A Data Sharing Agreement Framework

Vipin Swarup, Len Seligman, and Arnon Rosenthal

The MITRE Corporation
7515 Colshire Drive
McLean, VA 22102
{swarup, seligman, arnie}@mitre.org

Abstract. When consumers build value-added services on top of data resources they do not control, they need to manage their information supply chains to ensure that their data suppliers produce and supply required data as needed. Producers also need to manage their information supply chains to ensure that their data is disseminated and protected appropriately. In this paper, we present a framework for data sharing agreements (DSA) that supports a wide variety of data sharing policies. A DSA is modeled as a set of obligation constraints expressed over a dataflow graph whose nodes are principals with local stores and whose edges are (typed) channels along which data flows. We present a specification language for DSAs in which obligations are expressed as distributed temporal logic (DTL) predicates over data resources, dataflow events, and datastore events. We illustrate the use of our framework via a case study based on a real-world data sharing agreement and discuss issues related to the analysis and compliance of agreements.

1 Introduction

Data sharing refers to the act of letting another party use data. It has become prevalent with the spread of Internet technologies such as email, websites, and P2P systems. Computer users routinely share ideas, documents, multimedia files, etc. while organizational entities share large-scale data sets, e.g., scientific datasets, counter-terrorism information, etc. Four critical capabilities that are essential to any data sharing system are: a means for providers and consumers to discover each other; a means for consumers to access data; a means for consumers to understand data; and a means to manage sharing policies. In this paper, we will focus on issues regarding cross-boundary sharing policies.

In organizations, sharing provides value by improving the ability of information recipients to achieve mission objectives. However, sharing requires information to cross various kinds of boundaries and this carries risks, e.g., risks of disclosure of the information to adversaries, and risks of compromise of the sources and methods used to collect the information. In addition to these mission benefits and risks of sharing, information providers also have incentives and disincentives to share. Possible incentives include prestige, satisfying legal requirements or organizational directives, rewards, and altruism, while disincentives include fear that consumers might misuse or fail to protect the data or not give due credit to the data's creator or provider.

The decision to share involves a trade-off between the risks of sharing and the risks of not sharing information. These risks can change dramatically when providers and consumers have confidence that certain obligations will be met by their sharing partners. This is greatly facilitated by policy management, which provides a mechanism to express and realize desired behavior of a data sharing arrangement (e.g., access controls, constraints on usage and subsequent sharing, guarantees about data quality and availability, compensation, penalties for violating agreements).

At the sharing transaction level, sharing policies have typically focused on meeting protection goals, e.g., usage control policies that are enforced using techniques such as digital rights management (DRM) engines and trusted computing platforms. At the organizational level, data sharing policies are expressed in documents called Memoranda of Understanding (MOUs) or Memoranda of Agreement (MOAs) that document the obligations and expectations of parties to the agreement. Current practice is to use textual memoranda, an approach with several disadvantages. First, there is little help for the writers, so important sharing issues are often omitted. Second, the documents are typically filed away in a drawer and seldom used thereafter. Third, it is hard to provide automated support for reasoning about the contents of textual memoranda.

Service Level Agreements (SLAs) have emerged as the paradigm used by organizations to express policies regarding service-level parameters. For instance, for network provider services, SLAs describe obligations over parameters such as availability, latency, throughput, packet loss, etc. Similarly, for data sharing services that focus on the sharing of information, SLAs can describe a variety of obligations over a similar set of parameters. However, there are several key aspects of data sharing services that are not addressed by traditional SLAs that focus on functional business services. First, data sharing obligations may require a provider to actively engage in actions that result in wider sharing of its data. These obligations may include parameters such as data freshness and quality, regular update dissemination, etc. Second, data sharing obligations may require a data recipient to engage in certain actions, e.g., further share the data, share derivatives of the data, share audit records of actions that it invoked on the data, notify the provider when the data is shared with a third party, etc. Third, obligations may restrict what the recipient may do with the data, e.g., whether the recipient can print a document. Finally, data objects subsume other data objects and this property can be exploited—e.g., an agreement about European Union persons' data is relevant to a demand for German persons' data.

