Provisions and Obligations in Policy Management and Security Applications

Claudio Bettini*, Sushil Jajodia†, X. Sean Wang†, Duminda Wijesekera†

* DSI, Università di Milano, Italy. bettini@dsi.unimi.it,
†Dept. of Info.& Software Eng., George Mason University, USA. {jajodia,xywang,dwijesek}@gmu.edu

Abstract

Policies are widely used in many different systems and applications. Recently, it has been recognized that a "yes/no" response to every scenario is just not enough for many modern systems and applications. Many policies require certain conditions to be satisfied and actions to be performed before or after a decision is made in accordance with the policy. To address this need, this paper introduces the notions of *provisions* and *obligations*. Provisions are those conditions that need to be satisfied or actions that must be performed before a decision is rendered, while obligations are those conditions or actions that must be fulfilled by either the users or the system after the decision. This paper formalizes a rule-based policy framework that includes provisions and obligations, and investigates a reasoning mechanism within this framework. A policy decision may be supported by more than one derivation, each associated with a potentially different set of provisions and obligations (called a global PO set). The reasoning mechanism can derive all the global PO sets for each specific policy decision, and facilitates the selection of the best one based on numerical weights assigned to provisions and obligations as well as on semantic relationships among them. The paper also shows the use of the proposed policy framework in a security application and discusses through an example various aspects of how the system may compensate unfulfilled obligations.

1 Introduction

Policies are widely used and play an important role in many different contexts. In a computer system, policies provide the basis for the design of its underlying control mechanisms. For example, an access control policy defines what information users are authorized to read or modify. Most research on policies, however, has been on traditional static policies where only "yes/no" decisions are made for user requests. For modern applications (e.g., business-to-business or business-to consumer applications), such traditional static policies are too inflexible to meet the complex requirements.

As an example, consider a loan application and management (payment collection etc.) system. It allows users to initiate a loan application process if they are already registered in the system. All users are given an opportunity to register with the system by supplying the necessary information, and if this step is successful, they have permission to proceed with the loan application process. Note that here the initiation of the loan application is not a statically assigned permission to users. Users are given the permission to apply for loan as long as they satisfy some conditions; if they don't satisfy these conditions, they may be able to perform certain actions to satisfy them.

Continuing with the example, assume a loan application is approved. In this case, the applicant will have the access to the funds under the condition that the user agrees to pay off the loan according to a certain payment schedule. Here again, such a condition is different from a statically assigned permission in the sense that the user promises to satisfy certain obligations in the future, and the system needs to be able to monitor such obligations and take appropriate actions if the obligations are not met.

From the example, we see that policies in many applications are complex, and a system requires flexible and powerful mechanisms to handle conditions and actions before and after certain decisions (access to the loan funds by the applicant in the example). Since the two sets of conditions and actions are conceptually different and require different management techniques, we distinguish between them by calling them *provisions* and *obligations*, respectively. Intuitively, provisions are specific actions to be performed be-

Proceedings of the 28th VLDB Conference,
Hong Kong, China, 2002

fore the decision is taken, and obligations are actions that will be taken in the future.

We base our formal policy model on logic rules as found in deductive databases. This is similar to static access control mechanism ([JS+01]). For example, a rule may specify that a user can access the funds of a loan only if the user is the owner of the loan. More complex rules may be possible such as the rule that a user can access an insurance policy if the user is the owner but the policy is not canceled, or the user is a manager of the insurance company.

In addition to logic rules, we attach two sets of predicates, namely provisions and obligations. We separate these two sets of predicates from the more static parts of the policy rules in order to manage them in special and specific manners. For example, the non-satisfaction of a provision does not necessarily lead to a denial of access; rather, it may prompt the user with additional actions. Similarly, an obligation is not a usual condition that is satisfied at the time of the access; rather, it may prompt the system to monitor certain promises because of this access.

At first glance, provision may appear to be similar to a precondition and obligation to a post condition. In designing and implementing software systems, a *precondition* of a method or function characterizes the conditions under which the method/function is expected to perform as specified [G81, LW94]. Hence, it is the responsibility of the user of the method/function to ensure that the method/function is called in an environment satisfying the precondition. Notice that the evaluation of a provision is only called for when the rule is being evaluated. Therefore, it does not constitute a precondition of the rule. This is quite different from a rule that says an access is granted provided some provisions are met. Secondly, the user of a method/function is able to determine the validity of the precondition prior to the call being made, whereas a provision satisfaction is called within the execution of an already called rule. Hence, they are different in spirit as well as in execution strategy.

Similarly, a method/function execution comes to a completion in an environment in which its stated *post-condition* is satisfied [G81, LW94]. If the method/function is called in an environment that satisfies its preconditions, then it is the responsibility of the method/function to ensure that the post-condition is being satisfied when the execution terminates normally. Notice that, as we have formulated, obligations are not the responsibility of the access control module, but that of the subject. Hence, they are not satisfiable upon the termination of a rule that grants an access. Notice that the obligation fulfillment, as stated in our loan example may take much longer time than being valid immediately after the granting of access privilege. In fact, at any moment of time, the reliability rating gives a measure of how much of the obligations have been met. Therefore, at best, generating the required set of obligations can be thought of as a post-condition of firing an access rule, but obligations themselves are not.

The focus of this paper is to reason about the policy rules in the presence of provisions and obligations. More specifically, the system must deduce what actions (if any) that a user may perform in order to gain access, and what promises (if any) that a user must make after gaining the access. It is possible that a number of different set of provisions and obligations lead to the same permission. Then we need a mechanism to allow the user or the system to choose the most appropriate ones. More specifically, our approach is the decomposition of the policy evaluation procedure in two steps; this decomposition enables the materialization of the model and of the specification of the alternative sets of provisions and obligations. This feature is particularly valuable in contexts, like access control, where the number of requests is much more frequent than the updates to the policy, and where efficiency is a concern.

Our reasoning framework is independent from the specific applications as long as they involve policies with provisions and obligations. The framework is also independent from the specific language used to define provision and obligation actions. However, in order to make our study more concrete and practical, we discuss a special application of our framework on security policies with provisions and obligations. Since obligations are promises from users to be fulfilled in the future, monitoring obligation is an essential part of the system that uses provisions and obligations in its policies. We refer the readers to [BJ+02] for more details.

The contribution of the paper is threefold. Firstly, we formalize the provisions and obligations in rule-based policy language, and provide a reasoning mechanism to derive relevant provisions and obligations for a particular policy decision. Secondly, we introduce a mechanism that allow the automatic choice among alternative provisions and obligations. Lastly, we give an example on how the framework can be used in a dynamic access control system.

