Static Type-Inference for Trust in Distributed Information Systems

Premkumar T. Devanbu, Michael Gertz, and Brian Toone

Department of Computer Science
University of California at Davis
Davis, CA 95616, U.S.A.
{devanbu,gertz,toone}@cs.ucdavis.edu

Abstract. Decision-makers in critical fields such as medicine and finance make use of a wide range of information available over the Internet. Mediation, a data integration technique for distributed, heterogeneous data sources, manages the complexity and diversity of the information schemas on behalf of clients. We raise here the issue of *trust*: is the information so obtained trustworthy? Each client can have different perspectives on the desired trustworthiness the information he or she needs. We consider here the *scaling problem* that arises from a very large number of users accessing information from many different sources. A mediator cannot be expected to manage the potentially quadratic scaling of trust relationships clients can have with information sources. Furthermore, the possibility of using untrustworthy data increases the risk that the resulting data will be unacceptable: a mediator might evaluate a complex query for a client, only to have the answer rejected because the client does not trust the sources of the information.

To help address these issues, we introduce a general *static trust-typing* model, which can infer the trust ratings of query plans, based on trust meta-data about the input data to the query, even before executing the query. We also define essential properties of such a trust-typing model, namely *correctness, precision and completeness*. We present an example of a trust-typing model and describe some algorithmic frameworks for the use of such trust-typing models in mediator-based query evaluation.

1 Introduction

In critical fields such as medicine, finance, environmental protection, and national defense, poorly-informed decisions can have unacceptable negative consequences. Thus, prompt access to the widest possible set of information sources is critical. A vast amount of information (e.g., drug interaction data, genomic data, ethnographic information, company financial histories, geophysical data) is becoming available over the Internet, and decision makers can benefit from unified access to this data. A unified schema, which supports querying over a diverse set of information sources can provide decision-makers with valuable access to a wide set of sources. A variety of mediator architectures (e.g., [4,8,10,20]) have evolved in response to this problem. Mediators shield data consumers from the

R. Meersman et al. (Eds.): CoopIS/DOA/ODBASE 2003, LNCS 2888, pp. 370–388, 2003.

concerns of schema awareness and query planning. Clients issue queries on an integrated schema provided by the mediator. The mediator knows a wide set of information providers and their schema; it maps a client's query to distributed queries over the sources that provide data relevant to the client's query, and then integrates the returned data to construct the final answer to the client.

However, data consumers, specially in an inter-networked WAN context, need to worry about the *trustworthiness* of information sources. As described in, e.g., [11,12,13,14], data sources may have varying levels of quality. In this context, consumers would be well-advised to carefully consider the sources of the data when using data provided in response to a query. In addition, client applications may have different levels of trust. For example, when requesting a legal case-record for malpractice lawsuits against orthodontists living in Waco, Texas, a client may want a *complete* list; but when asking the locations of nearby Sri Lankan restaurants that are Glatt Kosher certified, clients might be satisfied with a partial list. Some sources may have complete data, and some sources might have incomplete or incorrect data. Current mediator architectures can process complex queries, and integrate data from multiple sources. However, they are not set up to handle the potentially quadratic scaling of possible trust relationships between clients and information sources.

In a WAN setting, mediators must also consider the possibility that the final result may have been computed using sources which may not satisfy the client's trust requirements. If the information providers and mediators charge for their services, the client might have to pay for data that she cannot use! It would therefore be desirable to determine *statically* during query plan generation and before query execution, if for a given query and a specific set of sources, the final answer would be acceptable to the client. Such a static typing scheme can be used by a mediator to prune query plans that, if executed, would result in answers to the query that do not satisfy the client's trust requirements.

In this paper, we first describe a mediator architecture that deals with the quadratic-scaling of trust relationships between clients and information sources (Sec. 2). In particular, we detail a generic model for *trust-type inference* in the style of programming languages type inference, which allows the static (or *partially static*) computation of the trust level of query results (Sec 3). Based on this general model, we provide an example of a static trust-typing algorithms in the context of a mediator architecture for relational databases as information sources (Sec. 4). After a discussion of related work (Sec. 5), we conclude the paper and outline our ongoing and future work (Sec. 6).

2 Trust Mediation

In our previous work [19], we described an architecture for trust mediation, which we briefly recapitulate here for completeness.

2.1 Mediated Query Systems

We build our trust mediation framework by extending the infrastructure typical of a mediated query system (MQS) (see, e.g., [4,10] for an overview). Multiple heterogeneous, distributed sources supply information to multiple clients. A mediator or collection of mediators connects clients to sources by integrating information from sources to satisfy client queries. Figure 1 shows the query/response interaction among components in a typical MQS. Dotted arrows indicate queries. Solid arrows indicate responses.

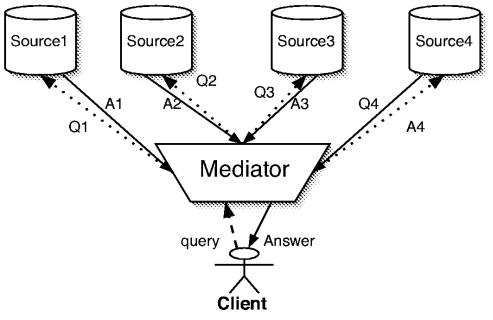


Fig. 1. Typical mediated query system. Note that the information sources used by the mediator must be trusted and acceptable to client.

With many clients and numerous information sources, mediators are in the untenable position of tracking a quadratically growing number of trust relationships in addition to their normal data integration tasks. Thus, in Fig. 1, clients must trust sources that are used to process queries. Clients must trust mediators to do correct data integration and query processing. Sources must trust clients and mediators to use the information provided in a proper way. It may be unnecessary to specify and account for every trust relationship in order to achieve the desired characteristics of a trustworthy distributed information system. For example, the trustworthiness of mediators can be assumed when the mediator exists within the same administrative and security domain as clients accessing the system. For simplicity, we will assume that the trustworthiness of mediators and access control over the information in the sources is either irrelevant or handled outside our trust mediation framework, e.g., using existing secure mediation techniques such as those proposed in [1,2,3,5].