Obligation policies are a key component of SLAs. An *obligation* is a constraint that a principal commits to satisfy in the future. Obligations are distinct from provisions (i.e., actions that a principal takes in order to gain access to a resource) and expectations (i.e., actions that one principal expects another to satisfy). Unlike access control rules which must always be satisfied, obligations may be violated. Hence, obligations are typically associated with penalty clauses that describe the consequences of violating the obligations. Obligation platforms provide support for obligation policies, maintain representations of obligations, enforce the satisfaction of (some) obligations, monitor and detect violations of obligations, and manage penalties associated with obligation violations.

In prior work, we have proposed the notion of a *data sharing agreement* (DSA) [19] as a variant of SLAs, and we have proposed an initial model for DSAs [20]. In this paper, we elaborate on our model for *data sharing agreements* (DSAs) that encode data sharing obligations. We focus on obligations about data stores and data flows, although the model can be extended to express obligations about the collection and processing of data, the use of data by parties, etc. We represent data stores as collections of typed values, dataflows as data streams between principals, and obligations as temporal constraints on data store and data stream events. Our model was developed by studying several real-world Memoranda of Agreements (MOAs) that are used by large U.S. government organizations to capture data sharing obligations. We present a specification language based on the model, as well as an example specification of a DSA that is motivated by a real-world MOA. We sketch a variety of analyses that are enabled by our approach and discuss technical challenges related to compliance. We conclude with a comparison with related work and future directions enabled by this work.

2 Data Sharing Agreements (DSAs)

DSAs share many aspects of SLAs, for instance, descriptions of the parties to the agreements, availability constraints, and temporal constraints like agreement lifetimes [14]. They also inherit from the general notion of agreements, that a party may incur obligations in return for benefits. For example, a consumer may promise to pay cash, to refrain from certain activities (e.g., non-compete agreements), or even to supply the consumer's own information to competitors. However, the primary purpose of DSAs is to capture data sharing clauses including descriptions of the data being shared, and obligations that constrain both the providers and consumers of the data and the data flows among them. The data may be described using standard techniques such as relational schema, XML schemas or DTDs, or object classes.

Obligations on data providers can concern both the need to send certain data (which the provider must then produce or acquire) and the quality of what is sent. They include recency constraints (data will reflect real world events, or data updates will be forwarded, with specified promptness or periodicity); visibility constraints (access will be provided to specified data views); and quality constraints (shared data will be of specified freshness, accuracy, precision, etc.).

Obligations on data consumers include protection constraints (e.g., data will not be copied, data will be deleted after 3 days, etc.); dissemination or sharing constraints (e.g., original source will be credited, subsequent research products based on the data will be shared back with the data providers; data will be shared with certain other consumers; providers will be notified of who has accessed the data); and generic security constraints (e.g., providers will be notified of security breaches in consumer networks). Note that parties may be data consumers in one

DSA, and providers (e.g., of derived products) in another. Parties may even be both data consumers and providers in the same DSA (e.g., providers of logistical data and consumers of end-user audit data).

All obligations may be conditional on events (e.g., receipt of a data object, detection of attack or compromise, etc.) and state predicates (e.g., declaration of local emergency, system failures, relationship among data values). Further, data sharing agreements may include obligations whose fulfillment depends on other DSAs or SLAs. For instance, suppose that B is obliged to ensure that C receives fresh and accurate data at regular intervals. In order to meet this obligation, B might rely on DSAs with data suppliers and SLAs with network providers. Finally, parties to a DSA may agree to inherit obligations from other DSAs.

A DSA's obligations impose global constraints on access control policies of various parties, even in future states. The exact nature depends on how the parties meet the obligations. For instance, suppose that a data recipient B is obligated to share data updates back with the provider A. Then either A must be given access rights to the data, or B must allow release and create a process that pushes the update notifications to A.

