A Simple Framework for Contracts in Federated Database Systems

William M. Farmer[*] Mark E. Nadel[†]

23 May 1995

Abstract

A federated database system is a collection of organizations, called a federation, and a collection of databases belonging to the organizations. Members of the federation can enter into contracts that specify how data is shared between the databases of the system. An important issue is how the contracts interact with one another. This paper presents a simple framework for writing contracts in federated database systems that limits contract interaction, and thereby, makes it possible to detect contention between contracts.

Keywords: federated database systems, information security, contract interactions, information sharing

ACM Categories: F.4, H.2, K.6

1 Foreword

The problem of formulating the basic doctrine supporting federated database systems is interesting and challenging, as is the design and implementation of such systems [2]. A large part of the difficulty is contributed by various underlying technologies. There are problems in designing single-owner distributed databases as well as in implementing consistency constraints on a

[*]Address: The MITRE Corporation, 202 Burlington Road, Bedford, MA 01730-1420. Phone: 617-271-2907. E-mail: `farmer@mitre.org`.

[†]Address: Lemma Inc., 25 Woodlawn Dr., Chestnut Hill, MA 02167. Phone: 617-244-6870. E-mail: `mnadel@math.harvard.edu`.

single database through the use of triggers, for example. As another example, the use of roles already occurs in single-owner systems, but in a federated database system there may be additional difficulties in authentication, which we do not think of as specifically a database issue. We want to find a framework in which we can say something significant about federated database systems without first having to settle these other questions. To enable this goal we will be guided by a principle of parsimony to include only what we deem most essential.

One of the central issues in the field of federated database systems is the creation of contracts to control the sharing of data between members of the federation. Our framework will provide a few simple constructs from which contracts are to be fashioned. The framework is somewhat analogous to specifying geometric figures as being constructible by ruler and compass: we only give the basic building blocks, but say nothing about a language for describing complicated constructions, nor about the range of geometric figures that can be constructed in this way.

One should expect several things from this framework. First, the constructs of the framework should be adequate for expressing a significant number of contractual objectives in an acceptable way. Second, the implementation of contracts following the framework should be possible using standard techniques. Moreover, the fact that the contract is expressed in this way should guide us directly to an implementation. Third, such contracts should be easy to understand. In particular, one should be able to easily detect conflicts between contracts. This objective should be supported by keeping contracts as "local" as possible, thereby minimizing the interaction between contracts.

One might imagine a contract as containing information concerning certain actions and, in addition, the rationale behind these actions. Our view is that this second aspect of contracts need not be handled on the same technical level as the first. The framework to follow will address the first aspect. The second aspect can be dealt with using more familiar contractual vehicles and does not need to be explicitly present in the implementation of the contract.

Although our intension is to address the problem of writing contracts for sharing data stored in databases, the reader will see that the framework we propose could be used equally well for writing contracts for sharing many other kinds of privileges.

2 The Basic Framework: Granting Rights

The heart of the federated database problem is to be able to share certain data between independent organizations without giving up full control of that data. Typically, one organization will permit another organization certain accesses to its data provided that the latter organization agrees to fulfill certain obligations. A problem that needs addressing is that obligations imposed by various contracts may conflict with one another. In the following we present a framework that eliminates potential conflicts between obligations by eliminating the need for obligations. In fact, we simply do not allow for obligations in the framework. One might say that there is only one global obligation, namely, to abide by the framework. (We also make the tacit assumption that an interorganizational access privilege can only be granted through a contract.) The desired effect will be achieved partly by the particulars of the contract and partly by the general framework itself. We make no attempt in this framework to address issues of implementation or assurance. Specifically, we will not be concerned with general mechanisms for guaranteeing that all parties must abide by the contract, though it will be possible for contracts to be written so as to impose some checks and balances toward this end.

We now present the *Basic Framework*. It's purpose in this paper is mainly pedagogical; it introduces the first half of the *Full Framework* which is presented later in §6.

Assume there is a collection of organizations O_i. For each i there is a database D_i which is completely controlled by O_i and to which O_i has complete access—which means O_i has the freedom to do anything it wants with D_i. (Later in the Full Framework, we will allow for an organization O_i to relinquish some of its access rights to D_i.) For each i there is a database d_i which is accessible by O_i only to the extent allowed by contracts and whose control is generally shared by a collection of organizations.

Our version of contracts will be asymmetric. In any contract there will be a single importer O_i and a set $\mathcal{O} = \{O_{j_1}, \ldots, O_{j_n}\}$ of exporters. Under such a contract O_i will be allowed certain accesses to $D_{j_1}, \ldots, D_{j_n}, d_i$, while O_{j_k} will be given certain accesses to d_i. More specifically, we will think of d_i as being partitioned into various parts $d_{i,c}$, where c ranges over contracts in which O_i is the importer. This partition, and even d_i itself, is only a conceptual device; we place no assumptions on how it will implemented. (In fact, it might be the case that d_i resides with the exporters rather than the importer.) Access to $d_{i,c}$ will be limited to the signatories of the contract c.

More specifically, any access granted to $d_{i,c}$ must be granted in c itself.

