on the Internet only provide immaterial incentives like status or karma, the Epinions system can provide hard cash.

BizRate BizRate runs a *Customer Certified Merchant* scheme whereby consumers who buy at a BizRate listed store are asked to rate site navigation, selection, prices, shopping options and how satisfied they were with the shopping experience. Consumers participating in this scheme become registered BizRate members. A *Customer Certificate* is granted to a merchant if a sufficient number of surveys over a give period are positive, and this allows the merchant to display the BizRate Customer Certified seal of approval on it's web site. As an incentive to fill out survey forms BizRate, members get discounts at listed stores. This scheme does not capture the frustrated customers who give up before they reach the check, and therefore tends to provide a positive bias of web stores. Thus is understandable from a business perspective, because it provides an incentive for stores to participate in the Customer Certificate scheme. BizRate also runs a product review service similar to Epinions, but which uses a much simpler reputation system. Members can write product reviews on BizRate, and anybody can become a member simply by signing up. Users, including nonmembers, who browse BizRate for product reviews can vote on reviews as being *helpful*, *not helpful* or *off topic*, and the reputation system stops there. Reviews are ordered according to the ratio of helpful over total votes, where the reviews with the highest ratios are listed first. It is also possible to have the reviews sorted by rating, so that the best reviews are listed first. Reviewers do not get any status and they can not earn money by writing reviews for BizRate. There is thus less incentive for writing reviews on BizRate than there is on Epinions, but it is uncertain how this influences the quality of the reviews. The fact that anybody can sign up to become a member and write reviews and that anybody including non members can vote on the helpfulness of reviews makes this reputation scheme highly vulnerable to attack. A simple attack could consist of writing many positive reviews for a product and ballot stuff them so that they get presented first and result in a high average score for that product.

Amazon Amazon is mainly an online bookstore that allows members to write book reviews. Amazon's reputation scheme is quite similar to the one BizRate uses. Anybody can become a member simply by signing up. Reviews consist of prose text and a rating in the range 1 to 5 stars. The average of all ratings gives a book its average rating. Users, including non-members,

can vote on reviews as being *helpful* or *not helpful*. The numbers of helpful as well as the total number of votes are displayed with each review. The order in which the reviews are listed can be chosen by the user according to criteria such as "newest first", "most helpful first" or "highest rating first".

As a function of the number of helpful votes each reviewer has received, as well as other parameters not publicly revealed, Amazon determines each reviewer's rank, and those reviewers who are among the 1000 highest get assigned the status of Top 1000, Top 500, Top 100, Top 50, Top 10 or #1 Reviewer. Amazon has a system of *Favourite People*, where each member can choose other members as favourite reviewers, and the number of other members who has a specific reviewer listed as favourite person also influences that reviewer's rank. Apart from giving some members status as top reviewers, Amazon does not give any financial incentives. However there are obviously other financial incentives external to Amazon that can play an important role. It is for example easy to imagine why publishers would want to pay people to write good reviews for their books on Amazon.

There are many reports of attacks on the Amazon review scheme where various types of ballot stuffing has artificially elevated reviewers to top reviewer, or various types of "bad mouthing" has dethroned top reviewers. This is not surprising due to the fact that users can vote without becoming a member. For example the Amazon #1 Reviewer usually is somebody who posts more reviews than any living person could possibly do if it would require that person to read each book, thus indicating that the combined effort of a group of people, presented as a single person's work, is needed to get to the top. Also, reviewers who have reached the Top 100 rank have reported a sudden increase in negative votes which reflects that there is a cat fight taking place in order to get into the ranks of top reviewers. In order to reduce the problem, Amazon only allows one vote per registered cookie for any given review. However deleting that cookie or switching to another computer will allow the same user to vote on the same review again. There will always be new types of attacks, and Amazon needs to be vigilant and respond to new types of attacks as they emerge. However, due to the vulnerability of the review scheme it can not be described as a robust scheme.

Google's Web Page Ranking System The early web search engines such as Altavista simply presented every web page that matched the key words entered by the user, which often resulted in too many and irrelevant pages being listed in the search results. Altavista's proposal for handling this problem was to offer advanced ways to combine keywords based on binary logic. This was too complex for users and therefore did not represent a good solution. PageRank proposed by Page *et al.* [56] represents a way of ranking the best search results based on a page's reputation. Roughly speaking, PageRank ranks a page according to how many other pages are pointing at it. This can be described as a reputation system, because the collection of hyperlinks to a given page can be seen as public information that can be combined to derive a reputation score. A single hyperlink to a given web page can be seen as a positive rating of that web page. Google's search engine is based on the PageRank algorithm and the rapidly rising popularity of Google at the cost of Altavista was obviously caused by the superior search results that the PageRank algorithm delivered.

Without specifying many details, Google state that the PageRank algorithm they are using also take other elements into account, with the purpose of making it difficult or expensive to deliberately influence PageRank. In order to provide a semantic interpretation of a PageRank value, a hyperlink can be seen as a positive referral of the page it points to. Negative referrals do not exist in PageRank so that it is impossible to blacklist web pages with the PageRank algorithm alone. Before Google with it's PageRank algorithm entered the search engine market, some webmasters would promote web sites in a spamlike fashion by filling web pages with large amounts of commonly used search key words as invisible text or as metadata in order for the page to have a high probability of being picked up by a search engine no matter what the user

searched for. Although this still can occur, PageRank seems to have reduced that problem because a high PR is also needed in addition to matching key words in order for a page to be presented to the user. PageRank applies the principle of trust transitivity to the extreme because rank values can flow through looped or arbitrarily long hyperlink chains. Some theoretic models including [39, 42, 82] do also allow looped and/or infinite transitivity.

2.4 Trust-aware Data Management

2.4.1 A Probabilistic Trust Model for Data Sharing in Cooperative Information Systems

Peer-to-peer (P2P) systems are typically used in loosely coupled environments, like the Web, where peers interact each other without previously established mutual agreements and knowledge. The model described in this section consider the problem of trusting P2P organizations that cooperate according to a *tight* interaction paradigm, commonly implemented in Cooperative Information Systems (CISs). These systems well-model real scenarios such as virtual districts in e-Business or public administrations in e-Government. When organizations composing a CIS exchange data each other, a relevant problem is how to trust each organization with respect to the quality of provided data. The model described in this section allows for trusting cooperating organizations, in which a trust value is assigned to each organization with respect to a specific data category. The trust level is assigned on the basis of a probabilistic model that considers the satisfaction of the receiving organization in a data exchange. For instance, in the Italian e-Government scenario the Department of Finance can have a high trust level with respect to Fiscal Codes of Citizens and a low trust level with respect to their Residence Addresses. The proposed model also allows to fix a threshold to discriminate between trusted and untrusted organizations.

In P2P tightly coupled systems, a first important difference w.r.t. loosely ones, is the complete knowledge of the identity of peers involved in data exchanges. The possibility to maintain an anonymous identity is common in peer to peer applications. However, malicious agents can exploit this weakness to spread undesired contents or dangerous applications, like virus. The use of IP addresses to avoid these behaviors was proposed [18], but this solution is poorly efficient; for example, spoofing techniques or use of dynamic IP address and proxies servers can easily make the method unreliable. The organizations that take part to a P2P tightly coupled system are instead known to each other and this reduces the probability of fraudulent behaviors. Nevertheless, we can't exclude that an organization has a dishonest behavior. For example, let us suppose that organizations Org_i and Org_j have the same kind of data $\mathcal{D}$. If $\mathcal{D}$ are somewhat crucial for Org_i's processes, Org_i could try to discredit Org_j to get a kind of data stewardship on $\mathcal{D}$ data.