2.2 Conceptual Architecture

Figure 2 shows a high level view of our approach. Trust authorities evaluate information sources and assign trust ratings based on precise, agreed-upon trust definitions. A trust authority may be an actual external entity such as the Better Business Bureau or a conceptual component consisting of a network of clients

willing to share their expertise and experience interacting with a source in order to establish a trust rating for a source (see, e.g., [7,17,18]). Whatever the implementation, ratings assigned by trust authorities are used by the mediator while processing client queries. In [19], we use a *trust broker* to warehouse trust meta-data provided by trust authorities; for simplicity, we omit that here.

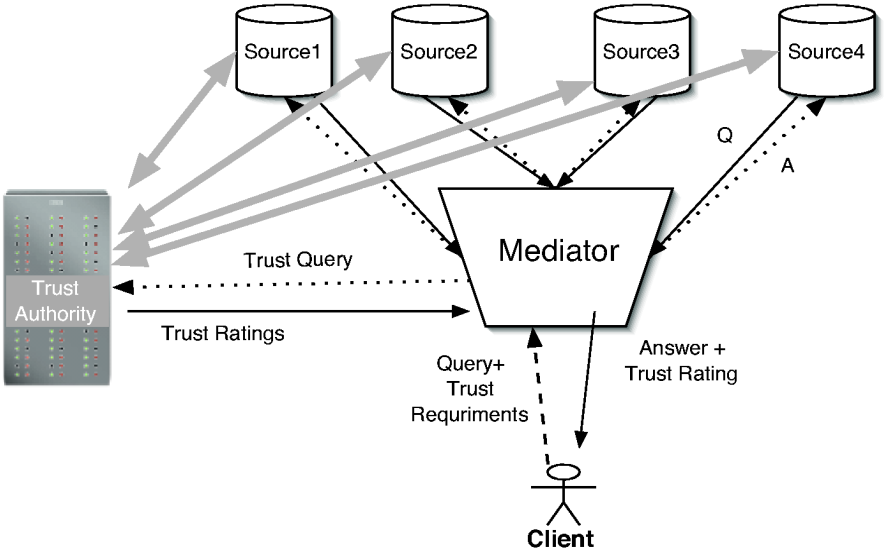


Fig. 2. Conceptual architecture for trust mediation. Gray arrows to Trust Authority represent assigning of trust meta-data. Dashed lines are query, and solid lines matching answers. Some labels are omitted for clarity.

The operation of our architecture begins when clients submit queries to a mediator. In particular, clients may attach *trust requirements* to the submitted queries. The mediator determines multiple query plans for the client query based on the global (mediated) schema. It is important to note here that multiple query plans exist because data may be duplicated in several sources. Recall that the trustworthiness of these sources may be different and thus the result of query plan execution may not be identical as is typical in a mediated query system. This is accounted for by the client through the specification of trust requirements. To select a query plan for execution, the mediator processes the trust ratings stored in the trust broker for the sources specified in each query plan. This processing step yields trust ratings for the integrated data that would be returned to the client for each executed query. The algorithmic framework for trust processing we present for performing this static analysis of query plans to determine trustworthiness of the integrated result is detailed in Sections 3 and 4. Operation of the framework continues as the mediator executes a query plan whose trust rating satisfies the client trust requirements. The retrieved data is

then sent to the client. The mediator notifies the client if no query plan satisfies the client trust requirements.

The fundamental contribution of this paper is a static trust-typing model that can be used by the mediator for more efficient and effective query planning. In the next section, we formalize the model upon which the architecture described in this section is built.

3 General Static Trust-Typing Model

In this section, we begin first with a general, abstract data model (Sec. 3.1). We then present a general model (Sec. 3.2) of static trust-typing within this general data model. We then discuss a general notion of a trust-type inference algorithm (Sec. 3.3). Finally, we present desirable properties of such a trust-type inference algorithm (Sec. 3.4).

3.1 Preliminaries

We start with a very general data model, to avoid commitment to a specific data model such as the relational or an object-oriented model. Consider an *information domain*, consisting of a finite set of primitive types

$$T_p = \{t_1, t_2, \ldots, t_{|T_p|}\}.$$

These would evidently include strings, integers, floats and so on. We also assume a finite set of *type constructors*

$$C = \{c_1, c_2, \ldots, c_{|C|}\}.$$

Type constructors (e.g., products, sums, records, etc) have different arities and are applied to primitive types to create a potentially infinite set of possible derived types (which include primitive types)

$$T_d = \{t_1, t_2, \ldots\}.$$

Within this framework, we also allow *data sets* Δ_i, which *populate* these types, shown thus

$$\Delta_i : \tau, \ \textit{where } \tau \in T_d.$$

Data sets are manipulated by a finite set of operators

$$O = \{o_1, o_2, \ldots, o_{|O|}\}$$

where each operator takes one or more inputs of specified types and then delivers an output of a specified type, thus:

$$o_i :: t_{i_1} \times t_{i_2} \times \ldots \times t_{i_m} \rightarrow t_{output}$$

We assume that operators have a well-defined computational semantics. That is, given input data items that instantiate the input types of an operator, there is

a well-defined function to calculate the output, which will then instantiate the output type. Operators can be used to build complex (nested), well-formed *query expressions*, as allowed by the type signatures specified for the operators. Also, operators can have rewrite rules associated with them that specify typical properties such commutativity, associativity etc., which may allow for optimization of queries by a query processor.