The rest of the paper is organized as follows. In the next section, we formalize policy rules with provisions and obligations and mechanism to reason about them. In Section 3, we present the system considerations on the choice of provisions and obligations that are derived from the formal model of the previous section. In Section 4, we discuss a dynamic access control mechanism in our framework. We relate our work with other contributions in Section 5, and conclude the paper in Section 6 with summary remarks and future research directions.

2 Policy rule evaluation with provisions and obligations

In this section we describe our formal approach to rule evaluation with provisions and obligations. Initially, we assume that the policy can be represented by a set of rules and facts in the form of a (positive) datalog program, but with each rule and fact associated with its provisions and obligations. Then, we will discuss how the language to represent policy rules can be extended to a more expressive one, in particular including negation.

2.1 Representation of the policy rules

Recalling the standard notions, given finite sets of variables $\mathbf{V}$, constants $\mathbf{C}$, and predicate symbols $\mathbf{Q}$, an atom is a formula $Q(t_1, \ldots, t_n)$ where Q is a predicate symbol and each t_i is either a constant or a variable symbol. An atom is said to be ground if it is variable free. A *rule* is a formula written as $A \leftarrow B_1, \ldots, B_m$ where $A, B_1, \ldots, B_m$ are atoms; A is called the *head* and $B_1, \ldots, B_m$ the *body* of the rule. In clausal form a rule is expressed as $A \cup \neg B_1 \cup \ldots \cup \neg B_m$. A *fact* is a ground atom represented by a rule with an empty body and the atom itself as its head. As a safety requirement, we assume that each variable appearing in the head of a rule also appears in the body.

For now, we ignore provisions and obligations, and define a *policy* to be a set of policy rules.

As in the general case of logic programs, the semantics of the set of policy rules is characterized by its least Herbrand model. In the case of datalog rules we also know that this model is finite and that can be effectively derived by bottom-up evaluation. If R is the policy, we use M_R to denote its least Herbrand model. In the following we use the term ‘model’ to refer to the least Herbrand model unless stated otherwise.

Example 1 The following is an example of a set of rules assuming the set of variable symbols is $\{x, y, z\}$ and the set of constant symbols $\{a, b, c\}$.

$Q_1(x) \leftarrow Q_2(x, y), Q_3(y)$
$Q_2(a, b) \leftarrow$
$Q_3(b) \leftarrow$
$Q_1(y) \leftarrow Q_4(z, y, c)$
$Q_4(c, a, c) \leftarrow$

In this case, $\{Q_1(a), Q_2(a, b), Q_3(b), Q_4(c, a, c)\}$ is the model. Indeed, the last three ground atoms were given as facts, and the only intensional predicate is Q_1 which can be derived by two different rules, that when applied both give $Q_1(a)$. □

In the case of policy rules, a certain ground atom being in the model means that the specific decision represented by that atom is supported by the policy rules. Note that until now we have ignored provisions and obligations, and hence each rule can be applied unconditionally as long as atoms in the body are satisfied.

2.2 Representation of provisions and obligations

We represent provisions and obligations by two disjoint sets of predicate symbols $\mathbf{P}$ and $\mathbf{O}$ that are also disjoint from the set of predicate symbols $\mathbf{Q}$ allowed in the policy rule specification language. On the other hand, the set of variable and constant symbols $\mathbf{V}$ and $\mathbf{C}$ admitted in the predicates is the same used in the policy rules. The predicate symbols in $\mathbf{P}$ and $\mathbf{O}$ may be of any nonnegative arity.

An atom is either one of the symbols $\top$, or $\bot$, or a predicate $P_i(t_1, \ldots, t_k)$ with $P_i \in \mathbf{P}$ or $O_i(t_1, \ldots, t_k)$ with $O_i \in \mathbf{O}$ and each t_i is either a constant from $\mathbf{C}$ or a variable from $\mathbf{V}$. When not clear from the context we distinguish these atoms from those in the policy, by calling them PO-atoms. Then, a PO-formula is either a PO-atom, or a disjunction of PO-formulas, or a conjunction of PO-formulas. A PO-atom is ground if it is variable-free, and a PO-formula is *ground* if each of the atoms in the formula is ground. An interpretation I of a PO-formula is a mapping from each ground atom to the constant **True** or **False**, with the atoms $\top$ and $\bot$ mapping to the constants **True** and **False**, respectively. The satisfaction of a PO-formula is defined inductively on the structure of the formula, as usual, considering ground atoms as a basis, and the conjunction and disjunction operators that appear in the formula.

For each policy rule R_i in R there is an associated PO-formula, denoted by ϕ_{R_i}, representing the provisions and obligations for that rule. We also impose the intuitive constraint that each variable appearing in ϕ_{R_i} must appear in the body of R_i. Note that since these predicates are not part of the policy rule specification (the datalog program), they do not appear in its model M_R.

Example 2 The rules in Example 1 are labeled here with provision and obligation predicates. Note that an obligation, e.g., $O_1(s, x, y)$, may involve constant symbols (s in this case) that do not appear in the rule's body, and not necessarily they appear in any other rule. For example, a company may accept a deal imposing as obligation that the other party devolves part of the income to a charitable organization. This organization may be identified by the constant s in O_1 and does not appear in any other policy rules.

Rule		**PO-formula**
(R_1)	$Q_1(x) \leftarrow Q_2(x, y), Q_3(y)$	$O_1(s, x, y)$
(R_2)	$Q_2(a, b) \leftarrow$	$P_1(b)$
(R_3)	$Q_3(b) \leftarrow$	$\top$
(R_4)	$Q_1(y) \leftarrow Q_4(z, y, c)$	$P_2(y, a) \wedge P_3(a) \wedge O_2(y, c)$
(R_5)	$Q_4(c, a, c) \leftarrow$	$\top$

□

2.3 Global Provision and Obligation Set

In the following we assign to each ground atom Q in the model M_R, a PO-formula Ψ_Q, called its *global provision and obligation set* (GPOS). Intuitively, a GPOS for Q represents the alternative sets of provisions and obligations that must be satisfied to derive Q.

Example 3 Considering the fragment of policy specification in Example 2, the GPOS $\Psi(Q_1(a))$ for Q_1 is $(P_1(b) \wedge O_1(s, a, b)) \vee (P_2(a, a) \wedge P_3(a) \wedge O_2(a, c))$. The reader can easily see that if provision $P_1(b)$ is satisfied (the corresponding actions have been taken) and obligation $O_1(s, a, b)$ has been accepted, the GPOS is satisfied since the first conjunctive subformula is satisfied. In this case, we know that $Q_1(a)$ can be derived. Indeed, the first three rules can be fired under these provisions and obligations. Similarly, the satisfaction of the second conjunctive subformula enables the firing of the last two rules that also derive

$Q_1(a)$. Since each of the other atoms in the model were given as facts in the policy rules, its GPOS is simply given by the associated PO-formula. □

To formalize the GPOS concept we need the following preliminary definition.