We think of the existence of $d_{i,c}$ as bound up in c. It does not exist before c is in effect nor after c is dissolved. In fact, we will regard a subsequent modification of c as the dissolution of c followed by the creation of a new contract.

As we pointed out, our formal notion of a contract is asymmetric. The reason for doing this is to be able to associate a unique d_i with each contract. We will use the term *pact* to denote informal agreements between organizations concerning data sharing. We envision that typical pacts, even those focused on a single issue, may require several formal contracts. The same organization may well play both the role of importer and exporter in a pact.

The framework will be built on top of the underlying database query language used by the organizations, which we will denote by $\mathcal{L}$. This language may be SQL-like or some kind of object-oriented language. Its exact definition is not important for the framework. Later we will mention a few very minimal requirements for $\mathcal{L}$. If different organizations are using different languages, then we will imagine that they are all embedded in $\mathcal{L}$ in a coherent way. We will not pursue the issue of heterogeneity any further here. We also assume that each organization has the *potential* to execute a command[1] against any set $\{D_{i_1}, \ldots, D_{i_n}\}$. Contracts will determine how much of this potential is to be realized.

In a distributed database setting, given a command σ in $\mathcal{L}$, it is sometimes assumed that to execute σ one must additionally specify the database or set of databases against which it is to be executed. Specifically, for each table reference one would associate a particular database. For the sake of notational brevity we adopt the opposite strategy: we assume that σ carries with it the relevant database references.[2]

Associated with each command σ in $\mathcal{L}$ is a set of databases called the *support* of σ, written

$$\mathrm{supp}(\sigma),$$

which includes all the databases that may be accessed during the execution of the command. By "database" we mean a D_i or a $d_{i,c}$. We say "may"

[1]We think of a command as a *request* to apply a function (such as the SQL `select`) that manipulates data in a database to arguments (such as a portion of a table) that specifies data in a database.

[2]If an actual language really works in the first way, we would simply regard one of our commands as one of those commands with the necessary database references.

rather than “will” since complex commands may contain branches or be data dependent. Thus databases explicitly mentioned in σ may not be accessed, while databases not explicitly mentioned may be. While we assume for the sake of our framework that $\mathrm{supp}(\sigma)$ is given, determining sharp values of the support of σ would probably be difficult in practice and one would be forced to conservative estimates.

A contract c with importer O_i and exporters $O_{j_1}, \ldots, O_{j_n}$ specifies a set of *(positive) access statements*

$$+(O_k, \sigma)$$

where:

- O_k is the importer or any one of the exporters.
- If O_k is the importer, then $\mathrm{supp}(\sigma) \subseteq \{D_i, D_{j_1}, \ldots, D_{j_n}, d_{i,c}\}$ but $\mathrm{supp}(\sigma) \neq \{D_i\}$.
- If O_k is an exporter, then $\mathrm{supp}(\sigma) = \{d_{i,c}\}$ or $\mathrm{supp}(\sigma) = \{D_k, d_{i,c}\}$.

It is understood that each member of O_k will only have access to those commands in $\mathcal{L}$ to which it is explicitly granted access by c, with the exception that it may, of course, execute any statement that involves only D_k. All basic and compound statements to be allowed must be explicitly specified in c, and no additional combining is allowed. Notice that, if a contract c contains an access statement $+(O_k, \sigma)$, then O_k as well as each O_j, such that $D_j \in \mathrm{supp}(\sigma)$ or $d_{j,c} \in \mathrm{supp}(\sigma)$, must be a party to c. While in a real implementation it will be important to develop a language to conveniently specify these sets of statements, it is best for now to think of these sets as being given extensionally. Also, the set of access statements specified by a contract is allowed to be infinite. (We discuss a language for specifying sets of access statements in §9.)

Using commands σ in access statements $+(O_k, \sigma)$ provides more flexibility than replacing σ with tables or attributes. Suppose, for example, we have some natural notion of subcommand in $\mathcal{L}$ (e.g., τ is a *subcommand* of σ if τ occurs in σ and τ is itself a command) and the contract c stipulates $+(O_k, \sigma)$. It will not generally be the case that c will also stipulate $+(O_k, \tau)$ where τ is a subcommand of σ.

It is also possible to specify more finely who exactly in an organization is allowed to execute a particular command σ by using roles or even personal IDs. Since facilities for using this technique are not specific to federated database systems, we will assume that they are provided by $\mathcal{L}$ itself, and we will not consider this matter further in our general remarks.

We now state the minimal requirements for $\mathcal{L}$. In addition to providing standard delete and update capabilities, we require that $\mathcal{L}$ have three different kinds of "select" commands:

(1) `Select_read_only` permits viewing via an operation that is trusted not to copy or pass information to other untrusted operations.

(2) `Select_create` permits the creation of a table in some d_k (but not necessarily full access to it).

(3) `Select_own` permits the creation of a table in some D_k (with full access to it).

This is the entire Basic Framework. Notice that we do not provide for obligations in a contract. How then do we achieve the effects for which obligations and their associated actions were intended? First, obligations to take certain actions may be incorporated directly into $\mathcal{L}$. For example, $\mathcal{L}$ may provide the means to construct a command that updates an audit log each time it is called, thereby satisfying a given auditing obligation. Second and most important, passive obligations concerning the sharing and protection of imported data are handled by the compartmentalized (in the nontechnical sense) way that data is stored.