A *schema* S is a collection of types

$$S = \{\tau_1, \tau_2, \ldots, \tau_{|S|}\},\ where\ \tau_i \in T_d$$

An *instance* of a schema S is just a collection of sets $\{D_1, D_2, \ldots, D_{|S|}\}$ of data items that populate each type in the schema. Operationally, we assume that an information system (or database system) that manages the data items and to which clients submit well-formed queries. The information system then evaluates these queries and returns answers. The extension of this notion to a client who uses multiple information systems to process a single query (where the input datasets to the queries may originate from different sources) is self-evident.

So we capture here a broad set of data models, including relational, and object-oriented and even semi-structured models. In the following section, we develop a static typing model for trust.

3.2 Trust

Consider a collection of information sources, each instantiating a schema. The schemas may overlap, i.e., different source schemas may share types. For example, there may be two different databases which offer data about the set of antibiotics suitable for treating Anthrax. The client may have different levels of *trust* in each database, and may issue different queries, some of which are critical (and require high trust) and some that are not.

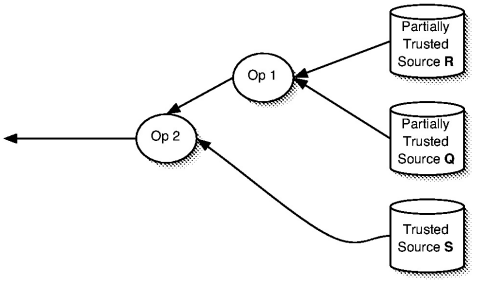


Fig. 3. Example of query answered using multiple sources, which are accorded differing levels of trust

Trust is a complex concept with many possible different meanings (see, e.g., [6,15]). Trust may refer to *aggregate* properties of an entire dataset, e.g, a user

might believe that the set of anti-anthrax drugs identified by a database may be incomplete. Trust may be *content-based*, referring to detailed beliefs about specific members of a data set. For example, a client may believe that a dataset has incorrect information about anti-Anthrax drugs whose manufacturer is in Europe. Trust may also be at the level of individual data instances. How does the client assign these beliefs? The rational way to do so would be based on *experience*. In practice, for example, we assign trust to information sources based on how useful, accurate and/or valuable the information they provide is in the real world.

Evidently trust varies with the trustee (in this case, the information source). In a distributed/mediated environment we might have a situation as shown informally in Fig. 3, with a query of the form $op2(op1(R, Q), S)$, where different sources are needed to process the query. In this context, the sources for R and Q are only partially trusted, but the source for S is fully trusted. Evidently, the client cannot fully trust the final result; but what should the trust be? One can certainly use the result data and see how it works out; but clearly it is preferable to have good *a-priori knowledge* of the trustworthiness of data sets or information sources rather than having actually determine this post-facto, after using the data to make business-critical decisions (consider the implications of prescribing drugs using information of unknown trustworthiness). We can take this one step further: rather than evaluating the final answer set *after* the query has been processed, we would like to *statically* determine the trust level of the final answer before executing the query against the sources (and thus perhaps avoid paying for the processing of sub-queries that might result in untrustworthy answers).

Thus, there is strong motivation for a semantically well-founded *a-priori* calculation of the trust level that can be associated with answer datasets returned by incompletely trusted information sources. This can be considered a meta-data calculation based on trust meta-data that is available a-priori about those sources. In fact, we wish to calculate the trust meta-data *statically*, i.e., based on the structure of the query plan and available trust-meta on the data sources.

Consider the following setting of information sources $DB_1, \ldots, DB_n$, with schemas $S_1, \ldots, S_n$. Without loss of generality, assume that each database schema is a singleton type. We now assume a simple trust meta-data model, where we assume an aggregate trust-type associated with each source. Each aggregate type is drawn from a (potentially infinite) trust-type poset B, with ordering relation R_B :

$$B = \{b_1, \ldots, b_{|B|}\}, \quad R_B = \{(x, y) \mid x, y, \in B\}$$

We show the *assignment* of a trust-type b to a data set D_i of a particular type T thus:

$$TT(D_i : T) = b$$

In the ensuing discussion, the actual datatype is usually not of concern, so we just denote the trust-type, thus:

$$D_i :: b$$

The ordering relationship R_B is used to model the fact that some sources can be more trusted than others. In general, this relationship is not a total order: not all trust relationships are comparable, and different trust ratings may be preferred under different circumstances[1]. This trust-type assignment may be performed by the clients themselves, or (for better scalability) by a credible trust authority on behalf of the clients, e.g., the Better Business Bureau, or a government agency. We note that there may be multiple information source with the same schema, which have different trust types assigned to them. So, there may be several different ways of evaluating a given query, by sourcing information from one of several alternative sources.

Now consider that the clients need to evaluate results returned for queries against a mediator; how can they develop trust-type assignments for the results of the queries? Clearly, clients need this information. It would be impractical for third parties to develop trust ratings for all possible queries and all possible ways of evaluating respective query results. It is also possible that a client needs information at a trust level that is so high that it is impossible to get this information, given the trust assignment of input sources. It is also possible that some evaluations of a given query (using some specific information sources or combination of sources) will provide answers at the required trust levels, and some would not.