Another important difference is the dynamism of the system, in terms of the number of participating organizations. In loosely coupled P2P systems, the number of peers changes frequently, due to the typical anonymous and transient nature of interactions. Conversely, P2P tightly coupled organizations typically have a stable process-based interaction.

These differences have two main implications on the trust model, namely:

- a more specific trust model can be considered, based on the definition of trust with respect to a category of data;
- misbehavior may be involuntary, e.g. due to temporary problems. Therefore, once an organization is classified as untrusted, it cannot remain in this state indefinitely, but specific reinstatement methods need to be provided.

When deciding the *atomic* unit to trust, a first hypothesis could be to trust the organization as a whole, with respect to the totality of exchanged data or more generally to the transactions

performed with other organizations. The method proposed in [5] is an example of this case.

In this case, the approach is of associating trust to an organization as a whole but proposes two major modifications, namely: *(i)* a specific type of transaction is considered, i.e. data exchanges; *(ii)* trust of an organization is evaluated with respect to a specific type of provided data.

More specifically, a trust value is associated to a couple $< Org_i, \mathcal{D} >$ where $\mathcal{D}$ is a data unit (see below). In this case we have a finest granularity level of trust on the sources. Organizations can choose different partners for data exchanges relying onto a wider range of possibilities in sources' selection.

Before describing the trust model, we provide some basic definitions:

Organizations. They are considered with respect to the role of providing data each other and consuming data from each other. Notice that in such a way they are peers, i.e. they can have both roles of data consumer and data provider. Furthermore, we suppose that organizations are independent and have a competitive behavior.

Data Unit. A data unit can be a generic view on data provided by organizations. As an example, a class `Citizen`, or a single attribute, such as `Name` of `Citizen`.

Source Unit. A source unit is a couple $< Org_i, \mathcal{D} >$, where Org_i is the organization providing the data unit $\mathcal{D}$.

Complaint. Given Org_i sending a data unit $\mathcal{D}$ to Org_j, Org_j can raise a complaint $C_{i,\mathcal{D}}$ stating that $\mathcal{D}$ data are low (not satisfactory) quality data.

Complaints are used to calculate trust of source units; specifically, the proposed method considers the overall number of data exchanges with respect to which complaints are raised.

Definition of a Trust Parameter for Source Units The trust level of a source unit is calculated on the basis of the number of complaints fired by other organizations. Let us assume that $\mathcal{O}$ is the set of peer organizations and that $\mathcal{C}_{i,j,\mathcal{D}} = \mathcal{C}(Org_i, < Org_j, \mathcal{D} >)$ is the complaint that Org_i fires with respect to the source unit $< Org_j, \mathcal{D} >$.

The number of complaints is not sufficient by itself to evaluate source units' trust. Therefore, a mechanism ia proposed to store the number of requests made to each source unit. The mechanism associates such information to each complaint, thus defining the following structure for messages exchanged within the P2P system in order to guarantee trust management:

$$\mathcal{C} =< Org_i, < Org_j, \mathcal{D} >, n >$$

where Org_i is the organization which has fired the complaint and $< Org_j, \mathcal{D} >$ is the source unit the complaint is referred to. The integer n represents the number of requests of $\mathcal{D}$ issued by Org_i to Org_j, starting from the last complaint that Org_i fired against Org_j.

Furthermore, let us assume that $n_{i,j,\mathcal{D}}$ is the number of complaints that Org_i sends to $< Org_j, \mathcal{D} >$. The following source unit's trust parameter is introduced:

$$R(< Org_j, \mathcal{D} >) = \frac{\sum_i \mathcal{C}_{i,j,\mathcal{D}}}{\sum_i n_{i,j,\mathcal{D}}} \quad \forall Org_i \in \mathcal{O}$$

The numerator represents the overall number of the complaints issued by organizations with reference to the data unit $< Org_i, \mathcal{D} >$. The denominator is the overall number of interactions in which the data unit $\mathcal{D}$ is sent by Org_j to other organizations.

High values of $R(\cdot)$ mean that the source unit is not trustworthy. Each interaction could be modelled as a random variable of a probabilistic process, that can have value 1 if a complaint is fired, 0 otherwise. Specifically, the random variable X is introduced such that:

$$X = \begin{cases} 1 & \text{if a complaint is fired} \\ 0 & \text{otherwise} \end{cases}$$

Without loss of generality, the variable X is supposed to have a binomial probability distribution with $P(X = 0) = p$ and $P(X = 1) = 1 - p$. Moreover, the following basic assumption holds: $p \ll (1 - p)$. In fact it is reasonable to suppose that the number of unsuccessful interactions is low with respect to the number of successful ones.

Therefore, $R(\cdot)$ is a random variable, linear combination of a large number of independent random variables that have the same distribution of probability. Thus, due to the "Central Limit Theorem", $R(\cdot)$ happens to have a normal probability distribution.

A Criterion for Trust. The calculation of $R(\cdot)$ values allows one to establish a sorting of source units on the basis of their trust values. Such a rating can be very important because an organization can choose to request data from another organization which has not the best trust value (though being trusted) but for example has a higher response time.

Nevertheless, it is important to distinguish between trusted and untrusted organizations. Therefore, a *threshold* is introduced, such that one can easily identify which organizations can be trusted for data with a fixed probability. A threshold also allows for applying a policy for punishing untrusted organizations. When $R(< Org_j, \mathcal{D} >)$ exceeds the threshold value, the source unit is automatically excluded from the system.

It is above discussed that $R(\cdot)$ is a normal distributed variable. So, the mean m and the standard deviation σ are calculated by exploiting properties of the normal distribution; the following trust criterion is provided:

$$\texttt{IF } R(< Org_j, \mathcal{D} >) \leq m + 2 \cdot \sigma \texttt{ THEN } < Org_j, \mathcal{D} > \texttt{TRUSTED}$$
$$\texttt{ELSE } < Org_j, \mathcal{D} > \texttt{UNTRUSTED}$$

According to the properties of the normal distribution, such criterion ensures that the right number of samples is selected with a probability at least equal to 95%.

The hypothesis of the same distributions of X variables for the different organizations could be removed. In this case, the given criterion continues to be valid on the basis of the Tchebycheff inequality [21], i.e.:

$$P(|X - \mu| > K \cdot \sigma) < \frac{1}{K^2}$$

where μ is the mean and σ the variance of $R(\cdot)$ samples. In the case of a normal distribution, the inequality is exactly reduced to the given criterion for $K = 2$. In the general case, the Tchebycheff inequality simply guarantees a probability lower bound of 75%.

Managing Malicious Behaviors

Cheating for Data Stewardship. In the hypothesis of the proposed model, it may happen that an organization sends a large number of complaints in order to discredit another organization on the same data it owns; in this way, it can conquer the stewardship on such data. In order to prevent from such a malicious behavior, it is introduced an adjusting term in the trust parameter definition, as follows:

$$R(< Org_j, \mathcal{D} >) = \frac{\sum_i \mathcal{C}_{i,j,\mathcal{D}} + \sum_i \mathcal{C}_{j,i,\mathcal{D}}}{\sum_i n_{i,j,\mathcal{D}} + \sum_i n_{j,i,\mathcal{D}}} \quad \forall Org_i \in \mathcal{O}$$

The second term of the numerator penalizes organizations requiring data they already have to other organizations and firing a large number of complaints. The term at the denominator corresponds to the term introduced to take into account the number of interactions.

