SecPAL4DSA: A Policy Language for Specifying Data Sharing Agreements

Benjamin Aziz[1], Alvaro Arenas[2], and Michael Wilson[3]

[1] School of Computing, University of Portsmouth, Portsmouth, U.K.
benjamin.aziz@port.ac.uk
[2] Department of Information Systems, Instituto de Empresa Business School, Madrid, Spain
alvaro.arenas@ie.edu
[3] e-Science Centre, STFC Rutherford Appleton Laboratory, Oxfordshire, U.K.
michael.wilson@stfc.ac.uk

Abstract. Data sharing agreements are a common mechanism by which enterprises can legalise and express acceptable circumstances for the sharing of information and digital assets across their administrative boundaries. Such agreements, often written in some natural language, are expected to form the basis for the low-level policies that control the access to and usage of such digital assets. This paper contributes to the problem of expressing data sharing requirements in security policy languages such that the resulting policies can enforce the terms of a data sharing agreement. We extend one such language, SecPAL, with constructs for expressing permissions, obligations, penalties and risk, which often occur as clauses in a data sharing agreement.

Keywords: Data Sharing Agreements, Deontic Logic, Security Policies.

1 Introduction

Data sharing is becoming increasingly important in modern enterprises. Every enterprise requires the regular exchange of data with other enterprises. Although the exchange of this data is vital for the successful inter-enterpriseal process, it is often confidential, requiring strict controls on its access and usage. In order to mitigate the risks inherent in sharing data between enterprises, Data Sharing Agreements (DSAs) are used to ensure that agreed data policies are enforced across enterprises [1].

A DSA is a legal agreement (contract) among two or more parties regulating who can access data, when and where, and what they can do with it. DSAs either include the data policies explicitly as clauses, or include existing enterpriseal data policies by reference. DSA clauses include deontic notions stating permissions for data access and usage, prohibitions on access and usage which constrain these permissions, and obligations that the principles to the agreement must fulfill. A DSA can be created between an enterprise and each of many collaborators. DSAs are represented in natural languages with their concomitant ambiguities and potential conflicts, which are exacerbated by DSA combinations. Therefore, analysing such natural language DSAs [2] is desirable before they can be enforced, which is usually done through a transformation into executable policy languages [3].

J.J. Park et al. (Eds.): STA 2011, CCIS 186, pp. 29–36, 2011.

This paper is mainly concerned with defining an approach for the modelling and enforcement of DSAs based on the SecPAL policy language [4]; a simple and powerful language defined by Microsoft Research. The approach involves enriching SecPAL with deontic predicates expressing permissions and obligations. These predicates can then be used to model DSA clauses. Second we extend the definition of clauses to associate them with penalties, in case clauses are violated. Finally, we provide a risk-based approach for calculating SecPAL queries, which takes into consideration the level of assurance of the infrastructure or the reputation of the user and the penalty associated with clauses.

There has been a fresh interest in the concept of data sharing agreements, motivated mainly by models exploiting Internet as a technological platform for businesses, in which sharing data and information is central. An initial model of DSAs was proposed by Swarup et al in [5]. The model is based on dataflow graphs whose nodes are principals with local stores, and whose edges are channels along which data flows. Agreement clauses are modelled as obligation constraints expressed as distributed temporal logic predicates over data stores and data flows. In [6], an operational semantics expressing how a system evolves when executed under the restriction of DSA is defined, and in [9], the model is encoded in the Maude term-rewriting system and several DSA properties are automatically verified.

Our work has been influenced by the work of Arenas et al. [2] on using the Event-B specification language for modelling DSAs. There, a method is defined for representing deontic concepts into event-based models, and how those models can be exploited to verify DSA properties using model-checking techniques. However, our work goes further by using an implementable specification language, SecPAL, and by including quantitative factors such as penalties and their associated risk. Closer to our approach is the work on using SecPAL for modelling privacy preferences and data-handling policies [8]. The work focuses on defining a notion of satisfaction between a policy and a preference with the aims of developing a satisfaction-checking algorithm. By contrast, our work has concentrated on giving precise formal definitions to the notions of permissions and obligations as key concepts in DSAs.