3.3 Trust-Type Inference

We propose an approach to processing *aggregate* trust meta-data or trust-types in the setting described above, which has a strong analogy to type systems in programming languages (see, e.g., [9]). Consider a query algebra $\mathcal{A}$ with operators

$$O = \{o_1, \ldots, o_{|O|}\}$$

as defined above. Consider a specific operator

$$o_i :: t_{i_1} \times t_{i_2} \times \ldots \times t_{i_m} \rightarrow t_{output}$$

A *trust typing system* associates with each such operator a *trust-type inference rule*

$$\frac{TT(D_{i_1} :: t_{i_1}) = b_{i_1} \;\&\; TT(D_{i_2} :: t_{i_2}) = b_{i_2} \;\&\; \ldots \;\&\; TT(D_{i_m} :: t_{i_m}) = b_{i_m}}{TT(o_i(D_{i_1}, \ldots, D_{i_m}) :: t_{output}) = f_{o_i}(b_{i_1}, \ldots, b_{i_m})}$$

where $b_{i_1} \ldots b_{i_m}, b_{output}$ are trust-type assignments. Given (1) an *a-priori* trust-type assignment for the input data sets to an operation o_i and (2) the inference rule, the associated trust computing function f_{o_i} allows the *static* (before query execution) derivation of a trust-type assignment for the result of evaluating the operation over those data sets. In general, the function f_{o_i} could return a set of

[1] We refer the reader to the earlier example of lawsuits and restaurants in Sec. 1, where different ratings are preferred.

trust-types rather than a single trust-type: in some situations (examples follow) more than one trust-type might be inferred.

Since query plans are nothing more than a nesting of query operators, this approach clearly can used for defining trust-type judgments for query plans as they are generated by a typical query processor in a mediator.

Definition 1. *A* query plan Q_p *in a query algebra is a well-formed expression using the operators defined in that algebra. The* input data set *to* Q_p *is the set of data sets* $\{D_1, \ldots, D_n\}$ *that form the inputs to the query plan. The* input trust-type assignments *for* Q_p, $IT(Q_p)$ *is the set of trust type assignments* $\{D_1 :: T_1, \ldots, D_n :: T_n\}$ *provided by some trust authority.*

Given a query plan generated by a mediator and a set of input trust-type assignments, one can use a trust typing system to produce, statically, a final trust-type rating for the result of the query. This provides several advantages:

1. Once the trust assignments of input data sets to a query are known, a mediator can compute the trust assignment of the result without any additional information.
2. Once trust ratings are known, the mediator does not have to actually execute a query to determine whether the query result will satisfy the trust requirements specified by the client.
3. Given a specific query and the required input data from several different alternate information sources, a mediator may be able to reformulate a query (plan) into a form that can be evaluated in order to satisfy the trust requirements.

However, in order to use a trust-typing system in this fashion, it must satisfy some important properties. We discuss these in the following section.

3.4 Soundness and Completeness

We define here two important properties of any static trust-type inferencing mechanism: soundness and completeness[2].

A Naive View of Soundness. By *soundness*, we mean that the judgments made by the trust typing system are accurate. This accuracy is established by comparison with a reliable "oracle" that can provide trust ratings; in our case, these ratings are provided by the trust authority. The trust authority provides ratings for the data available from the information sources. With the ratings on the data that constitute the inputs to a query plan, the trust typing system can now derive a static trust typing judgment for the final result of the query, if executed. The accuracy of this judgment can be evaluated empirically. First,

[2] In programming language type systems, completeness in any useful sense is typically an impossible goal, given a Turing-complete programming language. Since query languages are usually not as expressive, completeness is relevant here.

based on the determined query plan, the query is executed by the mediator, producing some final query result. The final result can then be subsequently considered by the trust authority, and given a rating. In this context, the trust typing system is considered *sound*, if the trusting rating judgment produced by the typing system for a given query plan and input data with a given rating *would be exactly the same rating* as what the trust authority would produce, were it given the query result. In general, we prove soundness of a typing system by considering each input rule in turn, and argue that its conclusions, given ratings on the input data, would be precisely the same as that of the trust authority.

However, in practice, this is not feasible. Imagine two information sources, each of which has partial data: one about the drugs to treat seasonal allergies, and the other drugs that are acceptable for patients with hypertension. The *intersection* of these two data sets would be drugs for seasonal allergies that can be used by hypertensive patients. Note here that even if two input data sets are rated partial, they may each have just those data items that would yield an *exactly correct* final result for their intersection; so, given the actual final result, a trust authority might in fact be able to rate it as exactly correct. However, a static typing system, without *a priori* access to this data, cannot make such a judgment. Given a rating, it has to take a pessimistic view on what the actual result data might be; so it must conclude that the result could be either exactly correct, or it could be missing some data. Thus, the notion of soundness has to be modified, to allow the static judgement to be pessimistically approximate.

A different view: Correct and Precise. Therefore, our notion of soundness has two aspects. First, we want our trust typing algorithm to be *correct*: given a final static typing judgement on a query (or query plan, to be more precise), there must be a particular configuration of data items on the input sources, consistent with the input type assignments given by the trust authority, that would lead to a final result that would be rated exactly the same way by *both* the typing judgement and the trust authority. Correctness guarantees that a derived typing judgment is never wrong. But this is not sufficient. We also need to be sure that the typing judgment does not miss any possibilities.

In general, several different configurations of input/output data may be possible for each operation, we do not want the type system to miss anything: we also want it to *precise*. That is, if it is possible that a particular configuration of inputs could lead to a result type rating by the trust authority, the static typing judgment should include that type. It is important here to note that *we assume that the trust authority behaves in some semantically consistent manner: viz.,* the authority does not capriciously confer trust ratings onto data sets, but does so in some well-defined manner. Only then can we hope to approximate the trust authority's behavior using a static typing model. We now present definitions for correctness, precision, and completeness.

Definition 2. *Consider a query plan Q_p, and input trust-type assignment $IT(Q_p)$, drawn from a trust-type poset (B, R_B), provided by a trust authority. Assume, without loss of generality, that Q_p is evaluated on some given set of*

inputs conforming to the input trust-type assignment, $IT(Q_p)$, and the result is evaluated by the trust authority and given the (empirical) trust-type $T_E(Q_p)$. Also assume that the trust-type algorithm derives the static trust type $T_S(Q_p)$.