3 DSA Model

A DSA pertains to specific data schemas and data instance sets. Data may be represented in any *data model* that provides constructs to represent data, and operations over those constructs (e.g., query, update). Three prevalent data models are the relational data model, the XML data model, and the object data model. Our language is based on the data model of the Cω research programming language [6] which integrates the above three models into a single succinct framework. It provides a uniform model for representing relational data (e.g., database tables), semi-structured data (e.g., XML documents), object data (e.g., Java objects), and data streams (e.g., sequences). It also provides a uniform means for expressing queries over data. We shall use the syntax of Cω in our examples and will provide informal descriptions of the semantics [6, 1, 2].

A DSA is an agreement between a set of principals regarding the sharing of data among themselves. In this context, data sharing refers to the explicit flow of data from one principal to another, and not to the subtler notions of information flow or data inference. Hence, we model a DSA as a set of sharing obligations expressed over a dataflow graph whose nodes are principals with local stores, and whose edges are (typed) channels along which data flows.

Principals can be specified in well-understood ways and include both simple principals (e.g., named individuals or organizations) and compound principals (e.g., groups, roles, etc.). Each principal is associated with a local data store. A *data store* is a function that maps (location) names to data values. The data values may be data relations, data streams, documents, objects, etc. as mentioned above.

Data resources describe the data to which a data sharing agreement pertains. A data resource is a tuple $\langle rn, p, DV \rangle$, where rn is a globally unique data resource name, p is the principal offering the data resource, and DV (i.e., a data view) is a set of tuples $\langle q, T\rangle$, where q is a query language expression over p's local data store and T is the type of the query's result. A data resource might describe both the data content that is being shared, and metadata such as the sharing context and attributes of principals.

A *dataflow* [15] F is a tuple $\langle s, d, T\rangle$ where s and d are principals and T is a type. F represents a data stream of values of type T flowing from source s to destination d. The source s can place value o of type T into the dataflow stream (which we write as $F.send(o)$, while the destination principal d can read values from the stream ($F.receive(o)$). The state of a dataflow F is a tuple $state(F) = \langle f, r\rangle$ where $f \in T^*$ is the sequence of values that have been placed in the stream, and $r \in T^*$ is the sequence of values that F's destination principal has read from the stream. Data values can include message identifiers to capture which values in the stream have been received.

An *obligation* O is a pair $\langle \psi, p\rangle$ where ψ is a formula in Distributed Temporal Logic (DTL) [9,12] and p is the penalty that is incurred if ψ is not satisfied. Obligation formulae specify properties of traces of dataflow events (sending and receiving data) and data store events (updating data stores). DTL is a generalization of Linear Temporal Logic (LTL) and includes both past-time and future-time temporal operators including **Y** (*previous*), **P** (*sometime in the past*), **H** (*always in the past*), **S** (*since*), **X** (*next*), **G** (*always*), and **U** (*weak until*). A DTL formula can refer to a specific principal's local data space and hence can be true only for that specific principal. Thus, the DTL formula $@_a[\psi]$ asserts that proposition ψ holds in the local context of principal a. A *penalty* can be any action imposed on a principal, e.g., the payment of money by one principal to another. In this paper, for simplicity, we assume that each event corresponds to a fixed time step (e.g., 1 hour or 1 day per time step). An alternate approach would be to have explicit clocks (e.g., a local clock per principal) and predicates over the clocks.

Finally, a *data sharing agreement DSA* is a tuple $\langle P, S, DS, DR, DF, O\rangle$ where P is a set of principals (i.e., the parties referenced in the agreement), $S \subseteq P$ is the set of signatories of the agreement, DS is a function that maps each principal in P to a data store (that represents the local data store of the principal), DR is a set of data resources that are views of the data stores in DS, DF is a set of dataflows between principals in P, and O is a set of obligations of principals in P. Note that only principals in S are signatories of the DSA and hence have agreed to the terms of the obligations that oblige them.

4 DSA Specification Language

We now sketch a specification language for DSAs based on the above model. A data sharing agreement specification includes the parties to the DSA, time bounds for validity, a timestep duration (to specify the fixed time duration of each event), a set of statements that specify the data resources referenced in the DSA, and a set of obligations expressed over the data resources, the local data stores of the parties, and dataflows among the parties. Parties and time bounds are expressed in the usual ways (e.g., using X.509 distinguished names, timestamps, etc.). In this section, we will present a language to specify data stores, dataflows, and obligations.