Bias Elimination. For the $R(\cdot)$ computation the following rule is adopted. If an organization has fired a very high percentage of complaints against a data unit then we do not consider them in $R(\cdot)$ calculation. In fact, this situation is due either to strict requirements on data quality or to malicious behavior. The fraction complaints-interactions, called $F(c, i)$, is a normal random variable. If we focus on the $F(c, i)$ values of an organization, we first calculate the parameters (i.e. the mean and the standard deviation) of the normal distribution of the other organization $F(c, i)$ values. Then, if $F(c, i)$ is higher than the mean value plus two times the standard deviation, the organization's complaints are not included in the $R(\cdot)$ calculation.

2.4.2 Trust-based query processing in P2P Systems: Piazza

Piazza deals with the problem of sharing semantically heterogeneous data in a distributed and scalable way. The participants in Piazza are data sources interested in sharing data. They prefer independent but related schemas for their data, and their queries will typically be posed from the context of their preferred schema. Rather than requiring global agreement on a single unified schema, query answering capabilities over an arbitrary network of local schemas and pairwise mappings between them are provided. These query answering algorithms take a query posed over any of these schemas and use the transitive closure of mappings to return all relevant data in that preferred schema. This achieves the schema mediation capabilities of a data integration system, but in a more extensible, decentralized way. As in peer-to-peer data sharing systems, any node is allowed to join the system and to contribute resources (schemas, mappings, data, or computation) that improve the overall environment. This architecture is referred to as a *peer data management system (PDMS)* and a node as a *peer*.

The authors introduced the notion of *peer data management system (PDMS)* which is now widely adopted.

Schema Mediation In contrast to a data integration environment, which has a tree-based hierarchy with data sources schemas at the leaf nodes and one or more *mediated schemas* as intermediate nodes, a peer data management system (PDMS) can support an arbitrary *graph* of interconnected schemas. Some of these schemas are defined virtually for purposes of querying and mapping. These are called *peer schemas*, and generally their relations *peer relations* will have an open-world assumption (i.e., the data returned by querying these relations may be incomplete). Queries in the PDMS will be posed over the relations from a specific peer schema. A peer schema represents the peers view of the world that is unlikely to be the same at different peers. Peers may also contribute data to the system in the form of *stored relations*. Stored relations are analogous to data sources in a data integration system: all queries in a PDMS will be reformulated strictly in terms of stored relations that may be stored locally or at other peers.

There are two types of schema mappings in Piazza. A mapping that relates two or more peer schemas is called a *peer description*, whereas a mapping that relates a stored schema to a peer schema is called a *storage description*. Peer descriptions define the correspondences between the views of the world at different peers. Storage descriptions, on the other hand, map the data stored at a peer into the peers view of the world. Thus, storage descriptions are similar to data source descriptions in a data integration system. Two main formalisms have been proposed for schema mediation in data integration systems. In the first, called *global-as- view* (GAV) [28], the relations in the mediated schema are defined as views over the relations in the sources. In the second, called *local-as-view* (LAV) [43], the relations in the sources are specified as views over the mediated schema. Piazza combines and generalizes the two data integration formalisms, and it extends them to the XML world in a way that keeps evaluation tractable. Two kinds of peer descriptions are supported: *equality and inclusion* descriptions. Peer descriptions have the following form: $[Q_1(P_1) = Q_2(P_2), (or [Q_1(P_1) \subseteq Q_2(P_2)\ for\ inclusions)]$ where Q_1 and Q_2 are conjunctive queries with the same arity and P_1 and P_2 are *sets* of peers. Intuitively, this

mapping statement specifies a semantic mapping by stating that evaluating Q_1 over the peers P_1 will always produce the same answer (or a subset in the case of inclusions) as evaluating Q_2 over P_2. To avoid ambiguity, relation names are prefixed at each peer with the peer name. The following statement is an example of an equality mapping between two peers: `UW` (University ofWashington) and `DBProjects` (a Database-Projects peer):

```
DBProjects:Member(pName, member) = UW:Member(mid, pid, member), UW:Project(pid,
pName)
```

A storage description can also be either an equality ($P : R = Q$) or an inclusion ($P : R \subseteq Q$); here Q is a query over the schema of peer P and R is a stored relation at the peer. As in the context of data integration, an inclusion description implies that that the data at the peer may be *incomplete*, which corresponds to the *open world assumption* [6]. The following storage description defines the stored relation `students` at peer `UPenn` in terms of `UPenn`s peer relations:

```
UPenn:student(sid, name, advisor) ⊆ UPenn:Student(sid, name),
UPenn:Advisor(sid, fid), UPenn:Faculty(fid, advisor)
```

The set of mappings of a PDMS defines its *semantic network* (or *topology*). Optimizing the topology of a PDMS is an interesting research problem. Some of the possible optimization criteria include: eliminating redundant mappings, reducing the diameter of a PDMS (to reduce information loss in query reformulation), and identifying semantically unreachable peers. Analyzing the semantic network of a PDMS requires the ability to compose mappings which is a challenging problem on its own.

Querying Query reformulation is perhaps the single most important aspect of query processing in a PDMS, since it is crucial for PDMSs ability to answer user queries. In this section, we outline the query reformulation algorithm implemented in Piazza. The input of the algorithm is a set of peer mappings and storage descriptions and a query Q. The output of the algorithm is a query expression Q' that refers to stored relations *only*. To answer Q we need to evaluate Q' over the stored relations. Recent techniques for adaptive query processing [33] are well suited for evaluating Q' in such a context.

Before describing the algorithm, two points need to be mentioned. First, by introducing an auxiliary relation (view) a description of the form $Q1(P1) = Q2(P2)$ can be rewritten as two simpler descriptions: $Q1(P1) = V$ and $V = Q2(P2)$. Second, an equality description can be rewritten a pair of inclusion descriptions. Hence, we can assume that all descriptions are of the form $V \subseteq Q(P)$ or $V \supseteq Q(P)$.

To provide some intuition for the algorithm, consider a PDMS in which all peer mappings are of the form $V \supseteq Q(P)$. This case is similar to unfolding GAV mappings in data integration. The algorithm proceeds by constructing a simple rule-goal tree [78]: goal nodes are labeled with atoms of the peer relations, and rule nodes are labeled with peer mappings. Let's begin by expanding each query subgoal according to the relevant peer mappings in the PDMS. When none of the leaves of the tree can be expanded any further, it's possible to use the storage descriptions for the final step of reformulation in terms of the stored relations.

At the other extreme, suppose all peer mappings in the PDMS are of the form $V \subseteq Q(P)$. In this case (that is similar to LAV mappings in data integration), let's begin with the query subgoals and apply an algorithm for answering queries using views [31]. The algorithm is applied to the result until it's impossible to proceed further, and as in the previous case, the storage descriptions for the last step of reformulation is used.

A major challenge of the reformulation algorithm is to combine and interleave the two types of reformulation techniques. One type of reformulation (unfolding) replaces a subgoal with a set of subgoals, while the other (rewriting) replaces a set of subgoals with a single subgoal.

Other challenges that we address when constructing a reformulation DAG in Piazza are avoiding redundant work through memoization and pruning [32] and choosing an optimal reformulation order.

3 P2P Overlay Networks

An *overlay network* is, as the name implies, a communication network built on top of another communication network [48]. An overlay network takes charge of various aspects like: basic connectivity among participants, message routing, management of participants joining or leaving the network, faults management.

The justification to these structures comes from the peculiar characteristics of the underlying infrastructure: the Internet. Due to its implicit unreliability, the TCP-IP network constituting Internet is unable to directly address the needs of a wide range of various modern applications that require fast, reliable and complex communication primitives. Due to the practical difficulties that an extension of the TCP/IP protocols would imply, the researcher efforts moved toward a layered approach where the existing network is left in its current state, and on top of it new algorithms and protocols are deployed in order to maintain "virtual" networks and provide through them the required communication primitives.