Since the effectiveness of contracts will depend to a great extent on the underlying framework which "hardwires" certain safeguards, rather than just on the terms of individual contracts, it is worth reiterating the framework in the special case of a single exporter. Let us assume we have a contract c between importer O_i and exporter O_e. Then c will specify certain access statements $+(O_i, \sigma)$ and $+(O_e, \tau)$ where $\mathrm{supp}(\sigma) \subseteq \{D_i, D_e, d_{i,c}\}$ but $\mathrm{supp}(\sigma) \neq \{D_i\}$, and $\mathrm{supp}(\tau) = \{d_{i,c}\}$ or $\mathrm{supp}(\tau) = \{D_e, d_{i,c}\}$.

- O_i will be explicitly allowed certain accesses to D_e and $d_{i,c}$, possibly in conjunction with D_i.
- O_e will be explicitly allowed certain accesses to $d_{i,c}$, possibly in conjunction with D_e.
- (O_i and O_e are always allowed full access to D_i and D_e, respectively.)

3 Contract Interactions

We wish to consider how contracts written in our framework can interact with one another. Before doing so, let us return to a more intuitive level to see how contracts, unconstrained by our framework, might interact. Two contracts are in *conflict* when one requires a certain action and the other prohibits it. That is, one contract contains an obligation and the other a prohibition which directly clash. A logically weaker form of interaction between contracts, which we call an *impingement*, occurs when one contract permits an action and the other prohibits it.

Since the Basic Framework contains neither explicit obligations nor explicit prohibitions, there is no possibility of conflict or impingement under this definition—at least at some formal level. This state of affairs warrants further investigation since one would intuitively expect the possibility of conflicts and certainly impingements. While the Basic Framework has no facility for specifying particular prohibitions, it does provide the structural prohibition governing $d_{i,c}$. However, since this prohibition is rigidly enforced and can never be overridden, it cannot be the cause of impingements. Later in the Full Framework (given in §6) we will introduce an explicit facility for specifying prohibitions, and this will indeed introduce the possibility of impingements.

While the Basic Framework, as well as the Full Framework to follow, do not offer a separate category of obligations, they do allow us to simulate the effect of having obligations, if $\mathcal{L}$ is sufficiently rich. Imagining obligations as separate entities, one would naturally think of granting access rights contingent upon the guarantee to fulfill certain obligations. For example, an exporter might grant an importer unlimited access to a certain command provided that the importer recorded each use in some audit log. In our approach the exporter would instead grant the importer unlimited access to a command which itself executed both the desired command and the log entry. A good analogy here is the distinction between a rental car company requiring that a car is always driven less than 65 miles per hour and the company installing a governor in the car to physically prevent the car from being driven over 65 miles per hour while requiring no cooperation on the part the driver.

What obligations can be effectively built into $\mathcal{L}$ is an important open question which we will not pursue in this paper.

There is a related phenomenon which should not be confused with conflicts and impingements. It is possible for commands of $\mathcal{L}$ authorized under different contracts to "bump into each other." That is, the effect of executing one command may depend on previous executions of other commands. For that matter, two commands authorized under the same contract could bump into each other. More generally, it might be difficult to understand the behavior of a single command of $\mathcal{L}$, especially if $\mathcal{L}$ is rich. The point is that this problem is attributable to $\mathcal{L}$ itself and is not exacerbated by the contracts in force.

4 Implementation Strategy

An important virtue of the framework is that there is an obvious and straightforward method for implementing it. One just places a guard between the user and the database system that permits and prohibits the execution of commands in $\mathcal{L}$ in accordance with the contracts in force. The objective is to make the guard simple and easy to understand. If, for each contract c in force, there is an algorithm which determines the membership in the set of $+(O, \sigma)$ for c, then a guard can easily be constructed by combining these algorithms. Of course, a general algorithm may or may not take into account the state of a database and even the real world. We will be interested in the case when the set of $+(O, \sigma)$ is decidable (in the sense of computability theory) for each contract c. Then, not only is there is an algorithm for testing which commands can be executed, but in addition the algorithm defines a specific immutable set which does not depend on the state of any database or the real world.

Recall that under the framework accesses are granted absolutely. However, much as how obligations can be simulated within the commands of $\mathcal{L}$ as discussed above, conditional access privileges can be simulated by granting access (absolutely) to a command in $\mathcal{L}$ which incorporates the conditional. For example, instead of granting access to some command σ on weekends only, one could grant unconditional access to a command τ which would first check the day of the week (assuming that is available in the database) and then execute σ if it is a weekend day and otherwise return an error message. In fact, using this approach may allow one to effectively capture a conditional access that one could not expect to capture otherwise. Suppose that we wished to grant access only to commands that run for less than one second. It is probably not feasible to precompute what these commands would be. Instead, we would have a command that would run along with a clock that would cut off after one second and return an error message.