As mentioned earlier, we use the data model of the Cω research programming language [6]. We shall rely on informal descriptions of Cω language constructs in this paper. Cω is a typed language with the usual primitive types (**boolean**, **integer**, **string**, and **void**) and four structured types: streams, which consist of zero or more members; anonymous structs, which are like structs in C but with the possibility of multiple fields with the same name; choice types, which are discriminated unions (i.e., a value of a choice type consists of one of the listed field types and a value of that type); and classes, whose members are objects that encapsulate data values and methods.

A snippet of our DSA language is presented in Figure 1. Informally, the statement "**Channel**(P,Q,T) c" declares c to be a (dataflow) channel from P to Q that carries values of type T. The expressions *c*.**src** and *c*.**dest** denote the source and destination of channel *c* respectively, while **getloc**(Q,x) denotes the value bound to location x in Q's data store. The predicates *c*.**send**(*o*) and *c*.**rcv**(*o*) are true if object *o* was just sent and received (respectively) on channel *c*, while the predicate **setloc**(Q,x,e) holds if Q's data store was just updated to bind location x to value e.

Expressions are typed terms that evaluate to a value of the appropriate type. The Cω expression language contains usual terms for boolean, integer and string literals, built-in operators, object and structure creation, and method invocation. The power of the language comes from the overloaded "dot" term. If e evaluates to a stream v, then $e.\{h\}$ returns the stream created by applying the expression h to each member of v. h may contain the special variable *it* which gets bound to each successive element of v. The language does not permit nested streams, and hence if h returns a stream as result, then the resulting stream of streams is "flattened" to a simple stream (see below). If e evaluates to a structure v, then $e.f$ returns the values of the fields named f in structure v. Since a structure may contain multiple fields with the same name, $e.f$ returns a stream of values. Finally, if e evaluates to a structure v, then $e[i]$ returns the value of the i'th field in structure v.

For instance, let:

x = { city = "Rome" ; ctry = "USA" ; ctry = "Italy" ; }
y = { city = "Paris"; ctry = "USA" ; ctry = "France" ; }
z = { info = x; info = y; }

Then, x.city is the string "Rome", x[2] is the string "USA", while x.ctry is the stream with two values "USA" and "Italy". z.info is the stream of the two structure values (x and y), while z.info.{it.ctry} is the flattened stream with four values ("USA", "Italy", "USA", and "France").

Statement terms include variable declarations and assignments, conditionals, loops, method invocations and returns, and blocks. For streams, the language provides a dot statement $e.h$ that applies the expression h to each element of the stream yielded by evaluating e (again, the variable *it* is bound to each successive value of the stream); a loop construct ("foreach") for iterating over all elements of a stream; and a "yield" term that returns a member of a stream. For instance, continuing with the above example, x.ctry.{print it;} applies the print method to each member of the x.ctry stream, hence printing out the two country names "USA" and "Italy".

The obligation language (see Figure 1) is based on distributed temporal logic. Formulas are evaluated at a specific temporal and distributed state. An obligation term includes a DTL formula, a predicate that specifies the cancellation policy for the obligation, and the penalty for violating the obligation formula. We use syntactically sugared DTL operators, e.g., (ϕ **innext** t) specifies that ϕ must hold sometime in the next t timesteps, while (ϕ **at** B) specifies that ϕ must hold in the local state of principal B.

Finally, a DSA is specified by a name, a set of principals, a sequence of statement and obligation terms, start and end timestamps representing the validity period of the DSA, and a timestep that specifies the time interval between successive states in the temporal model.

5 Examples

We now present several examples of sharing obligations that can be expressed in our framework. For brevity, we have taken significant liberty with the syntax here, e.g., by omitting certain clauses such as cancellation and penalty clauses in obligations, and by using time intervals and timestamps rather than time-step counts. In these examples, let A, B, and C be principals; $c1$ be a channel (of type T) from A to B; $c2$ be a channel from B to C; and $c3$ be a channel from B to A.