Definition Given an interpretation I of a PO-formula, a rule R_i in R is said to be *I-enabled* if there exists a ground substitution σ for the variables in ϕ_{R_i}, such that ϕ_{R_i} under σ is satisfied by I. A rule is said to be *I-grounded* if it is obtained by an I-enabled rule by applying σ to the head and body of the rule.

Note that multiple I-grounded rules may exist for each rule in R and that they are not necessarily variable-free. A rule R_i with $\phi(R_i) = \top$ is considered I-grounded for any interpretation I. We can now formally define GPOS.

Definition Given a set R of policy rules with associated PO-formulas, and a ground atom $Q(\bar{a})$ in the model M_R, the *global provision and obligation set* (GPOS) of $Q(\bar{a})$ is the ground PO-formula Ψ_Q such that an interpretation I satisfies Ψ_Q if and only if $Q(\bar{a})$ is in the model of the set of all I-grounded rules from R.

Before showing how the GPOS of an atom can be computed, we should consider a semantics aspect which may allow the system to obtain equivalent but simpler PO-formulas. It is not unusual that in order to satisfy a provision we may be forced to satisfy other provisions. For example, in a certain site policy, in order to satisfy a provision which requires a user to have entered his credit card number, it is necessary for the user to have first registered with the site, and hence this second requirement, which may be a provision itself used in a different authorization rule, is implicitly satisfied by the first. We say that provision P_2 is *subsumed by* provision P_1, (denoted by $P_2 \preceq P_1$) when satisfying P_2 implies satisfying P_1. Note that if these are ground atoms this relation can only be explicitly stated by the administrator or the site designer. In other cases, if provisions are represented as complex formulas in a different logic where subsumption is decidable, a subsumption hierarchy may be automatically derived.

In the following we assume that any algorithm computing the GPOS of an atom uses the subsumption hierarchies to substitute with $\top$ any provision or obligation if there is another one in the same conjunctive formula that is subsumed by it.

2.3.1 A bottom-up model and GOPS computation

If we want to precompute the GPOS for all of the ground atoms, it is convenient to compute it together with the derivation of the model of the rules. However, standard algorithms for model computation have to be substantially modified for this purpose.

Assume $Q_1, \ldots, Q_n$ are the predicate symbols appearing in the program R. In the computation, we associate each Q_i with a set $\hat{Q}_i$ of pairs of the form $(\bar{a}_i, \psi)$ where $\bar{a}_i$ is a tuple of constants. Two such sets of pairs are said to be *equivalent* if for each pair $(\bar{a}, \psi)$ in one, there is a a pair $(\bar{a}', \psi')$ in the other such that $\bar{a} = \bar{a}'$ and ψ is logically equivalent to ψ', and vice versa.

To perform the above computation, we use the function **Eval**($R_j, q_{j1}, \ldots, q_{jm}$), where R_j is a rule in R, with atoms $Q_{j1}, \ldots, Q_{jm}$ appearing in its body, and for each $i = 1, \ldots m$, q_{ji} is a pair in the set $\hat{Q}_{ji}$. The function does the following: if using the ground atoms identified by $q_{j1}, \ldots, q_{jm}$, and rule R_j, we can derive a ground atom $Q_k(\bar{a})$, then the function returns the pair $(\bar{a}, \psi)$ where ψ is the conjunction of the PO-formulas in all q_{ji} and $\phi(R_j)$ with variables instantiated accordingly to the tuples in $q_{j1}, \ldots, q_{jm}$.

The algorithm to derive the model of the set of rules and all the GPOS is reported in Figure 1. Clearly, the algorithm efficiency can be greatly improved by a number of possible optimizations. However, for the sake of simplicity in the presentation, the algorithm does not include any of them.

```
INPUT: a set $R$ of policy rules with associated $PO$-formulas.
OUTPUT: the model of $R$ with each atom $Q$ associated with its global set of provisions $\Psi_Q$.
  1. For i=1 to n do $S_i := \emptyset$.
  2. repeat
       (a) For i=1 to n do $\hat{Q}_i := S_i$.
       (b) For each rule $R_j$ and combination of pairs $q_{j1}, \ldots, q_{jm}$ from $\hat{Q}_{j1}, \ldots, \hat{Q}_{jm}$, respectively,
           where $Q_{ji}$ appears in the body of $R_j$, do
           $S_k := \hat{Q}_k \cup (\bar{a}, \psi)$ where $Q_k$ is the head predicate
           in $R_j$ and $(\bar{a}, \psi)$=Eval($R_j, q_{j1}, \ldots, q_{jm}$).
     until $S_i$ is equivalent to $\hat{Q}_i$ for all $i$, $1 \leq i \leq n$.
  3. The model for $R$ is the set of all ground atoms $Q_i(\bar{a})$ such that the set $\hat{Q}_i$ contains at least a pair $(\bar{a}, \psi)$.
     The GPOS for atom $Q_i(\bar{a})$ is $\psi_1 \vee \ldots \vee \psi_k$, where $(\bar{a}, \psi_1), \ldots, (\bar{a}, \psi_k)$ are all the pairs in $\hat{Q}_i$.
```

Figure 1: The algorithm for model and GPOS computation

Example 4 Consider the fragment of policy specification in Example 2. In this case $n = 4$, since we have 4 predicates appearing in the rules, and all sets S_i with $i = 1, \ldots, 4$ are set to be empty in step 1. The sets $\bar{Q}_i$ get the same value at step 2.a in the first iteration, and hence no pair q_i is available for step 2.b. The rules are considered one by one, but only R_2, R_3, and R_5 lead to a nonempty result for the **Eval** function, since these are the only ones with a nonempty body. At the end of the first iteration, we have $S_1 = \emptyset$, $S_2 = \{(\langle b, a \rangle, P_1(b))\}$, $S_3 = \{(\langle b \rangle, \top)\}$, $S_4 = \{(\langle c, a, c \rangle, \top)\}$. Since the termination condition is not verified a second iteration is performed. The pairs contained in the S_i sets are now assigned to the $\bar{Q}_i$ sets. Let us consider in detail step 2.b when rule R_1 is considered. Q_2 and Q_3 are the predicates in the body of R_1, hence we can disregard q_1 and q_4. There is only one possible combination of q_2, and q_3, since each set $\hat{Q}_2$ and $\hat{Q}_3$ contains a single pair. The function **Eval**$(R_1, (\langle b, a\rangle, P_1(b)), (\langle b\rangle, \top))$ returns the pair $(\langle a\rangle, P_1(b) \wedge O_1(s, a, c))$ that is the new value for S_1. The iterations of step 2.b considering R_2, R_3, and R_5 derive the same pairs as the ones derived in the previous iteration of the repeat statement, while the function **Eval** during the application of R_4 returns the pair $(\langle a\rangle, P_2(a, a) \wedge P_3(a) \wedge O_2(a, c))$ that is inserted in S_1. The termination condition of the "repeat" loop is not satisfied yet and a third loop iteration is performed. In this iteration no new pair will be added, but the $\bar{Q}_1$ is assigned its final value from S_1. The loop termination condition is now true, and by step 3, the GPOS for $P_1(a)$ is $(P_1(b) \wedge O_1(s, a, b)) \vee (P_2(a, a) \wedge P_3(a) \wedge O_2(a, c))$, while the ones for $Q_2()$, $Q_3()$, and $Q_4()$ are the single formulas in the corresponding pairs. □