Overlay networks for object storage and retrieval in a completely distributed fashion provide two fundamental primitives:

- A *put* function: to store an object somewhere in the overlay
- A *get* function: to locate and retrieve an object previously stored.

These fundamental primitives are at the basis of a global storage infrastructure.
Three aspects characterize an overlay network:

- How the overlay is built and maintained. This implies all the algorithms used to manage new nodes joining the network, old nodes leaving it and node faults.
- Which strategy is exploited to store objects in the network, i.e. on which node an object is physically stored once a *put* operation is invoked.
- Which strategy is used to locate and retrieve objects through the *get* primitive.

Basing on how these aspects are treated, we can distinguish between two groups of overlay network systems: *structured systems* and *unstructured systems*.

Structured systems build and maintain some distributed data object structure. This structure is then used both to organize nodes in the overlay and to place stored objects. All these overlay networks organize nodes in a precise structure. Node joining the system must follow a defined join algorithm to become part of the overlay. This algorithm is able to organize connections between nodes in order to build, in a distributed fashion, complex regular structures. Various types of graphs are employed: besides rings [76, 63] and their derivatives, we can encounter d-dimensional thoroidal spaces [60], general meshes [81], butterflies [46], De Bruijn graphs [53], etc. Note that, the usage of a specific structure for node connections is not necessarily a prerequisite for the implementation of specific algorithms for *put* and *get* primitives. Nevertheless, algorithms can leverage structures to increase overall performance.

Object placement is usually executed just using the structure previously deployed. The implementation of the *put* primitive consists of an algorithm which selects in which point (node) of this structure the object will be placed. Given this placement strategy the *get* primitive can be implemented exploiting some high performance traversal technique appropriate for the chosen structure.

Structured overlay networks, in their various forms, are by far the most used infrastructures for global data object storage systems. Systems exploiting such overlays are, for example, CFS [17], PAST [64], Glacier [30], Oceanstore [10], P-Grid [4] and Ivy [52].

Unstructured systems approach the problem from a completely different perspective: their goal is to maximize independence and self-organization among nodes; for this reason systems based on this approach usually employs random algorithms for overlay network's construction and maintenance. Actually, the unstructured approach did not attained a wide success in the area of global data object storage (with the noteworthy exception of the Pangaea [69] system).

In [16] is underlined the problem of queries efficiency: it's stated that for *point queries*[1] good performances can be obtained using P2P systems based on placement of contents at nodes using hash functions. For other types of queries, such *range queries* or *text queries*, this kind of overlay is no so good and then a *Semantic Overlay Network* (SON) appears more suitable.

3.1 Overlay Networks supporting point queries

In this section are showed the insights of four overlay networks, namely Chord, Tapestry, Pastry and P-Grid. The systems here introduced allow to highlight in a clear manner basic characteristics proper of any structured overlay network, since they are simple but represent a wider range of more complex and sophisticate systems.

3.1.1 Chord

Chord [76] is the implementation of a *distributed hash table* (DHT). It is able to create and maintain a distributed abstract space, called *key space*, where objects can be easily stored and retrieved, exactly as in a standard centralized hash table.

The *key space* is a sequence of ordered locations, each identified by a key, that constitute a ring-shaped sequence (i.e. the location with the smallest key follows the one with the larger key); keys are represented by integer numbers chosen in a predefined interval $[0,2^h]$ with $h \in N$. h is a predefined constant used to control the size of the *keyspace*, i.e. the maximum number of locations it contains. The *key space* is continuous, i.e. does not contain holes: a location exists for each possible key; this is achieved in Chord decoupling the *key space* from the set of nodes that effectively maintains this space. Each node constituting the Chord network is represented in the *key space* by a *key* that is usually calculated applying a consistent hash function [40], namely *h(x)*, to the node's IP address; this key is defined as the node's *id*.

Keys are assigned to nodes in a straightforward way: each location, identified by its key k, is stored on the first node whose identifier *id*, is equal to or follows k in the key space. This node is called the *successor node* of key k, and is denoted by *successor(k)*.

In Figure 13 a key space (h=4) populated by three nodes (id=1,5,10) is represented. If we consider the location identified by key 7, it is physically stored in the node with id 10, i.e. its successor.

Each node n maintains a table with h entries, called the *finger table*. The i-th entry in the table at node n contains the identity (i.e. the IP address) of the first node n' whose id follows *n's id* by at least 2^{i-1} on the key space, i.e. *n'=successor(id+*2^{i-1}*)*, where $1 \leq i \leq h$ (and all arithmetic is modulo 2^h). We call node n' the *i-th finger* of node n. Figure 13 reports the finger tables of the three nodes that constitute the network. Dotted grey lines departing from each node represent keys pointed by the respective fingers.

The *put* primitive in Chord is called *insert(key, value)* and take two parameters: a valid key and the object that we want to store on the location corresponding to that key. The key is used to globally identify the object and is thus usually obtained applying the hash function to a globally known and unique property of the object (e.g. its name). The implementation of

[1] query where the search key is known exactly

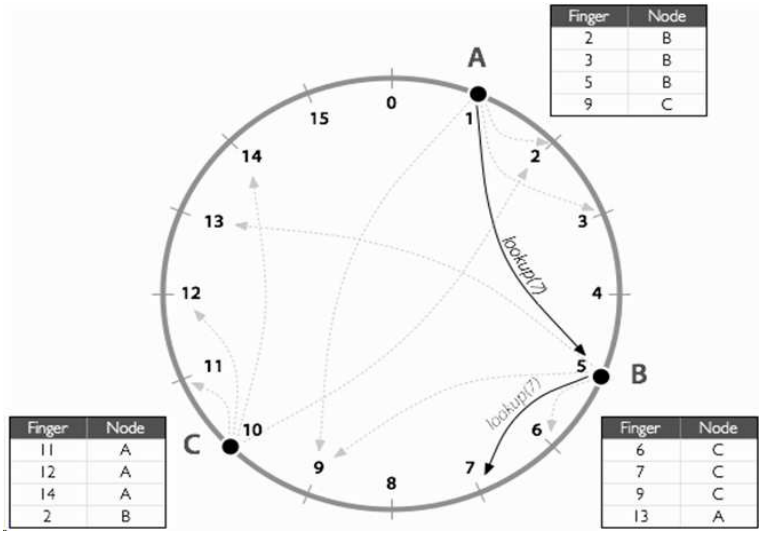

Figure 13: Chord Network

insert is based on another function: *lookup(key)*. The sole purpose of *lookup* is to return the node that stores *key*. When a node n executes *lookup(k)*, and k does not appear in its finger table, i.e. n does not know *successor(k)*, it forwards the message to another node in the network, closer than itself to the desired key. To accomplish this task, n searches its finger table for the closest finger preceding k, and forwards the message to that node: intuitively, each step toward the destination halves the distance to it; this means that, with high probability, the number of nodes that must be visited by a message to reach its destination in a M-node network is O(log M). Let us explain this through an example based on the small Chord network depicted in Figure 13. Suppose *lookup(7)* is executed on the node with *id* 1, i.e. node A. This node does not have a finger pointing directly to key 7, thus it looks in its finger table and forwards the *lookup(7)* query to the node associated with the 3^{rd} finger pointing toward key 5, i.e. node B. As soon as node B receives this query it looks in its finger table and finds a finger pointing directly toward key 7. The node associated with this key is the one with *id*=10, i.e. C, actually the successor of key 7. The message will be thus forwarded to node C, its final destination.