Responsive forwarding: B will send C on channel $c2$ each object it receives from A on channel $c1$ within 24 hours of receiving it:

```
(forall T o) if c1.rcv(o) then
    (c2.send(o) innext 24 hrs
at B until 1/1/2007
```

Nondisclosure agreements: If B receives object o from A on channel c1, then B will not thereafter send o to any other principal for a year.

```
(forall T o) if c1.rcv(o) then
    (forall Channel c) if (c.dest ≠ A) then
        (not c.send(o) until 1 year)
at B until 1/1/2007
```

Usage notification: If B receives object o from A on channel c1, then for the next 365 days, B will notify A each time B sends o to another principal. We construct the notification object via an externally defined function called "notify".

s	::=	. . . **Channel**(P,Q) c . . .	
e	::=	c.**src** | c.**dest**	
	|	**getloc**(Q,x) | . . .	
p	::=	c.**send**(e) | c.**rcv**(e)	
	|	**setloc**(Q,x,e) | . . .	
ob	::=	obf **cancelif** p **penalty** penalty	Time-bound obligation
obf	::=	p	Simple predicate
	|	obf **and** obf	Conjunction
	|	obf **or** obf	Disjunction
	|	**not** p	Negation
	|	(**forall** τ x) obf	Universal quantification
	|	(**exists** τ x) obf	Existential quantification
	|	obf at Q	obf holds in the local data space of principal Q
	|	**if** p **then** obf	Conditional obligation formula
	|	(obf)	Precedence brackets
	|	obf **innext** i	obf holds sometime during the next i states
	|	obf **until** i	obf holds for the next i states
	|	obf **until** p	obf holds until p holds
	|	obf **atnext** i	obf holds at the i'th next state
	|	obf **inprev** i	obf held sometime during the previous i states
	|	obf **fromprev** i	obf held during all the previous i states
	|	obf **fromprev** p	obf held since the last time that p held
	|	obf **atprev** i	obf held at the i'th previous state
DSA	::=	**DSA** id **with** **principals** $\overline{\mathrm{Q}}$ **statements** $\overline{\mathrm{s}}$ **obligations** $\overline{\mathrm{ob}}$ **from** timestamp **to** timestamp **timestep** duration	Data sharing agreement

Fig. 1. DSA Language Snippet

```
(**forall** T o) **if** c1.rcv(o) **then**
   (**forall** Channel c)
      **if** (c.send(o) **and** c.dest $\neq$ A) **then**
         c3.send(notify(B,c.dest,o)) **innext** 1 day
   **until** 365 days
**at** B **until** 1/1/2007
```

Recurrence: A will send B the latest update to object o every 24 hours.

```
**if** c1.send(o) **then**
   c1.send(update(o)) **innext** 1 day
**at** A **until** 1/1/2007
```

Privacy: Privacy obligations may arise when a person or an organization shares personal data with another organization. They may also arise when an organization receives data about a human, imposed by government (via laws) rather than the provider.[1] The simple privacy obligation specified below asserts that A must delete all personal information about B from location x within one year of A's storing it.

```
(**forall** T e) **if** setloc(A,x,e)
      **and** (e.subject = B) **then**
   setloc(A,x,null) **innext** 1 year
**at** A **until** 1/1/2007
```

The above examples can be combined to form complex obligations. For instance, if B receives data O from A, then B will not share O with foreign citizens, and will notify A about US citizens who are shown the data. Furthermore, even for US citizens, B will not reveal that A was the source. Our model is quite powerful and can capture a wide variety of such data sharing policies. To illustrate this, we now describe a sanitized and substantially simplified fragment of a data sharing agreement that is based on a real U.S. government Memorandum of Agreement (MOA). We have changed organization names and altered their missions to facilitate release of this information. The agreement is between the military's fictional Antarctica Command (USANTCOM), which maintains a logistics database we shall call AntLog, and the fictitious Logistics Information Agency (LIA), which consolidates logistics information from many sources into the Logistics InfoMart (LIM) and provides analysis on logistics requirements and feasibility of both hypothetical and actual operations. We have augmented the agreement with additional terms not found in the real MOA in order to illustrate important features of data sharing agreements.