Theorem 1 *The procedure in Figure 1 terminates and it is correct.*

2.3.2 A top-down procedure to compute a GPOS

If the model M_R of the set of rules R has already been computed, the global provision set $\Psi_{Q(\bar{a})}$ of a ground atom $Q(\bar{a})$ in M_R can be derived by the procedure in Figure 2. Intuitively, the procedure uses a top-down strategy starting from $Q(\bar{a})$, and for each rule/fact used in the derivation, it collects the associated set of provisions and obligations. Disjunction is inserted when alternative 'ground' rules are applied.

INPUT: a set R of rules with associated PO-formulas, the model M_R, a ground atom $Q(\bar{a})$ in M_R.

OUTPUT: the GPOS $\Psi_{Q(\bar{a})}$.

1. For each rule R_Q and ground substitution σ for all of its variables, such that:
   a) its head predicate can be unified with Q through σ
   b) Each ground atom B^g in the resulting body is in M_R.

   do
   We construct a formula $\Psi(B^g_1) \wedge \ldots \wedge \Psi(B^g_m) \wedge [\phi(R_Q)]_\sigma$ where each B^g_i is a ground atom in the body with $\Psi_{B^g_i}$ computed recursively as described for $\Psi(Q)$, and $[\phi(R_Q)]_\sigma$ is the ground version (under σ) of the PO-formula associated with R_Q.
   If an atom B^g_i has already been considered in a previous recursion of the current derivation, let $\Psi_{B^g_i} = \bot$. (we detect a useless cyclic derivation).

2. Ψ_Q is given by the disjunction of the formulas obtained for each of the above qualifying rule and substitution.

Figure 2: Top-down derivation of a global provision set

Example 5 Consider the fragment of policy specification in Example 2, and suppose we want to derive the GPOS for $Q_1(a)$. Rule R_1 with substitution $\{x/a, y/b\}$ matches both conditions 1.a and 1.b., and no other substitution can be used since $Q_2(a, b)$ and $Q_3(b)$ are the only ground instances of Q_2 and Q_3, respectively, in the model. Then, $\Psi(Q_1(a)) = \Psi(Q_2(a, b)) \wedge \Psi(Q_3(a, b)) \wedge O_1(s, a, b)$. Applying recursively the procedure to derive $\Psi(Q_2(a, b))$ and $\Psi(Q_3(a, b))$, we find only one applicable rule for each of them with empty body. Hence we easily obtain $\Psi(Q_2(a, b)) = P_1(b)$ and $\Psi(Q_3(a, b)) = \top$, which are substituted in the above formula obtaining $\Psi(Q_1(a)) = P_1(b) \wedge O_1(s, a, b)$. We have a second iteration of step 1 since rule (R_4) also matches conditions 1.a and 1.b with the only possible substitution $\{z/c, y/a\}$. In this case $\Psi(Q_1(a)) = \Psi(Q_4(c, a, c)) \wedge P_2(a, a) \wedge P_3(a) \wedge O_2(a, c)$ from which $\Psi(Q_4(c, a, c))$ is dropped since it evaluates to $\top$ by applying R_5 with the empty substitution. By step 2 the required GPOS is given by the disjunction of the formulas derived in each iteration of step 1. Hence, $\Psi(Q_1(a)) = (P_1(b) \wedge O_1(s, a, b)) \vee (P_2(a, a) \wedge P_3(a) \wedge O_2(a, c))$. □

2.4 Extensions to the policy representation language

One of the most significant extensions to the expressiveness of the language is probably allowing negation of atoms in the body of a rule. For example, with this extension a policy may establish that a certain individual can access certain data if a certain other individual cannot access the same data based on the policy rules.

This extension can be achieved quite naturally by allowing the policy rules to be represented as a *stratified or locally-stratified datalog program with negation*, for which we know minimal fixed points of the evaluation can be found, one of which is identified as the intended model. We recall that rules are stratified if whenever there is a rule with predicate S as its head and a negated atom in the body with predicate Q, there is no path in the dependency graph[1] from S to Q. Local stratification [Prz88] essentially impose the same condition but considering ground instantiations of the rules. There are well known techniques to check and find a stratification [Ull88]. Note that rules should also be *safe*, i.e. all variables must be limited either by appearing in a positive predicate in the body or by being (even indirectly) equated to a constant or to a limited variable.

The introduction of this limited form of negation requires that we also extend the logic for PO-formulas introducing negation as an additional logic operator, as it will be clear from the following discussion. The algorithm to derive the GPOS must take into account the stratification. In the case of the algorithm illustrated in Figure 1, its steps must be applied for each stratum, starting from the lowest one. When considering stratum i, the derivable ground instances of all predicates appearing at lower strata and their

[1] The dependency graph has a node for each predicate and a directed edge from Q_1 to Q_2 is in the graph if predicate Q_1 appears (possibly negated) in the body of a rule and predicate Q_2 is in the head.

associated GPOS have already been computed. This means that if $\neg Q$ appears in the body of a rule at stratum i, since stratification guarantees that all rules deriving Q will be at a lower strata, all atoms $Q(\bar{a})$ that can be derived (with the associated GPOS) have already been identified. Hence, $\neg Q$ can be true (i.e., the rule can be applied) for all possible instantiations $Q(\bar{b})$ with $\bar{b} \neq \bar{a}$ without imposing any provisions, and also for $Q(\bar{a})$ but with associated provisions $\neg\Psi(\bar{a})$. The inclusion of this step and the repetition of the rule evaluation cycle for each stratum leads quite straightforwardly to the desired extension of the algorithm in Figure 1. We do not report here the details of the algorithm.

Other extensions to the policy rule specification language may be obtained considering rules expressed in datalog with order or temporal constraints. An example of security policy using this language can be found in [BBFS98].

3 The choice of rules: a system perspective

The GPOS of an atom indirectly represent the alternative sets of policy rules that can be used to derive the atom. Indeed, if we transform the global provision set (GPOS) of a certain ground atom in disjunctive normal form (DNF), by Definition 2.3, each conjunctive subformula represents one set of provisions and obligations that is sufficient to satisfy in order to derive the atom. The definition also says that the necessary rules are the I-grounded rules in the policy, where I is an interpretation satisfying the subformula.