In this setting, a trust typing algorithm is correct*, if for each type $\mathcal{T} \in IT(Q_p)$, it is possible to construct an artificial input data set D_A for the query plan Q_p so that*

1. *The trust authority's ratings for the input data set D_A would be precisely $IT(Q_p)$, and*
2. *$T_E(Q_p) = \mathcal{T}$.*

Definition 3. *In the same setting, a trust typing algorithm is* precise*, if it is impossible to construct an artificial input data set D_A for the query plan Q_p so that*

1. *The trust authority's ratings for the input data set D_A would be precisely $IT(Q_p)$, and*
2. *the query result is given the trust rating $\mathcal{T}$ by the trust authority, where $\mathcal{T} \notin T_E(Q_p)$.*

Completeness. By this, we mean that the typing system is able to provide for judgments for every possible query expression. This is a measure of the expressiveness of the type system; can it provide ratings for any kind of query plan? Essentially the desired property is the following: given any given query plan, and ratings on the input data, a trust rating can be given to the final result. We show this by induction: for each operation, and each possible input trust-type rating, we show that a rating on the result can be produced by the trust typing system. Completeness then follows by structural induction.

Definition 4. *A trust-typing system for a query algebra $\mathcal{A}$ is* complete*, if for every operator o_i in the algebra, and for every possible input type assignment to the inputs to the operation, the trust-typing system includes an inference rule that infers a trust-type judgment for the result of the operation.*

4 An Example Trust-Type System

Consider a trust-type system such that every source rated using this system will match exactly one of five trust-types, C, I, E, O, W. The intuition behind this type system is based on a notion of *data coverage*. An information source is rated C if is complete, *i.e.,* if it has exactly the "correct" data set[3]. I refers to "incomplete", some data items are missing from what the source provides; E refers to "excessive", if all correct data items are present at the source, but some erroneous items are also present; O refers to "overlapping", meaning some

[3] We provide precise semantics for correctness later; first, we just provide an intuitive notion of this.

correct tuples are missing, and some erroneous ones are present; and finally W refers to the case where the source provides only wrong data items. As for ordering, we take C to be the most trusted, and W to be the least trusted; the other three trust-type are in between, and are mutually incommensurate, since they are each preferable under different circumstances.

Definition 5. *The* basic coverage trust-typing system *(CTS) is defined by* $B^b = \{C, I, E, O, W\}$*, and the poset on* B^b *is defined as* $R_B^b = \{$ *(W,E), (W,O), (W,I), (E,C), (I,C), (O,C)* $\}$*;* (X, Y) *means that* X *is a lower trust level than* Y*.*

As we will see below, when using our inference algorithms, it is possible that the result of a query plan may be given multiple ratings. For example, the result of the intersection of two data sets rated I might be rated I, C: both results are possible.

We can now provide a semantics for the trust-typing system, that is, we can precisely define the conditions under which we expect a trust authority TA to provide each of the above ratings to a source S. To do this, we assume that a trust authority has its own (presumably infallible) view of the set of data items that a source *should* have; this view of TA on S is denoted S_{TA}. The trust authority then compares its view of the source with the dataset actually provided by the source S and assigns ratings as defined below.

Definition 6. *A semantically consistent trust authority* TA *assigns trust-types in the basic coverage trust-typing system (CTS) for a source* S *as follows*

$$TT(S) = \begin{array}{ll} C \text{ (complete)} & , \text{ if } S = S_{TA} \\ I \text{ (incomplete)} & , \text{ if } S \subset S_{TA} \wedge S_{TA} \neq \emptyset \\ E \text{ (excessive)} & , \text{ if } S \supset S_{TA} \wedge S \neq \emptyset \\ O \text{ (overlapping)} & , \text{ if } (S \cap S_{TA} \neq \emptyset) \wedge (S \not\subset S_{TA}) \wedge (S_{TA} \not\subset S) \\ W \text{ (wrong)} & , \text{ if } (S \cap S_{TA} = \emptyset) \wedge (S \neq \emptyset) \wedge (S_{TA} \neq \emptyset) \end{array}$$

For example, the trust authority TA rates S complete if S contains exactly all data items TA expects, i.e., $S = S_{TA}$. TA rates S wrong if there is no data item in S that is also in the trust authority's view S_{TA} and both sets of data items S_{TA} and S are non-empty. The definitions are a little tricky for the case where either S_{TA} or S_A might be empty: e.g., if $S_{TA} = \phi$, a non-empty source should be rated *excessive*, not *wrong*; and an empty source should be rated *complete*, not *wrong*. The above definition does work correctly, and in fact admits a correct, precise, and complete inference algorithm, as we shall see next.

4.1 Algorithm for CTS Trust-Type Inference

There are five possible trust-types in the basic coverage type system. Since the focus in this section is on relational databases as information sources a mediator operates on, we use the relational algebra to specify queries and assign trust-types to respective (intermediate) query results. We use the following six operators of the relational algebra, four binary operators and two unary operators: set union ($\cup$), set intersection ($\cap$), set difference ($-$), cross product ($\times$), selection

(σ), and projection (π). There is a total of 80 input trust-type combinations: for each commutative binary operator ($\cap, \cup, \times$), there are 15 possible combinations of input trust-types $((I,I),(I,C),(I,O),(I,E),(I,O),(C,O),\ldots)$. For the non-commutative set difference, there are 25 possible input trust-type combinations, and for the two unary operators there is a total of 10 possible input trust-types. These 80 input trust-type configurations naturally lead to 80 trust inference rules. The basic 80-rule inference algorithm handles the basic type system, as presented in Def. 5, in practice, however, multiple type ratings, as envisioned in Def. 6, can occur when inferring types of query plans. Each of these types represent a possible trust rating of an intermediate result. If such a result is used in another operation, each possible trust-type rating must be considered in turn, with an appropriate inference rule, and the result trust-type rating will be union of all the resulting trust types. In case of a binary operation, each pair of trust-type ratings from the two inputs should be considered.