MOA: The DSA in this example addresses data services that permit authorized LIM users to access the AntLog database service via a LIM data brokering service. The brokering service exposes specific data views to LIM users

[1] Some obligations (e.g., those imposed by law) are implicit—they are not part of any explicit DSA even though the parties are obliged to satisfy them. Such implicit obligations may be specified in our model although they are not the focus of this paper.

and only permits them to execute specific queries over those data views. The AntLog database service is hosted on AntLog database servers, while the brokering service is hosted on LIM data brokering servers.

USANTCOM must notify LIA whenever the AntLog database schema is changed or a new data feed into AntLog is created; this notification must occur 30 days in advance of the change. AntLog engineers must provide technical assistance to LIA to create and optimize the SQL queries that LIM users are permitted to execute against the AntLog database. USANTCOM must provide the LIA manager of the LIM data brokering service with credentials to access a group account on AntLog.

LIA must ensure that all LIM users of the LIM data brokering service are authenticated, and that only cleared and authorized users can use the provided group credentials to access AntLog via the data brokering service. Further, LIA must audit all such successful transactions. LIA is also responsible for maintaining valid (interim or final) accreditation status for all LIM systems and networks and must ensure that there are no actual or probable security breaches on those systems and networks. Finally, LIA must notify USANTCOM and AntLog administrators of any violation of these assertions, and the penalty will be that USANTCOM may disconnect AntLog from all affected systems/servers.

Figure 2 contains a specification of the data brokering DSA. In this instantiation, dbs is a data view of AntLog and represents a relational table whose rows consist of a resource descriptor ("rsrc"), current location ("curloc"), and destination location ("destloc"). query1 returns the current locations of all resources, while query2 returns all resources which are not yet at their destination. The obligations encode the constraints described above.

6 Analysis and Compliance

A formal data sharing model enables us to reason about a range of properties:

1. Is it possible for me to satisfy all my obligations? E.g., we're obliged to share all data with a business partner but we receive product documentation under a stringent nondisclosure agreement; is there a sequence of events that will satisfy my obligations? If we assume that all data types are finite, and if we restrict ourselves to the LTL fragment of our language, then satisfiability is tractable and is answerable by standard model checkers (e.g., NuSMV).

2. Do I need a new DSA, or are my information needs already covered by existing DSAs? This can be formulated as subsumption, i.e., does an obligation entail another? Again, if we assume that all data types are finite, and if we restrict ourselves to the LTL fragment of our language, then entailment is

```
DSA InfoMart with
  principals USANTCOM, LIA, AntLog, Broker,
    LIM, LIMusers
  signatories USANTCOM, LIA

  statements
    Channel(Broker,AntLog) cba;
    Channel(USANTCOM,LIA) cul;
    Channel(LIA,USANTCOM) clu;

    struct { String rsrc; String curloc; String destloc } * dbs;
    dbs = getloc(AntLog,tracker).{ {rsrc = it.rsrc; curloc = it.cloc;
                                    destloc = it.dloc; } };
    String query1 = dbs.{it.curloc};
    String query2 = dbs.{ if it.curloc ≠ it.destloc then it.rsrc };

  obligations
    if dbs.update(u) then
      cul.send(notify("schema update")) inprev 30 days

    if clu.rcv("credentials", "AntLog") then
      cul.send(credentials("AntLog",uname,passwd))
        innext 24 hrs

    if cba.send(q) then
      (q == query1 or q == query2) and
      (exists Principal Q) (exists Channel(Q,Broker) cqb)
        (cqb.rcv(q) and cqb.authenticated()) inprev 24 hrs
        and setloc(Broker,audit,getloc(Broker,audit).add(Q,q))

    if LIM.system.securitybreach() then
      clu.send(notify("breach", "LIM")) innext 2 hrs
    penalty USANTCOM may disconnect AntLog


  from 6 May 2003
  to indefinite
```

Fig. 2. USANTCOM–LIA Data Brokering DSA

tractable and is answerable by standard model checkers.