In this section we investigate the problem of selecting a minimum set of provisions and obligations, and hence indirectly of rules, sufficient to derive what the user is asking for. We will see that the *minimality* criteria is not simply based on the number of provisions and obligations. We start with the following definition.

Definition A set of ground PO-atoms is called a *valid provision and obligation set* (VPOS) for a ground atom Q, if the conjunction of the atoms in the set logically implies the GPOS of Q.

The set of PO-atoms in each conjunctive subformula in the DNF representation of a GPOS is a VPOS accordingly to Definition 3. If one of the subformula is equivalent to $\top$ we have an empty VPOS. Satisfying all the provisions and obligations in a VPOS for atom Q makes it derivable accordingly to the policy. However, since the provisions and obligations must be typically satisfied by an external user performing certain actions, the system needs to carefully examine alternative VPOS, possibly identifying a *best* choice among them that will be proposed to the external user.

3.1 Simplifying a GPOS and comparing VPOS

Certainly, the best choice is an empty set of provisions and obligations, i.e. identifying a VPOS equal to $\top$, since this means that the atom can be derived without any provision nor obligation.

Hence, the first step performed by the system when evaluating a GPOS, is to verify if some of the provisions and obligations appearing in the formula are already satisfied. Note that for obligation predicates the satisfaction has simply the meaning of the obligation being accepted by the subject that is supposed to fulfill it. In practice this test involves a lookup in a data structure or in a database, similarly to what is done for *external* predicates. Any satisfied predicate is replaced by $\top$ in the formula.

When negation is allowed in the policy rules, as described in Subsection 2.4, negation may be applied to a PO-formula during GPOS generation. This leads to having negated ground atoms in the formula. Intuitively, a negated provision or obligation simply means that the system should check that it is not satisfied, i.e., the corresponding actions have not been taken. Hence, during the simplification of a GPOS, each negated atom $\neg P$ such that P evaluates to **True** is substituted with $\perp$ while if P evaluates to **False** is substituted with $\top$. The same for obligation predicates.

The process illustrated above may lead to identifying a VPOS equal to $\top$, that is certainly the best we can hope for. Otherwise, since all of the VPOS contain at least a predicate, we need to compare them.

In order to compare different VPOS, an ordering on provision and obligation predicates must be established. Considering the notion of subsumption described in Section 2, we already have a way to decide between two alternative provisions P_1 and P_2 if we know that P_2 is subsumed by P_1. Intuitively, P_1 will only involve a subset of the actions required to fulfill P_2, and hence it should be preferred as probably easier to fulfill. However, there are other semantic-based considerations which could make a provision (or set of provisions) preferable to another one. For example, the action of sending a confirmation email may be evaluated easier to satisfy than the registration at a site, while they may appear as alternative provisions for the derivation of a certain authorization in a security policy. The obligation of notifying the supplier to obtain more licenses each time we make new installations of its software is likely to be preferable than the one asking to buy separate copies of the software.

Our approach is to assign a numeric weight $W_P > 0$ to each provision predicate P (and similarly W_O for each obligation predicate O) with the intuitive meaning that lower weight provisions are preferable to higher weight ones.

Considering the relation with subsumption defined above, the assignment of weights should always satisfy the following condition:

$$\text{If } P_2 \text{ is subsumed by } P_1 \text{ then } W(P_1) < W(P_2)$$

Clearly, the same must hold for obligations. This rule guarantees that the weight system implicitly applies the preference criteria about subsumed provisions illustrated above.

In general, the weight system allows the system to assign a global weight to each alternative set of provisions.

More formally, for each VPOS the system computes its weight as the sum of the weights of the predicates appearing in it. Note that even if two atoms in a VPOS have the same predicate, and hence the same weight, both participate in the sum since they are different ground instances. This technique implicitly defines a partial order on different derivations of the same atom and allows the system to support a certain decision by asking the *minimum set of provisions and obligations* allowed by the policy.

Definition A *best provision and obligation set* (BPOS) for an atom Q is a VPOS having the minimum weight among all the VPOS for that atom.

If an empty VPOS exists, it will have 0 as global weight and will be the unique BPOS. In Figure 3 we report a concise description of the procedure to derive the BPOS. The function **Eval-atom** used in the procedure simply checks if a provision/obligation (represented by a ground atom) is currently satisfied in the system.

1. Transform GPOS in DNF. For each conjunctive subformula, the set of its atoms forms a VPOS.
2. **For** each VPOS from Step 1 **do**:
    (a) For each predicate P_1 in VPOS such that **Eval-atom**(P_1)=**true** replace P_1 with $\top$
    (b) If a VPOS is the empty set, it is the unique BPOS and the procedure terminates
3. **For** each VPOS returned by Step 2 **do**: $W(\text{VPOS}) = \sum_i W(P_i) + \sum_j W(O_j)$ where i and j are indexes over all provisions and obligations appearing in the VPOS
4. Any VPOS whose weight is equal to $Min_l(W(\text{VPOS}_l))$ where l is an index over all weighted VPOS, is a BPOS.

Figure 3: The procedure to find BPOS

Theorem 2 *Given a policy specification, a ground atom in its model, and an assignment of weights to provisions and obligations predicates, the procedure illustrated in Figure 3 derives all BPOS for that atom.*

Example 6 Consider the fragment of policy specification in Example 2. Suppose, based on a user request, we need to derive $Q_1(a)$. By computing the model of the policy rules we see that $Q_1(a)$ can be actually derived and its associated GPOS as computed in previous examples is $\Psi(Q_1(a)) = (P_1(b) \wedge O_1(s,a,b)) \vee (P_2(a,a) \wedge P_3(a) \wedge O_2(a,c))$. In this case the formula is already in DNF and we can identify VPOS$_1$=$P_1(b) \wedge O_1(s,a,b)$ and VPOS$_2$=$P_2(a,a) \wedge P_3(a) \wedge O_2(a,c)$. Note that in this example no subsumption relation holds between P_2,P_3, and O_2, as well as between P_1 and O_1, otherwise some of them would have been dropped during the GPOS derivation. In step 2, the procedure evaluates the atoms in each VPOS, and suppose that in this case only the provision $P_2(a,a)$ is already satisfied and hence substituted with $\top$. None of the VPOS is equal to $\top$, hence the procedure continues with step 3. Suppose now that O_1 is subsumed by O_2, i.e., fulfilling obligation O_1 implies fulfilling also O_2. Suppose also that the weights are assigned as follows: $W(P_1) = 1$, $W(P_2) = 2$, $W(P_3) = 2$ $W(O_1) = 3$, and $W(O_2) = 1$. Since P_2 can be ignored, we obtain $W(\text{VPOS}_1) = 4$ and $W(\text{VPOS}_2) = 3$. Hence, in this case there is a unique BPOS for $Q_1(a)$ equal to $P_3(a) \wedge O_2(a,c)$. □

While here we assume that a single weight is given for a provision or obligation predicate, the technique is easily extended to take into account different weights assigned to different ground instances of the same predicate.