The implementation of the *insert* function is straightforward through this powerful *lookup* function. The *lookup* function is also used to realize the *get* primitive, whose implementation is also rather obvious. Chord also includes smart algorithms used to dynamically handle nodes joining or leaving the overlay network. The reader can refer to [76] for the details. The Chord overlay network is used as a distributed storage service by CFS [17] and Glacier [30].

3.1.2 Tapestry

Also Tapestry [81] is the implementation of a DHT, but it uses a completely different overlay structure. In fact it is mostly based on the routing mechanism introduced by Plaxton et al. in [58].

The work in [58] proposed a mesh-like structure. Each node in the system is assigned an *id* (it is always possible to apply a hash function on the node's IP address). The ids are then used to construct a mesh among nodes, as shown in Figure 14 (where *ids* are expressed with hexadecimal values). In this figure, each link is labeled with a level number that denotes the stage of routing that uses this link. The *i-th* level neighbor-links for some node n point at the 16 closest neighbors whose *ids* match the lowest *i-1* nibbles of node n's *id* and who have different combinations of the *i-th* nibble; If a link cannot be constructed because no such node meets the

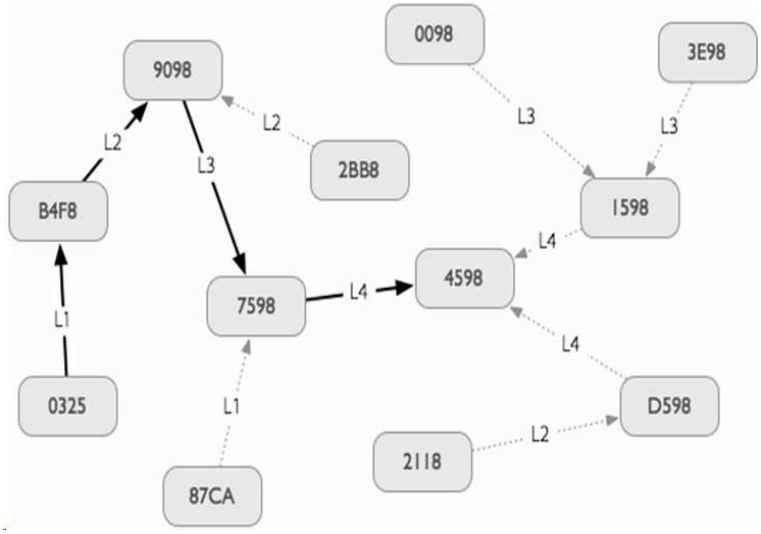

Figure 14: Tapestry Mesh

proper constraints, then the scheme chooses the node that matches the constraints as closely as possible. This process is repeated for all nodes and levels within a node.

The key mechanism of this approach is that links form a series of random embedded trees, with each node representing the root of one of these trees. As a result, the neighbor links can be used to route lookup requests from anywhere to a given node, simply by resolving the node's address one link at a time: first a level-one link, then a level-two link, etc. Figure 14 shows how a lookup request for key 4958 can be routed in this way from node 0325 to node 4598.

This structure can be used to store objects in a structured manner, just generating an identifier for each object in the same shape of node's *ids*. Then each object is mapped to the single node whose *id* matches the object's identifier in the most bits (starting from the least significant); The improvements added by Tapestry to [58] are the ability to allow dynamic join and leave of nodes, and failure tolerance. Tapestry is part of the Oceanstore [10] global storage system.

3.1.3 Pastry

Pastry [63] is the third implementation of a distributed hash table we explore in this chapter. It actually mixes techniques seen both in Chord and in Tapestry.

Like Chord, also Pastry exploits a ring shaped abstract space for keys, but differently from it, Pastry can exploit a proximity metric to increase performance and even includes a mechanism to tolerate node failures. Each Pastry node maintains a *routing table*, a *neighborhood set* and a *leaf set*. A node's routing table R is organized into $\lceil log_B N \rceil$ (with B=2^b) rows with 2^b-1 entries each. Entries at row n of R each refers to a node whose *id* shares the present node's *id* in the first n digits, but whose $(n+1)^{th}$ digit has one of the possible 2^b-1 values other than the $(n+1)^{th}$h digit in the present node's *id*. The *leaf set* is populated with the L nodes whose *ids* are the closest to n's *id* in the key ring, while the *neighborhood set* contains L nodes close to n with respect to a predefined proximity metric (L is a parameter of the system).

The leaf set is used by Pastry to ensure ring connectivity and correct behavior of *lookup* operation besides failure of $\lfloor L/2 \rfloor$ nodes with adjacent *ids*. The *neighborhood set* is instead used each time a node must be added to the routing table, in order to select it in a group containing only "close" nodes with respect to a predefined proximity metric.

The *lookup* function is realized in Pastry through an algorithm that is quite similar to

the one employed by Tapestry. Even in this case request routing proceeds with consecutive approximations of the searched key with node *ids*: at each step the algorithm forwards the request to a node whose id matches the searched key for at least one more bit, with respect to the current node *id*. Through this mechanism Pastry is able to route messages in at most a logarithmic number of hops, with respect to the total number of nodes. The Pastry overlay network is used as a storage substrate by PAST [64].

3.1.4 P-Grid

P-Grid [4] philosophy slightly departs from what we have seen from Chord, Pastry and Tapestry. It implements a distributed binary search tree that can be built and maintained over any overlay communication infrastructure, either structured or unstructured.

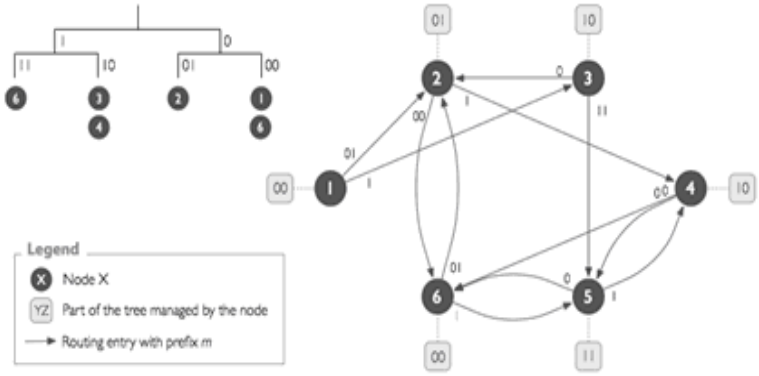

Figure 15: A binary search tree (left) along with its P-Grid implementation (right)

In P-Grid each node, identified as usual by a binary id, maintains only part of the overall tree, which comes into existence only through the cooperation of individual peers. Every node's position is determined by its path, that is, the binary string representing the subset of the tree's overall information that the node is responsible for. For example, the path of node 4 in Figure 15 is 10, so it stores all data object items whose key begins with 10. The paths implicitly partition the search space and define the structure of the virtual binary search tree. As Figure 15 shows, multiple nodes can be responsible for the same path. Node 1 and Node 6, for example, both store keys beginning with 00. Such replication improves P-Grid's robustness and responsiveness.

P-Grid's approach to routing is simple and efficient: for each bit in its path, a node stores the address of at least one other node that is responsible for the other side of the binary tree at that level. Thus, if a node receives a lookup request for a key it cannot directly satisfy, it must forward the request to a peer that is "closer" to the result. Suppose that node 6 in Figure 15 executes *lookup(100)*. That node is not responsible for that key so it makes a lookup in its routing table for an entry related to keys starting with bit 1. The routing table in node 6 contains a pointer to node 5, thus the request is forwarded to this site (whose *id* is "closer" to the searched key). Node 5 looks at the next bit of the searched key and note that it is a 0, while its *id* is actually 11. This means that node 5 cannot handle the searched key. Looking in its routing table it finds a pointer to node 4 whose id starts with bits 10. The lookup request is then forwarded to node 4 that is actually the final destination. The P-Grid construction algorithm, whose details can be found in [4], guarantees that node routing tables always provide at least one path from any node receiving a request to one of the nodes holding a replica so that any query can be satisfied regardless of the node queried.