3. What actions must I take in order to meet all my obligations? Which customers must I send data to, or notify about potential changes to my data schema or to the frequency of data updates? Can we automatically generate dataflows from a DSA that satisfy the DSA's dataflow obligations? This is largely an unexplored problem since, unlike access control policies, obligations are not always enforceable. In fact, obligation formulae place constraints on future behavior and they may well be violated by parties, e.g., due to changes in external circumstances. Obligations carry penalties that describe actions to be taken when the obligation formulae are violated. Irwin et al. [13] formulate compliance in terms of accountability, namely whether a given policy can ensure that the system can hold principals accountable for obligation constraint violations.

7 Related Work

Several researchers have proposed policy languages for expressing obligations for data protection. Park and Sandhu [18] propose a usage control model that constrains how a data consumer can use data. Bettini et al. [4, 5] propose an access control model based on provisions and obligations, where a reference monitor creates obligations while making an authorization decision. In contrast to this body of work, our focus is on data sharing obligations, not on data protection obligations. Thus, for instance, we support obligations that require data consumers to share data further with other consumers; such obligations may not be associated with any authorization decision.

Ponder [8] and PDL [7] use event-condition-action (ECA) rules to express obligations. In those models, obligations are actions that are triggered immediately when certain events occur and certain predicates are satisfied. In contrast, our model is based on a temporal logic that can place temporal constraints on when obligations must be satisfied. For instance, our model can express obligations that are activated when certain events occur (or predicates are satisfied) in the future.

Bettini et al. [3] propose an obligation model in which events are represented as predicates; an event predicate is true in a state iff the corresponding event occurs in that state. We take a similar approach but we also incorporate a general data model and a dataflow model, and our work is motivated by several real-world MOAs.

Gama and Ferreira [11] describe a policy language (xSPL, Extended Security Policy Language) and enforcement platform (Heimdall) for specifying and enforcing complex policies, including history-based and obligation-based policies. Our focus, in this paper, has been on the data specific aspects of such policies.

Several systems use deontic logic [17] to express obligations. Other systems are based on temporal logics that enable the specification of temporally-constrained obligations. We have adopted Hilty et al.'s distributed temporal logic framework [12].

A distinction between our work and prior work is that we distinguish between expectations and obligations. Formulas that express constraints on future actions are expectations; they become obligations only when the obliged parties accept them as obligations, for instance, by signing them. In contrast, most previous work does not require obliged parties to accept (or agree to) the obligations.

Firozabadi et al. [10] address the sharing of scarce resources and they address conflicts between entitlements, obligations, and resource scarcity. Data, on the other hand, may be freely replicated. Thus, it is an opportunity rather than a conflict if many parties want the same data item. Subsumption and derivation of data schemas and queries are well-understood problems (e.g., [16]) and can be exploited in sharing data with multiple parties.

Irwin et al. [13] present an analysis and modeling approach for obligations. They present the notion of accountability as a key goal for obligation mechanisms. In their model, obligations are defined as actions that subjects are obliged to take on a set of objects, within a time window. They provide detailed analyses for determining whether a system will remain accountable, given obligation and authorization policies. Obligations in our model are at a much higher abstraction level, and it would be interesting to map our abstract obligation specifications to low-level action specifications.

8 Conclusion

In this paper, we have presented a model for data sharing agreements. The model is based on a dataflow graph whose nodes are principals with local stores, and whose edges are (typed) channels along which data flows. Data sharing constraints are expressed as DTL predicates over data stores and data flows. These constraints can include both *past* events and *future* events, and may hold only at certain principals' local states. We have argued why this approach is central to the problem of secure information sharing, one of the most fundamental problems in information security.

Our work has been motivated by several real-world data sharing agreements (currently expressed as textual Memoranda of Agreement). We illustrated the expressive power of our language by expressing a sanitized fragment of a real MOA. Since our language can be given a precise formal semantics, DSAs specified in our language are amenable to a variety of automated analyses.

This work is an important first step towards our ultimate goal of building a comprehensive technical infrastructure for secure information sharing policies. We are beginning to develop a prototype platform that manages and enforces DSAs. This platform will include wizards to assist in the creation of comprehensive and consistent DSAs; repositories to assist in the lifecycle management of DSAs; distributed agents to monitor and enforce the terms of the DSAs, when possible; and modules to provide automated analysis capabilities.