Our goal is to prove the following theorem for the basic coverage trust-typing algorithm:

Theorem 1. *The coverage trust-type system is correct, precise, and complete for the above six operators of the relational algebra and for trust authorities that assign semantically well-founded trust ratings.*

Proof. *Completeness* is demonstrated by listing all the 80 possible rules, which provide inference rules for all possible combinations of input types for all possible rules. We omit it here for brevity.

Correctness and *precision* can be demonstrated by considering each of the entire set of inference rules, and arguing correctness and precision. For brevity, we just provide proofs of correctness and precision for a few rules, for some of the combinations that illustrate the techniques.

Figure 4 shows some of the inference rules for projection operator (π), along with Venn diagrams that establish for each case a configuration that shows that the rule is correct, i.e, that there is a specific configuration of the input data set conforming to the input trust-type that leads to the output trust-type. For example, in row 3 on the rightmost column, the Venn diagram provides the correctness witness (existence proof) for the rule

$$\frac{S :: O}{\pi(S) :: E}$$

The actual dataset is shown with the dashed rectangle, and the dataset from the source is the solid ellipse. The actual, expected projection result is shown in the solid semi-ellipse, whereas projection applied to the data provided by the source yields the solid ellipse. Thus in this case, the output can be rated E, for excessive. This establishes that with an input conforming to an O (overlapping) rating from a trust authority, it is possible to have an output that would be rated E by the same authority. In this way, we can provide justification for each of the cases in the right column. This establishes *correctness* for the projection rule in the case where the input is rated O. Similarly, arguing by case, and providing

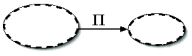
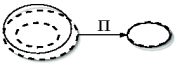
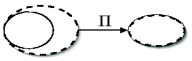
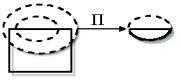
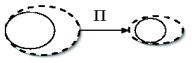
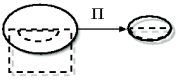
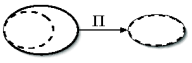
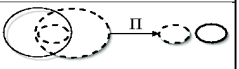
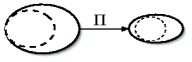
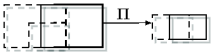
Inference Rule	Witness (for correctness)	Inference Rule	Witness (for correctness)
$\frac{S::C}{\pi(S)::C}$		$\frac{O::C}{\pi(S)::C}$	
$\frac{S::I}{\pi(S)::C}$		$\frac{S::O}{\pi(S)::I}$	
$\frac{S::I}{\pi(S)::I}$		$\frac{S::O}{\pi(S)::E}$	
$\frac{S::E}{\pi(S)::C}$		$\frac{S::O}{\pi(S):W}$	
$\frac{S::E}{\pi(S)::E}$		$\frac{S::O}{\pi(S)::O}$	

Fig. 4. Inference rules for projection operator; closed ellipses or rectangles represent sets. Sets with solid boundaries represent the actual dataset provided by the source S, and the dashed-line sets indicate the actual (or correct dataset, as assumed by the trust authority).

Venn diagrams, we can argue correctness for the entire rule-set for projection. Later we show some similar correctness arguments for the non-commutative binary operator, set-difference.

We also need to show that the static trust-typing inference rules for projection are *precise*, *viz.,* that a semantically consistent trust authority could never produce a rating other than what is provided by static inference. In the case of projection, we can see that an input rated *overlapping* leads to every possible rating for the output; so in this case, the inference rule is self-evidently precise. We now consider the other case

$$\frac{S :: I}{\pi(S) :: C, I}$$

The rule states that with an *incomplete* (I) input, projection yields an output that is either complete or incomplete. We now argue that no other output rating from a semantically consistent trust authority is possible, for *any* input that is rated I. If a trust authority provides such a rating, it does not conform to the semantics. In contradiction, suppose the result is rated W. This means that there is some result tuple t in it that should not be, as per the trust authority's view of the output. By the semantics of projection, this is only possible if there was at least one tuple τ in the input that gave rise to t, that the trust authority would not expect to see in the input; however, the input is rated I, so such a tuple

cannot exist if the authority is semantically consistent. By a similar argument, we can show that the output cannot be rated E or O.

We now present a similar discussion for the asymmetric, binary set-difference operator. Figure 5 shows 4 of the 25 different rules for set difference for the case where one of the inputs is rated C and the other I. Note the asymmetry in the inferred output ratings (non-commutativity of set difference).