There is a possible confusion which we wish to avoid. The language $\mathcal{L}$ should be thought of as fixed. When one organization requests access to some (simple) command and another organization only grants access to some related (complicated) command which might involve extra tests, etc., this does not mean that $\mathcal{L}$ has changed. Both the requested and granted commands are commands in $\mathcal{L}$. We are not suggesting that $\mathcal{L}$ be modified to accommodate such fine points, but rather that $\mathcal{L}$ is expressive enough in the first place to capture these fine points as they arise.

5 Examples

We pause to give examples of how the framework could be used to create some small, but interesting contracts.

5.1 Hospital/Research Example

A hospital is willing to give access to its records to some research organization which is allowed to use this data for scientific purposes only. Together the parties write a contract to this effect, which we call c.

Suppose there is a table called Hospital_patient_records in the hospital's database D_{hos}. The research organization is permitted by c to make a copy of the table Hospital_patient_records in $d_{\text{res},c}$ using a `Select_create` command. Let us call this table Research_patient_records. There are two reasons for this choice. First, the research organization wishes to freeze a particular state of Hospital_patient_records, while the original copy in D_{hos} is "live" and will be constantly evolving. Second, the research organization should not be given full access to this table, and in particular, should not be allowed to extract anything "personal" concerning it. Finally, c permits the research organization to execute certain "statistical" type commands of $\mathcal{L}$ on Research_patient_records.

To expand upon this example, let us suppose that the research organization has similar contracts with many other hospitals. These contracts allow the insertion of the results of statistical commands into a table in D_{res}. The contents of this table are not considered to be of a confidential nature by the respective hospitals, and so the research organization is given full control of this table. Notice that by the principles of the framework the research organization does not have permission to execute any command that simultaneously involves more than one of the corresponding $d_{\text{res},c}$.

Here is a variation on this example. In this version the research organization is permitted to use a `Select_own` command to create a table in D_{res} over which it has thus complete control. Forming this table involves having access to Hospital_patient_records which presumably includes an attribute for the name of the patient. The created table, however, does not include this attribute but only the social security number, which is also presumably included in Hospital_patient_records. This table is deemed sufficiently anonymous to require no further protection, but it contains enough information to identify patients across hospitals. (We are tacitly assuming that there is no facility for obtaining a person's name from his social security number and that it is infeasible to infer a person's name from his social security number or other data in his hospital record.)

5.2 Airline/Travel Agent Example

FBN Airlines maintains a database that includes two tables. The first, FBN_flight_info, contains entries for each flight number and date, giving, among other information, the number of seats available at each price. The second, FBN_reservations, has a listing for each flight reservation, giving the flight number, date, name of the traveler, price, etc. In particular, there is a column in this table for the booker of the reservation. For security reasons, that column takes values which are, in effect, passwords known only within the booking agency that did the booking.

FBN Airlines enters into the following contract with each independent travel agency with which it does business. In these contracts, FBN is the exporter and the individual travel agents are importers. A travel agent may execute commands of $\mathcal{L}$ to read any of the information in the first table, with the exception of those entries concerning spaces held for free flights for frequent flier participants, which may only be accessed by FBN personnel.

A travel agent may insert a reservation in the second table, corresponding to any of the available seats he can see in the first table. He may copy such a reservation back into his own main local database so that he has easier access to it and can use it for his own records. He may also modify or delete a reservation in the second table so long as he is the original booking agent. It is not feasible to misuse the booking agent column since we are assuming that these values are closely held within the particular agencies, and because only certain values will be accepted under that attribute.

Naturally, some sort of transaction discipline is required to keep the two tables properly coordinated. However, this is handled as usual and is not a concern of the contract. Notice that there is no need for $d_{i,c}$ databases.

5.3 Juvenile Record Example

There are two organizations: the Drug Enforcement Agency (DEA) and the Attorney General (AG). From the point of view of this contract (called c), DEA is the importer and AG is the exporter. Among the tables in D_{AG} there are AG_case_recs (with attributes case_#, person_id, person_type, active_status, and investigator_id) and AG_persons (with attributes person_id, name, phone_#, address, birth_year, birth_month, birth_date, and juvenile_status). DEA is allowed to execute a command that will transfer the data from AG_case_recs into a new table DEA_case_recs (with attributes case_#, person_id, person_type, active_status, and investigator_id) in $d_{\text{DEA},c}$. However, the investigator_id column will initially be left null and the DEA will not have access to this column in AG_case_recs. It is intended that the DEA will insert its own investigators' IDs in this column. The DEA is also allowed to execute a command that will transfer all the data from AG_persons into a new table DEA_persons in $d_{\text{DEA},c}$.

According to certain regulations, when juvenile offenders reach the age of 18, their case records are to be deleted if they have no current open criminal case. The AG is required by law to seal the records of such individuals. Following our parsimony principle, exactly how the AG makes this determination is not essential to the contract. The only concern relevant to the contract is that certain deletions take place on the required records in $d_{\text{DEA},c}$. This can be implemented in a number of different ways. For example, AG could be granted access to the necessary delete commands. AG might perform such deletions manually or, perhaps, install a trigger in D_{AG} that would perform the deletions when required. Even more autonomously, one could imagine that each time DEA transfers a record the same command installs a trigger in the DEA database to monitor these deletions. Just what method would actually be followed would be specified in the contract.