3.2 BPOS extensions and system provisions

Different preference criteria can be used to refine the notion of BPOS considering the specificity of a particular domain and application. For example, the minimization of provisions may be given precedence with respect to that of obligations. In this case a BPOS is defined as a VPOS having the minimum weight of obligations among all the VPOS with the minimum weight of provisions. Consistently, in the procedure of Figure 3, the weight of a VPOS should be defined as a pair (W_P, W_O) over which the new minimization criteria can be easily applied in Step 4 of the same procedure. The same idea can be pushed forward considering different types of provisions, one of which deserves to be discussed.

Until now we have considered provisions mainly as actions that an external user should perform in order for a policy rule to be applicable. However, in many cases there are actions that the system itself should perform before considering a policy rule applicable: we call these *system provisions* and, syntactically, a disjoint set of predicates should be used to distinguish them from the others. These provisions should also have an associated weight. A reasonable strategy may consider these provisions of secondary importance with respect to the ones we will have to ask the user to satisfy. In this case a BPOS may be selected minimizing first user provisions, then obligations and, if still there is more than one candidate, system provisions. The procedure in Figure 3 can be easily extended to implement this strategy as briefly discussed above.

3.3 Materialization and other system issues

Upon a user request for a decision represented by a ground atom Q, the system will return the unique BPOS to the user if one exists (in the best case is the empty one), otherwise it may be instructed to randomly select a BPOS or to leave the choice to the user. These issues are typically application dependent. We should note however, that these choices can affect the possible optimizations applicable to the algorithms. For example, the weighting technique may be used to prune some of the derivations during the model construction, if we admit the possibility of ignoring some of the VPOS. This clearly rules out the possibility of giving all alternative choices to the user, that in some application contexts may be a useful feature.

A critical system consideration is the opportunity of precomputing the policy model and the GPOS formulas. While the general procedures we have described above do not assume precomputation we discuss here under which conditions this technique can be applied.

If we precompute the model and the policy rules include some external predicate which is 'state'-dependent we have a potential problem since the model will be computed at a specific time, and changes in the system state will not be reflected in the model.

A solution to this problem is considering each 'state'-dependent predicate appearing in a policy rule as a system provision, hence including it in the PO-formula associated with that rule. In this way all possible derivations involving the rule will be considered, and only at evaluation time (when computing BPOS) the satisfiability of the 'state'-dependent predicates will enable or disable the derivation according to the state of the system at the current time. Technically, this can be easily achieved by extending step 2.a in Figure 3: if the **Eval-atom** function for one of these predicates returns **False** the atom is replaced by $\perp$. A weight greater than zero is assigned to each predicate denoting a system provision.

The set of predicates that are considered 'state'-dependent is actually decided by the system administrator, depending on the specific application requirements. Indeed, some of the predicates that are semantically 'state'-dependent may be left in rules in order to speed-up the system at runtime if, for example, changes in their value occur very rarely compared with policy evaluation requests. The price to pay for this speedup is the re-materialization process to be done whenever one of their value changes.

Another interesting issue arises if we do not assume a single transaction for the interaction with the user when the system asks for provisions and obligations, the user performs the necessary actions, and finally the system satisfies its request. The problem is independent from materialization and it is illustrated by the following example: Suppose at time t the user makes a request for Q and the system replies asking the user to satisfy a provision and/or to accept an obligation, the user performs the required actions and makes a second request to the system at time t'. However, the system denies the derivation because a 'state'-dependent predicate in the body of a rule involved in the derivation is no more satisfied at t'. This system behavior is certainly undesirable from the user point of view. One way to alleviate the problem is to include in the system answer to the user, not only the required provisions and obligations, but also information about the current 'state', i.e., contextual information that if not changed will guarantee the sufficiency of satisfying the required provisions and obligations. For example, a site may accept a transaction in any business day, provided the user is registered. Here *being in a business day* is a 'state'-dependent condition. The not-yet-registered user that asks to perform a transaction will be answered to satisfy the provision and will be given the guarantee that if that is done and the request is made in a business day the transaction will be allowed. Technically, this simply requires including the'state'-dependent predicates from the BPOS in the answer to the user. Clearly, there are many other possible ways to address this problem, including that of offering a few alternatives to the user.

3.4 Obligations management

A user accepting a policy with obligations agrees to fulfill them. In order to ensure that agreed upon obligations are fulfilled, the system monitors *obligation fulfilling*, and in case of failure, takes necessary compensating actions. Such compensating action could range from decreasing the *trustworthiness* of the user, replacing *unfulfilled* obligations with (perhaps costlier) alternatives, and/or taking punitive actions such as informing relevant authorities of the defaulted or terminating the policy in-force. In order to replace obligations with more stringent ones, the user need to be informed of changes in contractual obligations. Similarly, fulfilling obligations as promised may result in a (positive) compensating action such as acknowledging payment of monthly fees and thanking the user, and perhaps upgrading her *trustworthiness*, referred to as the *reliability rating* similar to the *credit rating* used by lending institutions in the United States. This is a complex issue and please refer to [BJ+02] for a more detailed discussion.

4 Applications to security policies

In this section we apply the theory we have developed to a specific domain: security policies for access control ([WL93, JS+01]). The literature on this topic is quite extensive; different languages have been proposed to specify access authorizations and authorization rules in different contexts (relational models, object oriented models, multimedia systems, XML, etc..), as well as to deal with concepts like delegation, negative authorizations, role-based authorizations, conflict resolution, and many issues concerning the administration of the policy.

The specification of a security policy can be easily extended with our notion of provisions and obligations for most of the proposed languages, since authorizations are specified by special predicates, and rules can be represented as datalog rules (in some case with negation). In particular this extension can be applied to one of the most expressive languages recently proposed [JS+01] as well as to languages that allow to express time-dependent and periodic authorizations [BBFS98].

In the following we do not consider a specific formalism, but use a simplified syntax that will be sufficient to illustrate the extension to including provisions and obligations.

Access authorizations are represented as a ternary predicate `access(object, subject, access-mode)`, with the intuitive meaning of authorizing the `subject` to access the `object` in a certain `access-mode`. Typical access modes are `write`, `read`, `modify`.

Authorization rules allow authorizations to be derived based on other authorizations and/or on certain predicates. Typed variables can be used in rules.

Objects and subjects may be organized in hierarchies, and typed variables together with predicates which establish relations in the hierarchy can be used in authorizations to provide access to all objects in a certain class, and/or by subjects in a certain group.

Provisions and obligations can be associated with both single authorizations (that we can see as facts) and authorization rules, in the form of a PO-formula.