2 Overview of Data Sharing Agreements

Data sharing requirements are usually captured by means of collaboration agreements among the partners. They usually contain clauses defining what data will be shared, the delivery/transmission mechanisms and the processing and security framework, among others. Following [5], a DSA consists of a definition part and a collection of agreement *Clauses*. The definition part includes the list of involved *Principals*; the start and end dates of the agreement; and the list of *Data* covered by the agreement.

Three types of clauses are relevant for DSAs: *Authorisation*, *Prohibition* and *Obligation* clauses. Authorisations indicate that specified roles of principals are authorised to perform actions on the data within constraints of time and location. Prohibitions act as further constraints on the authorisations, prohibiting actions by specific roles at stated times and locations. Obligations indicate that principals, or the underlying infrastructure, are required to perform specified actions following some event, usually within a time period. The DSA will usually contain processes to be

followed, or systems to be used to enforce the assertions, and define penalties to be imposed when clauses are breached.

The set-up of DSAs requires technologies such as DSA authoring tools [6], which may include controlled natural language vocabularies to define unambiguously the DSAs conditions and obligations; and translators of DSAs clauses into enforceable policies. We represent DSA clauses as guarded actions, where the guard is a predicate characterising environmental conditions like time and location, or restrictions for the occurrence of the event, such as "*user is registered*" or "*data belongs to a project*".

Definition 1: (**Action**). An action is a tuple consisting of three elements $\langle p,an,d\rangle$, where p is the principal, an is an action name, and d is the data.

Action $\langle p,an,d\rangle$ expresses that the principal p performs action name an on the data d. Action names represent atomic permissions, where actions are built from by adding the identity of the principal performing the action name and the data on which the action name is performed. We assume that actions are taken from a pre-defined list of actions, possibly derived from some ontology. An example of an action is "*Alice accesses product data*". We shall consider in this paper two types of clauses only: permissions and obligations. Clauses are usually evaluated within a specific context represented by predicates for environmental conditions like location and time.

Definition 2: (**Agreement Clause**). Let G be a predicate and $a = \langle p,an,d\rangle$ be an action. The syntax of agreement clauses is defined as follows:

C ::= IF G THEN $P(a)$ | IF G THEN $O(a)$

A permission clause is denoted as IF G THEN $O(a)$, which indicates that provided the condition G holds, the system may perform action a. An obligation clause, on the other hand, is denoted as IF G THEN $O(a)$, which indicates that provided the condition G holds, the system eventually must perform action a. A data sharing agreement can be defined as follows.

Definition 3: (**Data Sharing Agreement**). A DSA is a tuple $\langle$*Principals*,*Data*, *ActionNames, fromTime,endTime,* $\wp(C)\rangle$

Principals is the set of principals signing the agreement and abiding by its clauses. *Data* is the data elements to be shared. *ActionNames* is a set containing the name of the actions that a party can perform on a data. *fromTime* and *endTime* denotes the starting and finishing time of the agreement respectively; this is an abstraction representing the starting and finishing date of the agreement. Finally, $\wp(C)$ is the set of clauses of the data sharing agreement.

3 SecPAL4DSA: SecPAL for Data Sharing Agreements

SecPAL [4] is a declarative authorization language with a compact syntax and a succinct unambiguous semantics. It has been proposed for modelling various security policy idioms, such as discretionary and mandatory access control, role-based access control and delegation control, and within the scope of large-scale distributed systems

like Grids [7]. It has also been recently suggested as a framework for modelling privacy preferences and data-handling policies [8].

A policy in SecPAL is represented as an *assertion context*, *AC*, which is simply a set of *assertions*, $A \in AC$, written according to the following syntax:

$AC ::= \{A_1,\ldots,A_n\}$
$A ::= E$ `says` $fact_0$ `if` $fact_1 \ldots fact_n$ `where` c
$fact ::= e$ `pred` $e_1 \ldots e_m \mid e$ `can say` $fact$
$e ::= x \mid E$

Where *c* is a constraint in the form of a logical condition, *E* is some principal entity (such as a user or a system process), *x* is a variable, *fact* is a sentence stating a property on principals in the form of the predicate, e `pred` $e_1 \ldots e_m$ or allowing delegations in the form of e `can say` *fact*. A main feature of SecPAL is that it is extensible in the sense that any set of predicates e `pred` $e_1 \ldots e_m$ can be added to a specific instance of SecPAL. Such predicates will be defined based on the specific domain for which the policies are required, for example, in the domain of scientific experiments, predicates could include *A canVisualise(data)* or *A canAnalyse(data)*.