Note that obligations are binding on the obliged parties and the parties may be subject to penalties for failing to meet their obligations. Hence, obligation systems are subject to a variety of attacks by adversaries. Attacks include creating obligations that are not undertaken by the obliged principals, freeing principals from their obligations, causing principals to violate their obligations, and preventing obligation monitors from detecting violations of obligations. Hence, a secure DSA system must protect against such attacks while managing DSAs and monitoring for potential violations of obligations.

References

1. Arvind Arasu and Jennifer Widom. A denotational semantics for continuous queries over streams and relations. *SIGMOD Record*, 33(3):6–12, 2004.
2. Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 1–16, 2002.
3. Claudio Bettini, Sushil Jajodia, Xiaoyang Sean Wang, and Duminda Wijesekera. Obligation monitoring in policy management. In *3rd IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2002)*, pages 2–12, June 2002.
4. Claudio Bettini, Sushil Jajodia, Xiaoyang Sean Wang, and Duminda Wijesekera. Provisions and obligations in policy management and security applications. In *VLDB*, pages 502–513, 2002.
5. Claudio Bettini, Sushil Jajodia, Xiaoyang Sean Wang, and Duminda Wijesekera. Provisions and obligations in policy rule management. *J. Network Syst. Manage.*, 11(3), 2003.
6. Gavin M. Bierman, Erik Meijer, and Wolfram Schulte. The essence of data access in Cω. In *Proceedings of the 19th European Conference on Object-Oriented Programming (ECOOP 2005)*, volume 3586 of *Lecture Notes in Computer Science*, pages 287–311. Springer, 2005.
7. Jan Chomicki, Jorge Lobo, and Shamim A. Naqvi. Conflict resolution using logic programming. *IEEE Trans. Knowl. Data Eng.*, 15(1):244–249, 2003.
8. Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The Ponder policy specification language. In *POLICY '01: Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, pages 18–38, London, UK, 2001. Springer-Verlag.
9. Hans-Dieter Ehrich and Carlos Caleiro. Specifying communication in distributed information systems. *Acta Inf.*, 36(8):591–616, 2000.
10. Babak Sadighi Firozabadi, Marek J. Sergot, Anna Cinzia Squicciarini, and Elisa Bertino. A framework for contractual resource sharing in coalitions. In *Proceedings of the 5th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2004)*, pages 117–126, 2004.
11. Pedro Gama and Paulo Ferreira. Obligation policies: An enforcement platform. In *Proceedings of the 6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2005)*, pages 203–212, 2005.
12. Manuel Hilty, David Basin, and Alexander Pretschner. On obligations. In *10th European Symposium on Research in Computer Security (ESORICS 2005)*, volume 3679 of *Lecture Notes in Computer Science*, pages 98–117. Springer, 2005.
13. Keith Irwin, Ting Yu, and William H. Winsborough. On the modeling and analysis of obligations. In *To appear in Proceedings 13th ACM Conference on Computer and Communications Security*, 2006.

14. Alexander Keller and Heiko Ludwig. The WSLA framework: Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management, Special Issue on E-Business Management*, 11(1), March 2003.
15. Gary T. Leavens, Tim Wahls, and Albert L. Baker. Formal semantics for SA style data flow diagram specification languages. In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, pages 526–532, 1999.
16. Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 95–104, 1995.
17. John-Jules Ch. Meyer, Roel Wieringa, and Frank Dignum. The role of deontic logic in the specification of information systems. In *Logics for Databases and Information Systems*, pages 71–115. Kluwer, 1998.
18. Jaehong Park and Ravi Sandhu. The $\mathrm{UCON}_{\mathrm{ABC}}$ usage control model. *ACM Transactions on Information and System Security*, 7(1):128–174, 2004.
19. Len Seligman, Arnon Rosenthal, and James Caverlee. Data service agreements: Toward a data supply chain. In *Workshop on Information Integration on the Web, at VLDB 2004*, 2004.
20. Vipin Swarup, Len Seligman, and Arnon Rosenthal. Specifying data sharing agreements. In *Seventh IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'06)*, pages 157–162, Los Alamitos, CA, USA, 2006. IEEE Computer Society.