5.4 Producer-Consumer

This is a little example without a story to make a small point that might otherwise go unnoticed. A contract c is written between an importer O_i and an exporter O_e. Under this contract O_e gets full access to $d_{i,c}$ including write privileges. O_i has certain access privileges to $d_{i,c}$ but no access privileges to D_e directly. Thus the data is "pumped" by the exporter to the importer rather than “sucked” from the exporter by the importer. The database $d_{i,c}$ plays the role of a shared buffer between D_e and D_i.

6 The Full Framework: Relinquishing Rights

Under the Basic Framework presented above, an organization O_k maintains the right to execute any command $\sigma \in \mathcal{L}$ such that $\mathrm{supp}(\sigma) = \{D_k\}$. In addition, O_k may be granted the privilege to execute some commands with other support. In the real world, the value of certain information is directly related to the fact that the information is not widely known. Quite often the potential danger of promulgating this kind of information only becomes significant when that information can be combined with some other information. As a result, an organization O_i may wish to control the access of another organization O_j to information that O_i does not control. For such purposes, we offer a primitive statement

$$-(O, \sigma)$$

which says an organization O may not execute the command σ. We call these *negative access statements.* We now present the *Full Framework.*

A contract in the Full Framework may contain both positive and negative access statements. The positive access statements are used in the same way as in the Basic Framework. The negative statements are used to revoke an access right either voluntarily by the recipient or by any of the holders of resources on which the command depends. More precisely, a negative access statement $-(O_k, \sigma)$ may only be used in a contract c in the following three ways:

(1) O_k is a party to c and $\mathrm{supp}(\sigma) = \{D_k\}$. In this case, O_k relinquishes its intrinsic right to execute σ.

(2) O_k is a party to c and $\mathrm{supp}(\sigma) \neq \{D_k\}$. In this case, O_k relinquishes its right to execute σ, even if it should be granted this right. It is not required that each organization with a database in $\mathrm{supp}(\sigma)$ be a party to c.

(3) O_j is a party to c and $D_j \in \mathrm{supp}(\sigma)$ or $d_{j,c'} \in \mathrm{supp}(\sigma)$ for some contract c'. In this case, O_k need not be a party to c and O_j agrees not to grant O_k the right to execute σ. It is not required that each other organization with a database in $\mathrm{supp}(\sigma)$ be a party to c.

Observe that, though we have separated cases (1) and (2), they are similar. Notice that negative access statements $-(O_k, \sigma)$ may appear in a contract c only if either O_k is a party to c or, for some party O_j to c, $D_j \in \text{supp}(\sigma)$ or $d_{j,c'} \in \text{supp}(\sigma)$ for some contract c'. Also, in cases (2) and (3), there may be organizations O_i with $D_i \in \text{supp}(\sigma)$ that are not a party to c. This is in contrast to the awarding of access rights which requires the consent of all interested parties.

6.1 Monogamy

Two organizations O_m and O_f decide to canonize their exclusivity in sharing information. They enter into a pact consisting of two contracts c_m and c_f. O_m is the importer and O_f is the exporter in c_m. In c_m, O_m is granted full access rights to D_f. In addition, O_m renounces its right to award any access rights to D_m to any organization other than O_f. That is, O_m agrees not to be an exporter in any other contract. There is no need for $d_{m,c}$ in this contract. The contract c_f is completely analogous to c_m with m and f interchanged. We will discuss how these contracts might be formalized later.

We could also imagine a more stringent form of monogamy in which O_m also agrees not to be an importer in any other contract, and O_f does the same.

7 More about Conflicts and Impingements

While the Full Framework is still not subject to conflicts (since there are still no obligations), it is clearly subject to impingements (as defined in §3). Let us suppose that a web of contracts already exists and some members of the federation are in the process of negotiating a new contract. They have arrived at a putative contract. They would like to be able to accomplish the following:

Detection. Determine if the putative contract creates any impingements with any of the existing contracts.

Identification. If so, identify the causes of the impingements.

Privacy. Do this with a minimal amount of sharing of contractual information.

Recall that we are always operating under the following defaults:

- Access by O_k to a command σ where $\text{supp}(\sigma) = D_k$ is assumed, and can only be relinquished explicitly by a contract to which O_k is a party.
- Access to any $d_{k,c}$ is limited to the parties to c and only as explicitly granted by c.
- Access by O_k to a command σ can only be relinquished contractually by O_k or by some O_j where $D_j \in \text{supp}(\sigma)$, or $d_{j,c} \in \text{supp}(\sigma)$ for some contract c.

Let c be a contract between an importer O_i and some set of exporters. Related to the notion of impingement are two weaker types of impingement that involve only a single contract. (A bona fide impingement requires two distinct contracts.) The first occurs when c contains a statement of the form $-(O_k, \sigma)$ where $\text{supp}(\sigma) = \{D_k\}$. The second occurs when c contains statements $+(O_i, \sigma)$ and $-(O_i, \sigma)$. While these interactions do not represent contract violations, they might occur inadvertently as a result of a misunderstanding of a contract specification, and so should be checked for carefully.