We recall that while predicate symbols in provisions and obligations cannot appear in the rules, constant symbols and variable symbols are shared, and in particular we have the constraint that each variable appearing in a provision/obligation predicate attached to a rule must appear in that rule's body.

4.1 An example of a security policy with provisions and obligations

In Table 1 we report a few of the conditional rules that may define a security policy for a b2b web site.

The rules have the following intuitive meaning:

- Rule R1 says that any user of the web site can read the contract proposals posted by the suppliers provided that he has registered at the web site.
- Rule R2 says that any manager of a supplier company can write a contract provided that he has registered at the web site.
- Rule R3 says that whoever has the authorization to write a contract has also the authorization to modify the contract_terms of that contract
- Rule R4 says any manager who can read a contract can also modify it (resulting in a counter-offer to the contract proposal) provided that he has registered at level 2 (i.e., he has provided a second level authentication for reserved operations), that he or the system notifies the supplier of the intention of proposing a counter-offer, and with the obligation of electronically signing the contract within 5 days if the changes are accepted by the counterpart.
- Rule R5 simply says that the authorization to modify an object implies the authorization of modifying all of its parts. In the case of the contracts this rule allows to derive the authorization to modify the contract terms if there exists an authorization to modify the contract.

Even in the context of access control authorizations, it is very likely that more than one derivation of the same authorization can be obtained using different sets of rules. While this may only have an impact on efficiency in traditional authorization systems, in the context we are considering, different sets of rules have associated different sets of provisions and obligations.

In the following we illustrate the application of the techniques described in the previous sections to this example.

Suppose a certain user asks for the permission to modify the terms of a contract proposal. This request is interpreted by the authorization system considering that the userid is `uid1`, that the user has the role of `manager` and the specific object he asks to modify is identified by the constant `contract1-terms` which is part of `contract1`. The system also has the information that `contract1`was originally issued by that user. Technically, the user is asking for the authorization

$$A1 = \texttt{access}(\texttt{contract1-terms}, \texttt{uid1}, modify)$$

that must be evaluated against the site security policy.

Table 2 reports the output of the procedure in Figure 1 applied to the fragment of the security policy reported in Table 1, limiting the constants to those required in this example. Each atom in the first column is included in the model and has a corresponding GPOS.

Note that there are two alternative sets of provisions and obligations for the desired authorization. Referring to the terminology used in the previous sections we have two VPOS associated with it. In this case, the first VPOS is associated with the application of rules R1, R4, and then R5, while the second with that of R2, and then R3. Intuitively, this is due to the fact that the policy allows both the issuer of the contract and a potential customer to modify a contract, even if requiring different conditions, provisions and obligations.

In the example all conditions are satisfied (i.e., the external predicates in(uid1, Manager) and issuer(uid1,contract) evaluate to true) but still the authorization has different VPOS.

According to what explained in Section 3.1 the system first tries to simplify the GPOS, checking if some of the involved predicates are already satisfied. For example, the system checks if user `uid1` is already registered. If that is the case, one of the VPOS (Register(uid1)) is equal to $\top$ and the authorization can be given without any provisions and obligations. Suppose now that the GPOS cannot be simplified, i.e., none of the involved provisions and obligationsis already satisfied. Then, the system must compare the two VPOS trying to identify the most convenient to satisfy for the user. In this example we can observe that the Register_at_level2(uid1) provision is subsumed by the Register(uid1) provision; formally, this fact is semantic information given by the site designers. By the condition imposed in Section 3.1 this subsumption implies that the weight associated with Register_at_level2(uid1) is higher than the one associated with Register(uid1), and since the weight for the provision Notify(uid1) and the obligation Sign_within_5days(uid1,contract1) are greater than zero, the second VPOS is selected as the best provision and obligation set (BPOS) and the user is simply asked to register at the site in order to modify the contract.

R1 = access(contract, s, read) ←
    PRV: Register(s)

R2 = access(contract, s, write) ← in(s, Manager),   issuer(s,contract)
    PRV: Register(s)

R3 = access(contract_terms, s, modify) ← access(contract, s, write),   partof(contract_terms, contract)

R4 = access(contract, s, modify) ← access(contract, s, read),   in(s, Manager),   issuer(s′,contract)
    PRV: Register_at_level2(s), Notify(s′)
    OBL: Sign_within_5days(s,contract)

R5 = access(o1, s, modify) ← access(o2, s, modify),   partof(o1,o2)

Table 1: A subset of rules in the site security policy

Atom	**GPOS**
access(contract1, uid1, read)	Register(uid1)
access(contract1, uid1, write)	Register(uid1)
access(contract1, uid1, modify)	(Register_at_level2(uid1) ∧ Notify(uid1) ∧ Sign_within_5days(uid1,contract1)) ∨ (Register(uid1))
access(contract1-terms, uid1, modify)	(Register_at_level2(uid1) ∧ Notify(uid1) ∧ Sign_within_5days(uid1,contract1)) ∨ (Register(uid1))

Table 2: Example output.

4.2 Domain related extensions

By applying the general theory developed in the previous sections of this paper to the domain of access control we identified two domain-dependent aspects which require a specific solution.

4.2.1 Provisions by different subjects

In a derivation path it is possible that a certain rule requires a provision to be satisfied by a subject different from the one requesting the access. For example, a rule may state that John can read a document if Ann can write that document, and there may be a conditional authorization for Ann to write that document provided she subscribe to a certain service. Since it is unlikely that the system asks to a third party to perform certain actions to allow a different user to access some data, these provisions are evaluated as conditions: either they are already satisfied or the rule cannot be applied.

4.2.2 Negative authorizations

When negative authorizations can be specified in the authorization language (as in [JS+01] and in several other proposals), the evaluation triggered by an access request is more involved.

The model of the access control policy rules may include negative authorizations like

$$access(\texttt{contract2-terms}, \texttt{uid}, -modify)$$

denying to uid the right to modify contract2-terms. This rule may have been given explicitly or derived by a chain of rule applications. Negative authorizations and rules deriving negative authorizations can also be subject to provisions, however these provisions typically are in the form of actions to be performed by the system and not by the subject involved in the authorization. For example, the rule

$$\text{RNEG} = \texttt{access}(\texttt{oid}, \texttt{uid}, -*)$$

$$\text{PRV: } notify(adm)$$

says that any operation on object oid by user uid should be denied, and an email notification should be sent to the administrator. A derivation of a negative authorization may also involve positive authorizations and rules, however all associated provisions and obligations except those to be performed by the system must be considered as conditions. Intuitively, we don't want to ask a user to perform certain actions in order to deny himself the access he requested, and, as explained above, we also don't want to ask other users to satisfy provisions for derivations originated by a different user's request. Hence, either the provisions and obligations associated with an authorization or rule are already satisfied or the authorization/rule cannot be used. Even in the presence of negative authorizations and rules, the criteria used to decide among different VPOS is the minimization of the sum of the weights associated with each one. Since, as explained above, there are only system provisions and obligations associated with a negative authorization in the resulting model, the minimization intuitively applies to system resources (memory space, latency time, computational power, etc.).