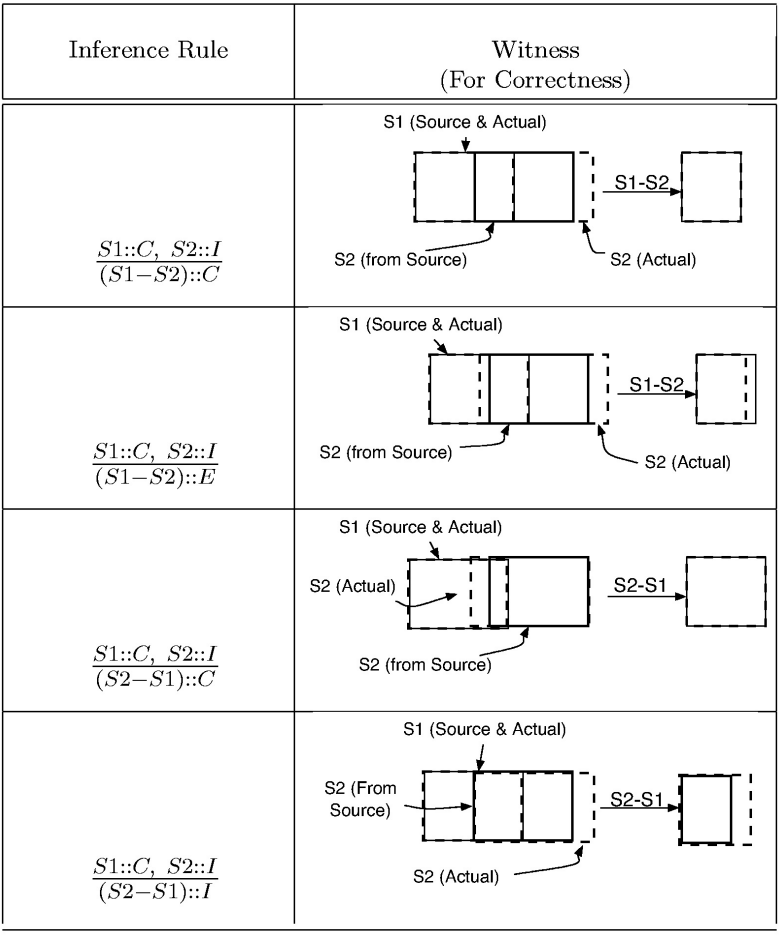


Fig. 5. Four out of twenty-five inference rules for the set-difference operator.

We can also argue precision. With the ratings { *S1::C, S2::I* }, no other ratings are possible for $S1-S2$ besides C or E. In contradiction, suppose $S1-S2$ were rated I by a trust authority. This implies that some tuple t is missing in the output result of $S1 - S2$ that the trust authority does not expect to be there. This implies that either this tuple is missing in $S1$ (not possible, since $S1$ is

rated complete) or this is an incorrect extra tuple in $S2$ (also not possible, since $S2$ is rated incomplete). Thus, it is impossible find inputs *S1::C* and *S2::I* so that a semantically consistent trust authority would rate $S1 - S2$ as excessive. Similarly, we can argue that the ratings W and O are impossible. The precision of the entire rule-set for the set-difference operator is argued in the same way.

4.2 Query Processing and Trust-Type Inference

We conclude this section with an outline of how the above algorithmic framework for trust-type inference can be integrated into a mediated query system. For this, we assume a standard query processor as it can be found in proposed mediator architectures (e.g., [8,20]).

Initially, a client specifies a query Q against a mediated schema and also specifies a trust requirement for the query result, denoted $TR(Q)$. A trust requirement can be a singleton or a set of trust types, drawn from a basic coverage trust typing system (see Def. 5). The mediator processes the query in the standard way. That is, it generates a set of query plans $qp_1(Q), qp_2(Q), \ldots, qp_k(Q)$ for the query Q and chooses the most cost-effective plan for execution against the set of information sources referred to in the query plan.

Our trust-type inference algorithm can be plugged into the query plan generation component of the mediator's query processor as follows.

1. A query plan $qp(Q)$ for a query Q is generated bottom-up; leaf nodes of the plan (if considered as a hierarchical, bottom-up structure), are assigned input trust ratings. Leaf nodes refer to information sources and ratings are assigned to these sources by a trust authority.
2. After trust ratings have been assigned to leaf-nodes, the query processor builds the query plan as a nested expression tree, using the operators available in the data model used for the mediated schema. The root node designates the query result. For each operator o applied to one or two input data sets (which can be either the information sources or intermediate query results), the trust ratings for the input data sets are known. Assume two input data sets S_1, S_2 with trust ratings $\{t_1, t_2\}$ and $\{t_1\}$, respectively (note that intermediate results can have more than one trust rating, see Def. 6). The query processor now does a lookup of the inference rule for both possibilities of input ratings, $S_1 :: \{t_1\}, S_2 :: \{t_1\}$ and $S_1 :: \{t_2\}, S_2 :: \{t_1\}$ and thus obtains the output result type(s), which are assigned to the (intermediate) result node obtained by applying o to S_1, S_2. Eventually, by building the complete operator tree for Q, a trust rating is assigned to the final query result. If the final trust rating does not satisfy the trust requirements specified by the client, the query plan is pruned from the set of potential query plans. Finally, the most cost effective query plan that satisfies the client's trust requirements executed after the above static trust-type inferencing.

There are some important aspects to note regarding the usage of trust inference rules during query plan construction. First, the lookup of result trust-types

in the presence of rated input data sets and an operator can be done very efficiently, and intermediate result nodes are simply "annotated" by trust ratings. Second, in general, the trust rating obtained for an intermediate query result can be "better" (with respect the underlying trust-type lattice) than the trust ratings of the input trust-types. That is, it is in general not possible to prune a plan before it is completely constructed. However, there are several cases where such a pruning can be done, i.e., when it is known that based on the rating of an intermediate result, the final query result will never satisfy the client's trust requirements. We are currently investigating this aspect in the context of mediated relational databases. Finally, it can happen that there is no query plan that satisfies the client's trust requirements. In this case, the mediator needs to provide the client with respective feedback, e.g., by stating the trust levels that could be assigned to the query result. Another interesting aspect is that the query plan generation is not purely cost-driven anymore. That is, one can construct scenarios where a query plan does not satisfy the client's trust requirements but a more expensive plan does. This is typically the case if the information sources referred to in a query plan contain non-disjoint data.

5 Related Work

The concept of trust in distributed, mediated information sources has mainly been investigated in the context *secure mediation* [1,2,3,5]. These works address the problem of managing client credentials to ensure that a mediator does not violate information source security policies when integrating data to satisfy client queries.