Genuine impingements that result from c can occur only if one of the following four cases holds:

(1) c grants $+(O_i, \sigma)$.

  (a) For some other contract c' to which O_i is a party, c' stipulates $-(O_i, \sigma)$. (O_i can detect this impingement by looking at the contracts to which it is a party.[3])

  (b) There is some other contract c' and some organization O_k with $k \neq i$ such that c' stipulates $-(O_i, \sigma)$, O_k is a party to c', and $D_k \in \text{supp}(\sigma)$ or $d_{k,c''} \in \text{supp}(\sigma)$ for some contract c''. Notice that O_k must also be a party to c and that, if $d_{k,c''} \in \text{supp}(\sigma)$, $c'' = c$. (O_k can detect this impingement by looking at the contracts to which it is a party.)

(2) c stipulates $-(O_i, \sigma)$.

[3]We are tacitly assuming here that there is a feasible algorithm that, given a contract $c' \neq c$, will decide whether c' stipulates $-(O_i, \sigma)$.

(a) O_i is a party to c and, for some other contract c', c' grants $+(O_i, \sigma)$. Notice that O_i must also be a party to c'. (O_i can detect this impingement by looking at the contracts to which it is a party.)

(b) There is some other contract c' and some organization O_k with $k \neq i$ such that c' grants $+(O_i, \sigma)$, O_k is a party to c, and $D_k \in \text{supp}(\sigma)$ or $d_{k,c''} \in \text{supp}(\sigma)$ for some contract c''. Notice that O_k must also be a party to c' and that, if $d_{k,c''} \in \text{supp}(\sigma)$, $c'' = c'$. (O_k can detect this impingement by looking at the contracts to which it is a party.)

Observe that we seem to have achieved the privacy goal already. Specifically, all impingements can be revealed by each organization examining those contracts to which it is a party. In particular, when a new contract is being contemplated, any resulting impingements can be detected in this way by the parties to that contract. We call this the *Localization Property.*

Before proceeding to the detection and identification goals, we introduce some convenient notation. Given a contract c, we define two sets $\mathcal{S}_c^+$ and $\mathcal{S}_c^-$ as follows:

$$\mathcal{S}_c^+ = \{(O, \sigma) : +(O, \sigma) \text{ is stipulated by c}\}.$$

$$\mathcal{S}_c^- = \{(O, \sigma) : -(O, \sigma) \text{ is stipulated by c}\}.$$

Also, let $\mathcal{I}_{c_1,c_2} = \mathcal{S}_{c_1}^+ \cap \mathcal{S}_{c_2}^-$. Typically, these sets will be infinite and specified intentionally. How they will be specified is the concern of the contract language, which we discuss in §9 below.

As we have observed above, an impingement occurs when $\mathcal{S}_{c_1}^+ \cap \mathcal{S}_{c_2}^- \neq \emptyset$ for two distinct contracts c_1 and c_2. (If $c_1 = c_2$ we are considering the weaker types of impingement mentioned above.) Thus, determining whether or not an impingement occurs is equivalent to determining if $\mathcal{S}_{c_1}^+ \cap \mathcal{S}_{c_2}^- = \emptyset$ for all distinct contracts c_1 and c_2.

To understand the rest of this section requires some familiarity with computability and automata theory (as, e.g., presented in [1]). Notice first that, if we require that each $\mathcal{S}_c^+$ and $\mathcal{S}_c^-$ is decidable, then each intersection $\mathcal{I}_{c_1,c_2}$ will also be decidable. (This only means that there is some algorithm which, given a pair (O, σ), will determine whether or not it is a member of $\mathcal{I}_{c_1,c_2}$.) However, the question of whether or not an intersection $\mathcal{I}_{c_1,c_2}$ is empty is, in general, undecidable.

Let us suppose instead that $\mathcal{I} = \mathcal{I}_{c_1,c_2}$ is specified as a context-free language (in the sense of Chomsky). It is well known that determining whether or not such an $\mathcal{I}$ is empty is decidable [1, Theorem 6.6]. Thus, the assumption that $\mathcal{I}$ is always context-free would satisfy the detection goal. Moreover, it would satisfy the identification goal as well, since we would have a context-free representation, i.e., a BNF, for $\mathcal{I}$ which would allow us to visualize the members of $\mathcal{I}$.

We seem to have uncovered a satisfactory condition on $\mathcal{I}$. In order to achieve our objective we must translate this condition back to conditions on $\mathcal{S}^+_{c_1}$ and $\mathcal{S}^-_{c_2}$. The most obvious approach, requiring all $\mathcal{S}^+_c$ and $\mathcal{S}^-_c$ to be context free, is not adequate since the intersection of two context free languages is not context free in general [1, Theorem 6.4]. However, the intersection of a context-free language L_1 with a regular language L_2 is context free, and there is a simple algorithm for constructing a context-free specification from the given specifications for L_1 and L_2 [1, Theorem 6.5].

Since we must consider all intersections, we must require that all the $\mathcal{S}^+_c$ are context free and all the $\mathcal{S}^-_{c_2}$ are regular, or vice versa. We have not given serious consideration to which option is preferable.