The main contribution of this paper is to propose a new instance of SecPAL for expressing assertions and queries on DSAs, called SecPAL4DSA. SecPAL4DSA extends the syntax of facts of the previous section with the following predicates:

$fact ::= permitted((an,d)) \mid obliged_{user}((an,d)) \mid obliged_{sys}((an,d))$

The $permitted((an,d))$ predicate implies that the user is permitted to execute the action *an* on the data *d*. On the other hand, $obliged_{user}((an,d))$ means that the user is obliged to execute *an* on *d* sometime in the future. Here, we do not deal with real-time temporal constraints on obligations, though these would be possible to incorporate. Finally, $obliged_{sys}((an,d))$ means that the system infrastructure is obliged to execute *an* on *d* sometime in the future.

The extra predicates we define above represent the deontic operators of permissions and obligations for users and systems. Note that at present, SecPAL does not allow negative predicates in the language of assertions because of issues related to the complexity of assertion deductions. Therefore, we do not deal with prohibitions and assume that any action not permitted is prohibited by default.

Example 1. In the context of DSAs, a principal *E* may represent any of the *signatories* of the DSA. So, for example, assuming there are two signatories, *A* and *B*, then the following two assertions express a DSA that demands payments for data accesses:

(**Assert1**) *A says B permitted*(*access*, *data*) `if` *A hasFinished*(*authenticate*, *B*) `where` *c*
(**Assert2**) *A says B* $obliged_{user}$(*pay*, *amount*) `if` *B permitted*(*access*, *data*), *B hasFinished*(*access*, *data*) `where` $currentTime \leq PaymentDeadline$

The first assertion will allow *A* to grant *B* the permission to *access data* if *A* successfully authenticates *B* (modelled by the predicate *hasFinished*) and some condition *c* holds (which could be relevant to some of *B*'s attributes). The second assertion states that whenever *B* has finished accessing the data, then it is obliged by *A* to pay an amount of money within some predefined deadline.

Next, we define precisely what we mean by the permitted and obliged deontic operators in the context of SecPAL4DSA.

3.1 Semantics of the Permission and Obligation Predicates

The semantics of permissions, user obligations and system obligations are given here in terms of Linear Temporal Logic (LTL). LTL formulae are defined based on a set of propositional variables, p_1,p_2 …, which are in our case the predicates themselves, and logical operators ($\neg$, $\wedge$, $\vee$, $\rightarrow$) including future temporal operators such as $\Box$ (always), $\Diamond$ (eventually) and $\bigcirc$ (next). They may also include past versions of these. Here, we shall write $P \Rightarrow Q$ to denote the formula $\Box(P \Rightarrow Q)$, and we write the two-way conditional, $B \lfloor A \rceil C$, to denote that if A is true, then C is true, otherwise if A is false, then B is true.

1. *Semantics of User Obligations*: We start with the semantics of user obligations. This semantics is defined in terms of two actions: *bF* and *bS*. *bF* represents a system action corresponding to the failure of executing an action by the user and *bS* is a system action corresponding to the success of the user in executing some action. *bF* would normally represent a penalty action that the system will execute in case of user failure. On the other hand, *bS* is a follow-up action that the system executes after the user successfully fulfils his obligation to carry some action. Either or both of *bF* and *bS* could be inactive actions, such as (*null*, *null*). However, more meaningful examples would be for *bF* to be the disabling of some system resources, bF=(*disable*, *resource*), and for *bS* to be the enabling of system resources, *bS*=(*enable*, *resource*). Now, assuming that $p(a)$ is the post condition of action a; this means that a has already occurred and its effect on the state of the system is modelled as the predicate $p(a)$, we can define the semantics of B $obliged_{user}(a)$ (i.e. B is obliged to do a) in terms of the semantic function, [*fact*]=*P* defined as follows:

   $[B\ obliged_{user}(a)] = \exists bF,\ bS\colon G \Rightarrow \Diamond\ ([Sys\ obliged_{sys}(bF)]) \lfloor p(a) \rceil [Sys\ obliged_{sys}(bS)]$

   where G is a general predicate on the current state of the system enabling and could include for example p(request), which is a predicate indicating that the action representing a request from the user to execute a has occurred. This meaning of user obligations is defined in terms of the meaning of system obligations, which we discuss next. The rationale behind this is that user obligation cannot be enforced; users by their nature can always violate an obligation. However, such violations can trigger corrective or compensatory system obligations.
2. *Semantics of System Obligations*: The semantics of system obligations, $Sys\ obliged_{sys}(b)$, on the other hand, are defined in terms of [*fact*]=*P*, as:

   $[Sys\ obliged_{sys}(b)] = G \Rightarrow \Diamond p(b)$

   Where G is a general predicate (e.g. on the system state) that enables the obligation. This meaning implies that if the action b is obliged to be executed by the system, then it will *eventually* be executed as expressed by the LTL operator $\Diamond$. In general, $\Diamond$ has no time limit, but this can be constrained by the

computational limits of the system or by the time limit of the DSA contract (i.e. its expiry date and time). The main aspect to note in the meaning of system obligations is that, unlike user obligations, system obligations can be enforced and their enforcement depends on the correct behaviour of the system components responsible for their fulfilment.

3. *Semantics of Permissions*: The semantics of permissions are also defined in terms of the special semantic function, [*fact*]=*P* as follows:

   $[User\ permitted(a)] = p(request) \Rightarrow \Diamond p(a)$

   This meaning simply says that to be permitted, as a user to execute an action, is the same as saying that when a request to execute that action occurs, this will eventually result in the execution of the action. Again, the $\Diamond$ operator leaves out any time constraints, which could be introduced using a real-time version of the operator or using next $\bigcirc$.

3.2 Mapping DSAs to SecPAL4DSA

Finally, we describe here an approach for mapping permission and obligation clauses in the language of [2] to assertions in SecPAL4DSA, such that DSAs can be enforced by a policy enforcement point.

As we mentioned earlier, one of the main features of SecPAL is that it expresses the root of authority in each individual assertion. This is equivalent to saying that each clause in a DSA must have a root of trust, which is more expressive than the normal DSA clauses. Therefore, we assume that for each clause in a DSA, a signatory, *A*, assumes the role of the root of trust for that clause. Based on this assumption, we define the transformation function, $F: C \rightarrow A$, as follows:

$F(\texttt{IF}\ G\ \texttt{THEN}\ P((p,an,d))) = A\ \texttt{says}\ p\ permitted((an,d))\ \texttt{if}\ G$
$F(\texttt{IF}\ G\ \texttt{THEN}\ O((p,an,d))) = A\ \texttt{says}\ p\ obliged((an,d))\ \texttt{if}\ G$

where *obliged* is either $obliged_{sys}$ or $obliged_{user}$ depending on whether *p* is a system or a user, respectively. The transformation function uses the structure of actions in the language of DSAs to construct the corresponding parts of SecPAL4DSA permission and obligation assertions.

4 Penalties in SecPAL4DSA

Another construct, which we use to extend the language of SecPAL4DSA is that of *penalties*, which are added to the definition of assertions to form what we call *penalty clauses,* defined by the following syntax:

Penalty Clause ::= (*A*, *Penalty*)

Where *Penalty*: *Principal* $\rightarrow$ N is a utility function mapping principals to some values (e.g. natural numbers representing a monetary concept such as money). Going back to *Example 1*, we can define a couple of penalty clauses as follows.

Example 2. Define the new penalty clauses based on the permission and obligation clauses as follows.

(**Clause1**) (*A* `says` *B permitted*(*access*, *data*) `if` *A hasFinished*(*authenticate*, *B*) `where` *c*, (*A* → 10))
(**Clause2**) (*A* `says` *B obliged*$_{user}$(*pay*, *amount*) `if` *B permitted*(*access*,*data*) `and` *B hasFinished*(*access*, *data*) `where` *currentTime* ≤ *PaymentDeadline*, (B → 50))

In Clause1, the failure of permitting the assertion to happen will incur a penalty on *A* of 10 units. This means that even though *A* was meant to authorise *B* to access the data, it failed to do so, which corresponds to a denial of service. Instead, in Clause2, the failure is related to *B* failing to make a payment by the specified deadline, which corresponds to a violation of an obligation. Therefore, it incurs a penalty of 50 units.