Our proposed static type inferencing framework provides an orthogonal aspect to these works and is most closely related to the management and handling of data quality aspects in data integration and source mediation [11,12,13,14,16]. Our model can be considered as an abstract formal framework, which can incorporate a variety of data quality aspects as one or more aggregated trust-types. However, while most works on managing data quality aspects in mediation deal with the run-time aspects of data quality, our work leverages existing mediation architectures and query processing strategies in particular by performing static type-checking of potential query results. Thus, our framework in general provides the chance of avoiding expensive query execution in case no query result will satisfy the (trust) requirements specified by clients. Finally, compared with work in data quality management, our proposed framework provides a formal framework for static typing and also an effective algorithmic infrastructure to deal with ratings of information sources in the context of recommender systems [7,17,18]. Currently, our results are limited to the basic completeness type system (CTS). In future work, we hope to extend it to other notions of data quality and trust.

6 Conclusions and Future Work

In this paper, we address the problem of obtaining trustworthy information for clients in mediated architectures deployed in a WAN setting. In such settings, not all information sources may be trustworthy; mediators must be careful to use query plans that only take data from sources that clients would find acceptable. We describe a general model of *static trust-typing* to infer the trustworthiness of the results of query plans, even before executing these plans. Such a static trust-typing approach can be used by mediators to prune and thus avoid query plans that would compute query results that would be unacceptable to clients because they do not satisfy the trust requirements specified by clients. In the general model, we discuss desirable properties of static trust-typing systems, *viz.*, correctness, precision, and completeness. We then give an example of a trust-typing system in the context of relational databases that does show these properties.

In our ongoing work, we are currently developing a fine-grained trust-typing system in which trust authorities can assign trust-types to different components of a source (object sets, object attributes etc). In combination with such a framework, we are also studying how such fine-grained trust assignments can be used in query plan generation to annotate individual data items in query results with trust levels, eventually to provide clients with more detailed information regarding the trustworthiness of query results.

References

1. J. Biskup, U. Flegel, and Y. Karabulut: Secure mediation: Requirements and Design. In IFIP WG11.3 13th International Working Conference on Database Security (DBSec 98), Kluwer, 127–140, 1999.
2. J. Biskup, Y. Karabulut: A Hybrid PKI Model with an Application for Secure Mediation. In Proceedings of the 16th Annual IFIP WG11.3 Working Conference on Data and Application Security, Kluwer, 2002.
3. K. S. Candan, S. Jajodia, V. S. Subrahmanian: Secure Mediated Databases. In 12th International Conference on Data Engineering (ICDE 96), IEEE Computer Society 1996, 28–37.
4. R. Domenig, K. R. Dittrich: An Overview and Classification of Mediated Query Systems. SIGMOD Record 28(3): 63–72, 1999.
5. S. Dawson, S. Qian, P. Samarati: Secure Interoperation of Heterogeneous Systems: A Mediator-based Approach. In the 14th IFIP International Conference on Information Security, Kluwer, 1998.
6. R. Fagin, J. Y. Halpern: I'm OK if You're OK: On the Notion of Trusting Communication. In Proceedings of the Symposium on Logic in Computer Science (LICS '87), IEEE Computer Society, 280–292, 1987.
7. W. Hill, L. Stead, M. Rosenstein, and G. Furnas. Recommending and Evaluating Choices in a Virtual Community of Use. In Proceedings of the ACM Conference on Human Factors in Computing Systems, CHI'95, ACM/Addison-Wesley, 194–201.
8. V. Josifovski, and T. Risch: Query Decomposition for a Distributed Object-Oriented Mediator System. *Distributed and Parallel Databases* 11(3): 307–336 (2002)

9. J. C. Mitchell, Concepts in Programming Languages. *Cambridge University Press*, 2003.
10. I. Manolescu, L. Bouganim, F. Fabret, E. Simon: Efficient Querying of Distributed Resources in Mediator Systems. In *Confederated International Conferences DOA, CoopIS and ODBASE 2002*, LNCS 2519, Springer, 468–485, 2002.
11. G. A. Mihaila, L. Raschid, M.-E. Vidal: Using Quality of Data Metadata for Source Selection and Ranking. In Proceedings of the Third International Workshop on the Web and Databases, WebDB 2000 (Informal Proceedings), 93–98, 2000.
12. M. Mecella, M. Scannapieco, A. Virgillito, R. Baldoni, T. Catarci, C. Batini: Managing Data Quality in Cooperative Information Systems. In *Confederated International Conferences DOA, CoopIS and ODBASE 2002*, LNCS 2519, Springer, 486–502, 2002.
13. F. Naumann: Quality-Driven Query Answering for Integrated Information Systems. Lecture Notes in Computer Science 2261, Springer, 2002.
14. F. Naumann, U. Leser, J. C. Freytag: Quality-driven Integration of Heterogeneous Information Systems. In Proceedings of the 25th International Conference on Very Large Data Bases, 447–458, 1999.
15. NIST (National Institute of Standards and Technology). Glossary of Computer Security Terminology. NIST Technical Report, NISTIR 4659, September 1991.
16. L. Pipino, Y. W. Lee, R. Y. Wang: Data Quality Assessment. In Communications of the ACM 45(4): 211–218, 2002.
17. D. Pemberton, T. Rodden and R. Procter: GroupMark: A WWW Recommender System Combining Collaborative and Information Filtering. In the 6th ERCIM Workshop "User Interfaces for all", ui4all.ics.forth.gr/UI4ALL-2000/proceedings.html, 2000.
18. Recommender Systems. Special Section in Communications of the ACM, Vol. 40, No. 3; March 1997
19. B. Toone, M. Gertz, P. Devanbu: Trust Mediation for Distributed Information Systems. In Security and Privacy in the Age of Uncertainty, IFIP TC11 18th International Conference on Information Security (SEC2003), Kluwer, 1–12, 2003.
20. V. Zadorozhny, L. Raschid, M. Vidal, T. Urhan, L. Bright: Efficient Evaluation of Queries in a Mediator for WebSources. In ACM SIGMOD International Conference on Management of Data, ACM, 85-96, 2002.