3.2 Overlay Networks supporting other type of queries: Semantic Overlay Networks

A "Semantic Overlay Network (SON) is a flexible network organization that improve query performance while maintaining a high degree of node autonomy. With Semantic Overlay Networks (SONs), nodes with semantically similar content are clustered together" [16].

In such systems, queries produce a result doing matching between the "object" of the query and the SON identified to give an answer; this result is produced forwarding query through the nodes that belong to that SON. In this way, every query will be addressed by nodes belonging to specific SON without involve nodes that haven't got the desidered content. This approach, unfortunately, creates some challenges like

- how nodes and queries shuold be classified
- how is the right level of granularity for the classification
- when a node should join a SON
- how to choose SONs to use to answer a query

3.2.1 CGM

CGM provides some mechanisms to response to some of this challenges; for example classification is done in an hierarchical way and each document and query is classified in one or more leaf concepts. In the following it's briefly described the process of building and using SONs used in [16].

First, it's evaluation of potential classification hierarchies, using the actual data distributions in the nodes (or a sample of them), to find a good hierarchy. When it agrees on hierarchy, this will be stored to some nodes of the system and will be used to define the SONs.

When a node wants to join the system, first floods the network with requests for the hierarchy (in a Gnutella [29] fashion) and then, runs a document classifier, based on the hierarchy obtained on all its documents, that assigns the node to specific SONs. When a node knows which is the right SONs where it can be inserted, it should contact some other nodes that belong yet to SON, i.e. flooding the network until nodes in that SON are found or by using a central directory

When the node issues a query, first it classifies it and sends it to the appropriate SONs; after the query is sent to the appropriate SONs, nodes within the SON find matches by using some propagation mechanisms.

3.2.2 JNC

In [34] the authors propose an algorithm to build semantic overlay and one other to realize searching in semantic overlay and evaluate them in a "small world". The basic idea is consider two type of link that can be used to built the semantic overlay and to "route" queries. First it considers a *short-range* links which are semantically similar to the node and second *long-range* links that help to increase recall rate of information retrieval, reducing network traffic. To join the semantic network is required to compute the *semantic summarization*, that is the fraction of documents belonging to each topic, and the *semantic similarity*, that is the mesure of semantic similarity between two peers. For any further information consult [34].

3.2.3 GridVine

GridVine [2] addresses other two main problems such that

- building scalable semantic overlay networks

- give consistent query answer and reconcile different mappings occouring in the SON

The approach used, divides logical layer from physical one: the physical layer consists of a structured p2p overlay network, that efficiently routing messages, while logical layer realizes the semantic overlay for managing and mapping data and meta-data schema. Therefore GrigVine provid support for *Semantic gossiping*, a specific schema reconciliation technique used by authors in their system and descibed in detail in [2]. The authors propose also an architecture and an implementation using potential of structured overlay networks to build interoperable large-scale semantic overlay networks. Implementation is provided within the standard syntactic framework of RDF/OWL and is based on P-Grid.

3.2.4 DESENT

DESENT is an unsupervised method for decentralized and distributed generation of SONs in p2p context.

The authors underline that one of the problems of SONs is the construction of these overlays, assuming the lack of knowledge of both global content and network topology. In a P2P architecture, each peer is initially aware only of its neighbors and their content. Thus finding other peers with similar contents to form a SON, becomes a tedious problem.

[22] shows a distributed and decentralized method for SON construction namely DESENT. DESENT provides an efficient mechanism for web search in unstructured P2P networks with good characteristics, including guarantees to find the answer of a query in less than flooding time.

The strategy for creating SONs is based on clustering peers by their content similarity. This is achieved by a recursive process that starts on the individual peers themselves. Through applying a clustering technique on the documents stored at the site, one or more feature vectors are created for each web site, essentially one for each topic a site covers. Then representative peers, each responsible for a number of peers in a zone, are selected. These peers will collect the feature vectors from the members of the zone and use these as basis for the next level of clustering. This process is applied recursively, until we have a number of feature vectors covering all available documents.

3.2.5 SOWES

SOWES [23] is a scalable approach to P2P web searching, similar for some aspects to DESENT, but with one substantial difference: SOWES is based on unstructured P2P networks.

SOWES employs semantic overlay networks (SONs), where peers containing related information are connected together in separate overlay networks; queries can be forwarded to only those peers containing documents that satisfy the constraints of the query context.

The main feature of SOWES is the unsupervised, distributed and decentralized generation of SONs. Peers that contain semantically related information become part of a logical cluster.

Each cluster is represented by one or more special peers which maintain connections to other cluster representatives. These special peers are responsible for forwarding queries to the most relevant clusters, avoiding the excessive cost of flooding the entire network.

3.2.6 PSEARCH

PSearch is a decentralized non-flooding P2P information retrieval system. In pSearch is used an extensions of VSM and LSI [7, 20] to generate the semantic space, and is used CAN [60] to organize nodes into overlay.

In this context a semantic overlay is a logical network where contents are organized around their semantics such that the distance (e.g., routing hops) between two documents in the network is proportional to their dissimilarity in semantics.

pSearch distributes document indices through the P2P network following document semantics generated by Latent Semantic Indexing (LSI). VSM and LSI represent documents and queries as vectors in a Cartesian space, and measure the similarity between a query and a document as the cosine of the angle between their vector representations.

CAN is used to create a semantic overlay by exploiting semantic vector (generated by LSI) of a document as the key to store the document index in CAN.

To find documents relevant to a query, it's only needed to compare the query against documents within a small region centered at the query, because the relevance of documents outside the region is relatively low. By doing this, the search space for the query is effectively limited and the accuracy is retained.

3.3 Overlay Networks for supporting trust

Peer to Peer (P2P) networks have open and decentralized nature that makes them extremely susceptible to malicious users spreading harmful content like viruses, trojans or, even just wasting valuable resources of the network. In order to minimize such threats, the use of community-based reputations as trust measurements is becoming mandatory. The idea is to dynamically assign each peer with a *trust rating* based on its performance in the network and to store it at a suitable place [74].

In general, a high *trust value* indicates that peers gained good reputation in terms of its past performances and thus they are more trustworthy; contrarily a low trust value means the peers had relatively poor quality of service (QoS) in the past and are rated with low reputations by other peers in the community.

Many models have been proposed for trust in p2p systems ([15, 73, 74, 3, 75]); in the following we discuss three of them.

3.3.1 TrustMe

TrustMe is an anonymous and secure protocol for maintaining and accessing trust rating information [74]. TrustMe uses Public Key cryptography schemes to provide security and it is resistant to various attacks. This system follows the user-based approach, that is a peer gives a reputation rating to another peer based on their interactions.

The authors focus their attention on the need of *anonymity* for two main reasons:

- to preserve peers reporting poor trust values from malicious attacks;
- to mantain anonymity while querying for another peer's trust value

To this aim, *TrustMe* places a trust rating of each peer at a random Peer X. A peer can anonymously issues a query and gets the true value without needing to know the identity of Peer X. Also a single reply message from Peer X is enough to make a decision.

3.3.2 SBWS Trust Framework

The framework is realized starting from the observation that "often lack global view of peers behavior making decision regarding reputation updates based on current actions only" [75]. Then it considers current peers' action within their usual behaviour.