4.1 Semantics of Penalty Clauses

We define the semantics of penalty clauses more formally in terms of the semantic function, *E*[(*A*, *Penalty*)], defined as follows:

$E[(A, Penalty)] = Penalty \in Penalty_{Log} \{ p(a) \} Penalty \notin Penalty_{Log}$
where A=*A* `says` *B* $dsa_{operator}(a)$ `if` *fact* `where` *c*
and $dsa_{operator} \in \{obliged_{user}, obliged_{sys}, permitted\}$

The meaning of a penalty clause depends essentially on whether or not the action of the deontic operator in an assertion has taken place or not. If this is the case, then the penalty specified in the clause does not belong to the state called $Penalty_{Log}$, which registers all the due penalties. Otherwise, if the action did not take place, then the penalty is logged in the state.

4.2 Policy Queries

The additional extensions of the deontic predicates and penalties that SecPAL4DSA incorporated into the policy language allow for richer semantics for the reference monitor interpreting queries generated as a result of user requests. One such interesting high-level semantics would be to include a risk-sensitive reference monitor that compares the probability of the failure of an assertion (i.e. failure of granting access, failure of fulfilling obligations) with the penalty incurred by that failure. For example, we could define a risk-calculation function, *R*: *Penalty Clause*→ N, which returns the risk of the failure of an assertion part of a penalty clause compared to the penalty associated with that failure.

For example, taking the penalty clauses of *Example 2* and assuming that the probability of failure of the first assertion is 0.7 and for the second is 0.05, then *R*(Clause1) = ((0.7 × 10)/100) = 0.07 and *R*(Clause2) = ((0.05 × 50)/100) = 0.025. This demonstrates that the probability of failure combined with the penalty can give indicate how delicate an assertion (i.e. DSA clause) is.

5 Conclusion and Future Work

This paper presented an extension to a popular policy language called SecPAL, for modelling and expressing data sharing agreements among enterprises with different

administrative domains. The new language, SecPAL4DSA, is capable of encoding permission and obligation clauses of a DSA, and can also express penalty clauses and provide a quantitative means based on risk levels for evaluating policy rules against requests submitted by external users for accessing and using local resources.

In its current form, SecPAL does not allow prohibitive clauses to be modelled due to issues related to decidability of queries. For future work, we plan to investigate other methods by which prohibitions can be modelled in terms of the current language constructs. Also, we are planning to consider other quantitative factors related to DSAs, such as modelling of bounded obligations. Finally, we plan to develop a query evaluation engine for the new language, SecPAL4DSA and evaluate its performance with regards to case studies taken from the domain of scientific data sharing.

References

1. Sieber, J.E.: Data Sharing: Defining Problems and Seeking Solutions. Law and Human Behaviour 12(2), 199–206 (1988)
2. Arenas, A.E., Aziz, B., Bicarregui, J., Wilson, M.: An Event-B Approach to Data Sharing Agreements. In: Méry, D., Merz, S. (eds.) IFM 2010. LNCS, vol. 6396, pp. 28–42. Springer, Heidelberg (2010)
3. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The Ponder Policy Specification Language. In: Sloman, M., Lobo, J., Lupu, E.C. (eds.) POLICY 2001. LNCS, vol. 1995, pp. 18–38. Springer, Heidelberg (2001)
4. Becker, M.Y., Fournet, C., Gordon, A.D.: SecPAL: Design and Semantics of a Decentralized Authorization Language. Journal of Computer Security 18(4), 597–643 (2010)
5. Swarup, V., Seligman, L., Rosenthal, A.: A Data Sharing Agreement Framework. In: Bagchi, A., Atluri, V. (eds.) ICISS 2006. LNCS, vol. 4332, pp. 22–36. Springer, Heidelberg (2006)
6. Matteucci, I., Petrocchi, M., Sbodio, M.L.: CNL4DSA a Controlled Natural Language for Data Sharing Agreements. In: 25th Symposium on Applied Computing, Privacy on the Web Track. ACM, New York (2010)
7. Dillaway, B.: A unified approach to trust, delegation, and authorization in large-scale grids. Microsoft Corporation, Tech. Rep. (2006)
8. Becker, M.Y., Malkis, A., Bussard, L.: A Framework for Privacy Preferences and Data-Handling Policies. Microsoft Research, Tech. Rep. MSR-TR-2009-128 (September 2009)
9. Colombo, M., Martinelli, F., Matteucci, I., Petrocchi, M.: Context- Aware Analysis of Data Sharing Agreements. In: 4th European Workshop on Combining Context with Trust, Security, and Privacy, CAT 2010, pp. 99–104 (2010)