8 Warranties

So far all of our contracts have traded only in access rights to commands of $\mathcal{L}$. Little has been said about the semantic significance of those commands. If an importer were paying an exporter for certain access rights, it would be only natural for the importer to want some sort of guarantee that when he executed a command the results of doing so would be what he expected. As a simple example, if an importer who had subscribed to some stock trading database were to execute a command like

```
price(IntPap, current)
```

it would get the current price of a single share of International Paper stock (whatever "current" is determined to mean). More generally, we now consider contracts in which the exporter provides a warranty concerning the semantic significance, relative to the external world, of executing various commands against its database. There are two aspects of semantics that we have in mind, though the distinction between them is not always completely clear. The first involves the semantics of $\mathcal{L}$ in the sense of programming language semantics. The second involves the meaning of the data in particular databases in the sense of its significance in the external world. It is this second aspect that we intend to consider here. We imagine the first as being the subject of the "standards committee" of the federation and not to be considered on a contract-by-contract basis.

8.1 Library Example

An organization O has a contract with a library L for library services. These include certain remote accesses to the library's database D_L. O can check the holdings (what L owns) and availability (what currently is in the library) of particular items in L. L guarantees that the commands intended for these purposes actually provide the correct information. Specifically, L might agree to update its table of holdings on a daily basis and agree to update the availability attribute of an item before allowing it to leave the library. In addition, members of O may reserve items through D_L, provided of course that they are available. In particular, if the library accepts such a reservation through D_L, it guarantees that the item will not be checked out by anyone else until the reservation expires and that the item was part of its holdings. (O might want L to guarantee that the item is actually in the library and can be found, but this is more difficult to guarantee.)

8.2 Conflicts

In order for L to make good on its guarantees, it will have to update D_L in a timely manner so that it faithfully reflects the current contents of the library. In addition, it will need to employ some kind of transaction discipline, especially if access is to be allowed to more than one user at a time. It may be that L does not do this correctly, and this would result in a violation of the contract. However, there is no question of a conflict of contracts here since there is only one contract after all. One can, however, find contract conflict situations of a very simple kind. Suppose that an exporter gives access to the same command on its database to two different importers. Suppose also that the exporter guarantees that this command means "X" to one importer and guarantees that it means "Y" to the other importer. If these meanings are incompatible, then there is a sense that these two contracts are in conflict. However, we do not regard this as a conflict in the same sense as we have been discussing above. There is no question of resolving conflicting goals at execution time. The command has been implemented in a certain way, and thus the command must violate either one or both of the contracts.

8.3 Comparison

In the actual world of federated database systems, warranties of this kind could be very important. However, the concerns they introduce are very different from those presented by the Basic or Full Frameworks, which are themselves quite similar. In negotiating a contract c under the Basic or Full Framework, the parties naturally take into account semantics in both of the aspects we have mentioned above. But the actual contract c itself should be thought of as specifying only syntactic constraints. For that reason we have available the relatively simple implementation strategy presented in §4. Conversely, because this strategy is available, it is important to have these contracts "computer readable" in some sense so that we can actually take advantage of the implementation strategy. In the case of warranties, it is the semantic constraints themselves that would appear in the warranty. We have no general strategy for enforcing such constraints, which should not be surprising given the difficulty of handling semantic constraints in an single unfederated database system. Therefore, there is less need to capture such warranties in computer-readable form, at least at present.

We imagine a full federated database contract consisting of two parts. The first part would correspond to a contract in the Full Framework and could be thought of as an input to the guard of the implementation strategy. The second part, which would correspond to the warranty and which might be expressed less formally, would serve as a guide to implementors and offer legal protection to the parties of the contract.

9 Remarks on the Languages

Underlying everything we have discussed above is a database query language $\mathcal{L}$. Remember that $\mathcal{L}$ serves as a parameter in the framework, and changes to $\mathcal{L}$ should not necessitate changes to the framework.

There is a second language $\mathcal{L}_{\mathrm{c}}$ about which we have said much less. This is the language specifically designed for writing contracts. For the most part this language will be used for describing sets of positive and negative access statements. A typical statement of $\mathcal{L}_{\mathrm{c}}$ might express that the exporter will not grant access rights to any command of $\mathcal{L}$ involving the table "cabal" to any organization that is not on a certain list. If warranties are also to be used with the framework, $\mathcal{L}_{\mathrm{c}}$ will need statements to express them, but these statements are likely to be quite different from the statements of $\mathcal{L}_{\mathrm{c}}$ associated with the Full Framework. As yet, we have not given much thought to the construction of $\mathcal{L}_c$, but one would want to have such a language in order to make the writing of complex contracts practical. The main purpose of $\mathcal{L}_c$ is to make it easy to capture useful subsets of $\mathcal{L}$ commands and collections of organizations in the federation, and to facilitate the detection of conflicts between access statements.