It defines an *anomaly* as any event that does not fit the normal behavioral profile of the peer and it is, thus, indicative of peer's unpredictability and unreliability.

Based on datamining techniques, it's possible to analyze peers' session data to underline every abnormal hidden behaviour and to estimate change to peer's *reputation value*.

3.3.3 SUP Trust Management system

[73] proposes a reputation-based, distributed trust architecture for P2P networks to identify malicious peers and to prevent the spreading of malicious content.

The protocol is based on the query-response architecture and it is suitable for operation in Gnutella like systems.

The aim of the protocol is to distinguish between malicious responses and benign ones by using the reputation of the peers. Every time that a transaction takes place, the result is written on some vector to make possible to calculate the *trust rating*.

References

[1] A. Abdul-Rahman and S. Hailes. Supporting Trust in Virtual Communities. In *Proceedings of the Hawaii International Conference on System Sciences*, Maui, Hawaii, January 2000.

[2] K. Aberer, P. Cudre-Mauroux, M. Hauswirth, and T. van Pelt. GridVine: Building Internet-Scale Semantic Overlay Networks. In *In International Semantic Web Conference (ISWC)*, 2004.

[3] K. Aberer and Z. Despotovic. Managing Trust in a Peer-2-Peer Information System. In *Proceedings of the Tenth International Conference on Information and Knowledge Management (CIKM01)*, 2001.

[4] K. Aberer, M. Punceva, M. Hauswirth, and R. Schmidt. Improving Data object Access in P2P Systems. In *IEEE Internet Computing*, pages vol. 6 issue 1 pp. 58–67, 2002.

[5] Karl Aberer and Zoran Despotovic. Managing trust in a peer-2-peer information system. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 310–317, Atlanta, Georgia, USA, 2001.

[6] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *PODS (1998)*.

[7] M. Berry, Z. Drmac, and E. Jessup. Matrices, Vector Spaces, and Information Retrieval. In *SIAM Review*, pages 41(2):335–362, 1999.

[8] L. Berti-Équille. Quality-Adaptive Query Processing over Distributed Sources. In *Proc. 9th Internation Conference on Information Quality (IQ 2004)*.

[9] L. Berti-Équille. Integration of Biological Data and Quality-driven Source Negotiation. In *Proc. ER 2001*, Yokohama, Japan, 2001.

[10] D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao, and J. Kubiatowicz. OceanStore: An Extremely Wide-Area Storage System. In *Technical Report UCB/CSD-00-1102, University of California at Berkeley, Computer Science Division*, 2000.

[11] Amir Boag, Donald Chamberlin, Mary F. Fernandez, Daniela Florescu, Jonathan Robie, and Jerome Simèon. XQuery 1.0: An XML Query Language. W3C Working Draft. Available from `http://www.w3.org/TR/xquery`, November 2003.

[12] V. Cahill, B. Shand, and E. *et al* Gray. Using Trust for Secure Collaboration in Uncertain Environments. *Pervasive Computing*, pages 52–61, July-September 2003.

[13] M. Carbone, M. Nielsen, and V. Sassone. A formal model for trust in dynamic networks. In *In Proc. of International Conference on Software Engineering and Formal Methods (SEFM'03)*, Brisbane, 2003.

[14] Abraham Charnes, William W. Cooper, and Edwardo Rhodes. Measuring the Efficiency of Decision Making Units. *European Journal of Operational Research*, 2, 1978.

[15] R. Chen and W. Yeager. Poblano: A distributed trust model for peer-to-peer networks. In *Technical report, Sun Microsystems*, 2003. http://www.jxta.org/docs/trust.pdf.

[16] A. Crespo and H. Garcia-Molina. Semantic overlay networks for P2P systems. In *Technical report, Computer Science Department, Stanford University*, October 2002.

[17] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, 2001.

[18] Ernesto Damiani, De Capitani di Vimercati, Stefano Paraboschi, Pierangela Samarati, and Fabio Violante. A reputation-based approach for choosing reliable resources in peer-to-peer networks. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 207–216, Washington DC, USA, 2002.

[19] Umeshwar Dayal. Query Processing in a Multidatabase System. In *Query Processing in Database Systems*, pages 81–108. Springer, 1985.

[20] S.C. Deerwester, S.T. Dumais, T.K. Landauer, G.W. Furnas, and R.A. Harshman. Indexing by Latent Semantic Analysis. In *Journal of the American Society of Information Science*, pages 41(6):391–407, 1990.

[21] Jay L. Devore. *Probability and Statistics for Engineering and the Sciences*. Duxbury Press, 5th edition edition, December 1999.

[22] C. Doulkeridis, K. Nø rvåg, and M. Vazirgiannis. DESENT: Decentralized and Distributed Semantic Overlay Generation in P2P Networks. In *Technical report, AUEB*, 2005. http://www.db-net.aueb.gr/index.php/ publications/technical reports/.

[23] C. Doulkeridis, K. Nø rvåg, and M. Vazirgiannis. The SOWES approach to P2P web search using semantic overlays. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, 2006.

[24] R. Falcone and C. Castelfranchi. *Social Trust: A Cognitive Approach*. Kluwer, 2001.

[25] Weiguo Fan, Hongjun Lu, Stuart Madnick, and David Cheungd. Discovering and Reconciling Value Conflicts for Numerical Data Integration. *Information Systems*, 26(8), 2001.

[26] Ariel Fuxman, Elham Fazli, and René J. Miller. ConQuer: Efficient Management of Inconsistent Databases. In *Proc. SIGMOD 2005*, Baltimore, MA, 2005.

[27] D. Gambetta. Can We Trust Trust? In *Trust: Making and Breaking Cooperative Relations*, pages 213–238. Basil Blackwell, Oxford, 1990.

[28] H. Garcia-Molina, D. Papakostantinou, A. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. *Journal of Intelligent Information Systems*, 8(2), March 1997.

[29] Gnutella. http://www.gnutella.com.

[30] A. Haeberlen, A. Mislove, and P. Druschel. Glacier: Highly durable, decentralized storage despite massive correlated failures. In *Proceedings of the 2ndt USENIX Symposium on NetworkedSystems Design and Implementation*, 2005.

[31] A. Halevy. Answering queries using views: a survey. *VLDB Journal*, 10(4), 2001.

[32] A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov. Schema Mediation in Peer Data Management Systems. In *ICDE (2003)*.

[33] Z. Ives, A. Halevy, and D. Weld. Integrating Network-Bound XML Data. *IEEE Data Engineering Bulletin*, 24(2), 2001.

[34] H. Jin, X. Ning, and H. Chen. Efficient search for peer-to-peer information retrieval using semantic small world. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, 2006.

[35] A. Jø sang. Trust-based decision making for electronic transactions. In *L. Yngstromand T. Svensson, editors, Proceedings of the 4th Nordic Workshop on Secure Computer Systems (NORDSEC'99)*, Stockholm University, Sweden, 1999.

[36] A. Jø sang. A Logic for Uncertain Probabilities. *International Journal of Uncertainty,Fuzziness and Knowledge-Based Systems*, 9(3):279–311, June 2001.

[37] A. Jø sang and R. Ismail. The beta reputation system. In *Proceedings of the 15th Bled Electronic Commerce Conference*, Bled, Slovenia, June, 2002.

[38] A. Jø sang and S. Pope. Semantic constraints for trust tansitivity. In *Proceedings of the Asia-Pacific Conference of Conceptual Modelling (APCCM)*, Newcastle, Australia, February, 2005.

[39] S.D. Kamvar, M.T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the Twelfth International World Wide Web Conference*, Budapest, May 2003.

[40] D. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, 1997.

[41] Won Kim and Jungyun Seo. Classifying Schematic and Data Heterogeneity in Multidatabase Systems. *IEEE Computer*, 24(12):12–18, 1991.

[42] R. Levien. *Attack Resistant Trust Metrics.* PhD thesis, University of California at Berkeley, 2004.

[43] A. Levy, A. Rajaraman, and J. Ordille. Querying Heterogeneous Sources Using Source Description. In *VLDB (1996).*

[44] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering Queries Using Views. In *Proc. PODS 1995*, San Jose, CA, 1995.

[45] E. P. Lim and Roger H.L. Chiang. A Global Object Model for Accommodating Instance Heterogeneities. In *Proc. ER'98*, Singapore, 1998.

[46] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st annual ACM symposium on Principles of distributed computing*, 2002.

[47] D.W. Manchala. Trust metrics, models and protocols for electronic commerce transactions. In *Proceedings of the 18th International Conference on Distributed Computing Systems*, 1998.

[48] A. Milani, L. Querzoni, and S. Tucci Piergiovanni. Data Object Storage in Large Scale Distributed Systems. In *in GLOBAL DATA MANAGEMENT Baldoni et al. (editors)*, 2006.

[49] Amihai Motro and Philipp Anokhin. Fusionplex: Resolution of Data Inconsistencies in the Data Integration of Heterogeneous Information Sources. *Information Fusion*, 2005.

[50] L. Mui, M. Mohtashemi, and C. Ang. A probabilistic rating framework for pervasive computing environments. In *Proceedings of the MIT Student Oxygen Workshop (SOW'2001)*, 2001.

[51] L. Mui, M. Mohtashemi, and A. Halberstadt. A computational model of trust and reputation. In *Proceedings of the 35th Hawaii International Conference on System Science (HICSS)*, 2002.

[52] A. Muthitacharoen, R. Morris, T.M. Gil, and B. Chen. Ivy: A Read/Write Peer-to-Peer File System. In *Proceedings of 5th Symposium on Operating Systems Design and Implementation*, 2002.

[53] Naor, Moni and Wieder, Udi. Novel Architectures for P2P Applications: the Continuous-Discrete Approach. In *In ACM Symposium on Parallel Algorithms and Architectures*, 2003.

[54] Felix Naumann and Matthias Häussler. Declarative Data Merging with Conflict Resolution. In *Proc. 7th International Conference on Information Quality (IQ 2002).*

[55] Felix Naumann, Ulf Leser, and Johan C. Freytag. Quality-driven Integration of Heterogenous Information Systems. In *Proc. VLDB'99*, Edinburgh, UK, 1999.

[56] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998.

[57] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object Fusion in Mediator Systems. In *Proc. VLDB 1996*, Bombay, India, 1996.

[58] C. Plaxton, R. Rajaraman, and A. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of ACM SPAA*, pages 311–320, 1997.

[59] D.H. PMcKnight and N.L. Chervany. The Meanings of Trust. Technical report, Technical Report MISRC Working Paper Series 96-04, University of Minnesota, Management Information Systems Reseach Center, 1996. URL: http://misrc.umn.edu/wpaper/.

[60] S. Ratnasamy, P. Francis, H. Handley, R. Karp, and S. Shenker. A scalable content addressable network. *Proceedings of ACM SIGCOMM'2001*, 2001.

[61] P. Resnick and R. Zeckhauser. Trust Among Strangers in Internet Transactions: Empirical Analysis of eBay's Reputation System. *M.R. Baye, editor, The Economics of the Internet and E-Commerce, Advances in Applied Microeconomics*, 11, 2002.

[62] P. Resnick, R. Zeckhauser, J. Swanson, and K. Lockwood. The value of reputation on ebay: A controlled experiment, 2002. URL:http://www.si.umich.edu/ presnick/papers/postcards/.

[63] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *In Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms, vol. 2218*, 2001.

[64] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, 2001.

[65] Thomas L. Saaty. *The Analytic Hierarchy Process*. McGraw-Hill, 1980.

[66] J. Sabater and C. Sierra. Regret: A reputation model for gregarious societies. In *Proceedings of the 4th Int. Workshop on Deception, Fraud and Trust in Agent Societies, in the 5th Int. Conference on Autonomous Agents (AGENTS'01)*, pages 61–69, Montreal, Canada, 2001.

[67] J. Sabater and C. Sierra. Social regret, a reputation model based on social relations. In *SIGecom Exchanges*, pages 44–56, 2002.

[68] J. Sabater and C. Sierra. Reputation and social network analysis in multi-agent systems. In *Proceedings of the First Int. Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS)*, July 2002.

[69] Y. Saito, C. Karamanolis, M. Karlsson, and M. Mahalingam. Taming aggressive replication in the Pangaea wide-area file system. In *ACM SIGOPS Operating Systems Review*, volume 36 issue SI, 2002.

[70] Monica Scannapieco, Barbara Pernici, and Elizabeth M. Pierce. IP-UML: A Methodology for Quality Improvement based on IP-MAP and UML. In R. Y. Wang, E. M. Pierce, S. E. Madnick, and C. W. Fisher, editors, *Advances in Management Information Systems - Information Quality (AMIS-IQ) Monograph*. Sharpe, M.E., April 2005.

[71] Monica Scannapieco, Antonino Virgillito, Carlo Marchetti, Massimo Mecella, and Roberto Baldoni. The DaQuinCIS Architecture: a Platform for Exchanging and Improving Data Quality in Cooperative Information Systems. *Information Systems*, 29(7):551–582, 2004.

[72] Eike Schallehn, Kai U. Sattler, and Gunter Saake. Extensible and Similarity-Based Grouping for Data Integration. In *Proc. of the ICDE 2002*, San Jose, CA, 2002.

[73] A.A. Selcuk, E. Uzun, and M.R. Pariente. Reputation-Based Trust Management for P2P Networks. url: citeseer.ist.psu.edu/selcuk04reputationbased.html.

[74] A. Singh and L. Liu. TrustMe: Anonymuous Management of Trust Relationships in Decentralized P2P Systems. In *In Proceedings of The Third IEEE International Conference on Peer-to-Peer Computing*, September 2003.

[75] N. Stakhanova, S. Basu, J. Wong, and O. Stakhanov. Trust Framework for P2P Networks Using Peer-Profile Based Anomaly Technique. In *ICDCSW '05: Proceedings of the Second International Workshop on Security in Distributed Computing Systems (SDCS) (ICDCSW'05)*, 2005.

[76] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications. *To Appear in IEEE/ACM Transactions on Networking*, August 2001.

[77] J. Schneider *et al.* Disseminating trust information in wearable communities. In *Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing (HUC2K)*, September 2000.

[78] J. Ullman. *Database and Knowledge-Base Systems (volume 2)*. Addison-Wesley, 1989.

[79] A. Withby, A. Jsang, and J. Indulska. Filtering out unfair ratings in bayesian reputation systems. In *Proceedings of the 7th Int. Workshop on Trust in Agent Societies (at AAMAS'04)*, ACM, 2004.

[80] Ling L. Yan and T. Ozsu. Conflict Tolerant Queries in AURORA. In *Proc. CoopIS'99*, Edinburgh, UK, 1999.

[81] S.Q. Zhuang, B.Y. Zhao, A.D. Joseph, R. Katz, and J. Kubiatowicz. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical report, University of California at Berkeley, Computer Science Division, 2001. UCB Tech. Report UCB/CSD-01-1141.

[82] C.N. Ziegler and G. Lausen. Spreading activation models for trust propagation. In *Proceedings of the IEEE International Conference on e-Technology, e-Commerce, and e-Service (IEEE'04)*, Taipei, March 2004.