In order to appreciate one of the issues that $\mathcal{L}_c$ should address, we return to the monogamy example (see §6.1). Suppose we try to formalize the requirement that O_m is not allowed to be the exporter in any contract. One might try to formalize this by considering the set of all statements $-(O_x, \sigma)$ where $x \notin \{m, f\}$ and σ is any command. This would be a solution to the problem, albeit a long-winded one, provided the federation remained static. Suppose that, when new commands are added or new organizations join the federation, the monogamy agreement is intended to apply to these as well. Then the above will not be sufficient. The analogous problem with positive access statements might be encountered as D_m and D_f grow. What is needed here is some facility in $\mathcal{L}_c$ for writing schemas in the logical sense (such as the schema for mathematical induction). These schemas would involve variables and constants that would range over entities such as organizations, commands, tables, command constructors etc. While the intention of the contract is fixed when it is written, the extensional meaning of the contract may change over time. Even if one did not choose to include such *dynamic* schemas in $\mathcal{L}_c$, one certainly would want as a minimum some notion of a *static* schema in $\mathcal{L}_c$. In fact, that is the point of $\mathcal{L}_c$.

There is a particular construct that might be included in $\mathcal{L}_c$ to achieve some of the effect of schemas. It would assert that some organizations' positive accesses would be limited to a certain set of commands. This would allow us to express directly what would amount to a schema of negative access statements.

While our observations in §7 on detecting impingements will still apply to the interpretation of contracts at any fixed time, it is possible to construct contracts using dynamic schemas which do not impinge when they are written, but which would impinge were the federation to change appropriately. As a result one should be careful as to if and how dynamic schemas should be used.

For practical reasons one would also want still another language layer. Because of contract requirements, when a user wishes to execute some simple command, he may in fact have to execute a much more complicated command that surrounds the simple command with various tests and actions. It would probably be unrealistic and unnecessary to burden the average user of the database with all these details. One could instead create a language $\mathcal{L}_\mathrm{i}$ to act as an interface to the real database language $\mathcal{L}$. $\mathcal{L}_\mathrm{i}$ would provide convenient syntactic mechanisms for selecting commands. For example, in the case described above it would provide the average user with a command that looked like the simple command he had in mind, but which really invoked a more complicated command. As a result the user is spared some of the burden of conforming to the contracts. We imagine that the different members of the federation might have their own interface languages, which would be strongly determined by the contracts at hand.

10 Final Remarks

Some readers may find our framework a spartan one. For example, we do not have a certain class of active obligations, and so in particular, we do not need to be concerned with whether the carrying out of an obligation would be in violation of some other contract. Allowing restrictions and obligations to combine in an unstructured way leads to a very complicated semantics for contracts for which it is not clear how to create a convenient abstraction barrier. In order to get an idea of the sorts of issues that would need to be addressed, consider the following.

Suppose an action is taken that activates several different contracts and requires tests or actions from a number of them. In what order are these tests or actions to be scheduled? In what environments are the tests or actions scheduled later to be executed? In the current environment, or in the one in existence when the activity was first required? What if one action negates the condition for another action that was originally true? Of course, just dealing with the first test or action may itself instigate still other activities. Cascading problems are possible in which, rather than contradicting one another, contracts may interact to generate an infinite sequence of actions. All these problems would be compounded with the difficulty of dealing with distributed databases and concurrent users.

Suppose it were possible to develop a framework that settles all these issues, as well as many yet to be anticipated. Then, it would, in general, be necessary to be cognizant of these details in order to understand the real meaning of a contract. We feel that this would be an undesirable situation, reminiscent of the worst aspects of low-level Prolog programming. We consider it a very significant point that our framework is, in some sense, immune from this problem. Suppose, for the sake of argument, that $\mathcal{L}$ was either enhanced or replaced by another language $\mathcal{L}'$. Further suppose that $\mathcal{L}'$ was improperly designed so that the meaning of certain expressions was either difficult or even impossible to determine in certain situations. Naturally, this incomprehensibility would be inherited by contracts written in our framework. However, it is not further compounded by our framework, and one certainly could not hope to do better than this. The approach we have taken provides an abstraction barrier between our framework and the underlying database language $\mathcal{L}$. The latter serves as a parameter in the framework, that we can freely change without modifying the framework. What the framework assumes about $\mathcal{L}$ is minimal. Obviously, this is a desirable quality of the architecture.

Under our framework the effect of a command does not depend upon the contracts in force except to the extent that they control access to the command. In addition, one can understand the effect of a contract in isolation independently from any other contracts. With unconstrained combinations of actions and obligations, it is impossible to understand the effect of a single contract in isolation. Even more strikingly, in order to predict the effect of even a single database command, one would need to take into account all the contracts in force.

In the foreword we mentioned three criteria against which our framework should be measured. These may be described as, expressivity, implementability, and understandability. We feel that the framework we have presented above is at least satisfactory with respect to the second and third. More empirical evidence will be required to determine how well it does against the first.

References

[1] Hopcroft, J. E., and J. D. Ullman, 1979, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley.

[2] Sheth, A. P., and J. A. Larson, 1990, "Federated database systems for managing distributed, heterogeneous, and autonomous systems," *ACM Computing Surveys*, Vol. 22, pp. 183–236.

Acknowledgments

This work was supported by the MITRE-Sponsored Research program under the Secure Federated Data Management project. Mark Nadel presented part of the paper at a special session entitled “Applied Logic” at the 101st Annual Meeting of the American Mathematical Society held 4–7 January 1995 in San Francisco, California.