5 Related work

Several recent contributions in the area of policy specification languages, policies for network management, and security policies can be found in [Pol01]. Examples of languages for policy specification are the PONDER language [DD+01], and the Policy Description Language (PDL) [LBN99]. "Obligation policies" in PONDER are event triggered condition-action rules for policy based management of networks and distributed systems. Similarly, PDL policies use the event-condition-action rule paradigm of active databases to define a policy as a function that maps a series of events into a set of actions. It is not the goal of our paper to propose a full-fledged policy specification language like PONDER, on the contrary our contribution emphasizes the role of policy rule conditions that can be made true by actions of external agents. By separating the specification and the processing of provisions and obligations from the rest of the policy, we provide a mechanism for identifying the *most convenient* way of deriving a specific policy decision. This is missing from the reasoning mechanisms underlying these languages and we believe its integration may be an interesting direction to pursue.

There have been numerous papers on access control policies (see e.g., [WL93, BBFS98, JS+01]), modeling a number of aspects including role-based access control, delegation, time-dependent access control, multiple access control policies, etc. All these models, however, assume that the system either authorizes the access request or denies it. The concept of provisional authorizations has only been introduced very recently. [KH00] proposes an access control system for XML documents where optional provisional actions are included in the specification of each authorization. A provisional action is defined as a set of functions (log, verify, encrypt, transform, write, create, and delete) used for extending the semantics of the authorization policy. In this case the above actions are taken by the system as part of the XML document transformation triggered by a user request. This concept has been generalized in [JKS01] where a formal treatment of provisional authorizations and rules is proposed. Here a provisional authorization imposes that an access can be granted provided that the user (and/or the system) take certain actions prior to authorization of his request. The paper identifies a class of logics that can be used to specify provisional authorizations and elaborates on the notion of *weakest precondition* under which an access can be granted. Our notion of provision is analogous, but we separate the syntax and semantics of provisions and obligations from authorizations. We also conceptually distinguish obligations from provisions: in [KH00] they simply differ syntactically for the value of the attribute "timing" which is `before` for the actions that describe provisions and `after` for obligations.

We also provide a technique to represent semantic relations between provisions/obligations, that allows the system to automatically identify preferable derivations among a set of alternatives. The approach in [JKS01] only partially achieves this goal since a lattice over logical implication is used to represent the relationships among different sets of provisions; the lattice cannot capture the semantics hidden in the fixed interpretation of the atomic predicates. We may also say that our work complements the one in [JKS01] since their contribution is focused on the model theoretic semantics of the language while ours is focused on the algorithms and techniques needed to derive the "most convenient" authorizations.

The technique we use to compare different VPOS has some relationship with techniques assigning weights to rules in rule-based systems. In [AC+91], for example, the authors consider production rule systems with priorities associated with the rules. User-defined priorities are assumed to override default priorities. The paper proposes an algorithm for determining the order between two rules given a default total order and an overriding partial order over some of the rules. Priorities may be seen as the weights we assign to provisions and obligations, and their algorithm may be adapted to our policy rules if we consider provisions and obligations as the conditions in the rules. However, our technique allows for a decoupling of the evaluation process, in order to precompute the policy model and only leave at runtime the comparison of VPOS and the evaluation of state-dependent conditions. This becomes a significant advantage in critical applications involving security policies.

There has been work on using logic rules for controlling rights. Work by Kagal and Finin [KU+02, KFJ01] has Horn Claus rules for stating access rights and distributed trust. Their rules permit conditional delegations and access, where the conditions in the conditional must be satisfied in order to obtain access privileges. According to [KFJ01], they plan to add obligations to their framework. Our work differs from theirs in many major ways. Firstly, we do have obligations. Secondly, our provisions are different from conditions that appear in conditional access permissions in a critical way. That is, these conditional predicates are not evaluated by the rule base responsible for the evaluation of rule. It is the responsibility of an external agent to communicate the fulfillment of the provisions to the rule execution/evaluation engine. For example, registering with an auction house may not be handled by the access control module that is responsible for granting the authority to place bids. Moreover, the conditions that Kagal et al. [KFJ01] refer to are evaluated as they are, independent of whether or not the same accesses can be obtained by satisfying different conditions. In contrast, our provisioning framework looks for the best provisions and asks an external agent to check for their validity.

Another line of research dealing with access policies is trust management. For example, the work by Blaze et al. [BFL96, BFS98] deals with trust management issues in a decentralized environment. Although trust can be used in granting accesses to protected objects, neither provisions nor obligations deal with trust based accesses.

6 Conclusion

In this paper, we illustrated that policy management in a complex system calls for more dynamic mechanisms like provisions and obligations. In order to successfully use provisions and obligations in a system, we provided a reasoning mechanism based on which a system may render decisions, ask for actions (as provisions) and promises (as obligations) from the user or the system. We believe the policy language with provisions and obligations and the reasoning mechanism is general and applicable in various situations. As an example, we explored their use in a security application explaining how the proposed mechanism deals with domain specific issues like negative authorizations.

The paper gives rise to a number of interesting issues to be explored further. Some are mentioned in the related work section above. Another interesting direction is to investigate specification and reasoning about provisions and obligations themselves. We provided some examples, but a more rigorous treatment is needed if the provision and obligations have complex structures. Monitoring obligations may also become an involved process when quantitative temporal constraints are part of the obligation specification, as it is realistic, for example, in complex policies regulating b2b sites. In this case temporal constraint reasoning techniques, as described e.g., in [BWJ97], may be integrated in the proposed architecture.

References

[AC+91] R. Agrawal, R. Cochrane, B. G. Lindsay. On Maintaining Priorities in a Production Rule System. in proc of VLDB, pp. 479–487, 1991.

[BBFS98] E. Bertino, C. Bettini, E. Ferrari, P. Samarati. An Access Control Model Supporting Periodicity Constraints and Temporal Reasoning. *ACM TODS*, 23(3):231–285, 1998.

[BFL96] M. Balze, J. Feigenbaum and J. Lacy. Decentralized Trust Management. IEEE Pro. 17th Symp. on Security and Privacy, 1996.

[BFS98] M. Balze, J. Feigenbaum and M. Staauss. Compliance Checking in the PolicyMaker Trust Management System. Proc. Financial Crypto'98, 1998.

[BJ+02] C. Bettini, S. Jajodia, X. Sean Wang, and D. Wijesekera. Obligation Monitoring in Policy Management. To appear in *IEEE 3rd Intern. Workshop on Policies for Distributed Systems and Networks*, June 2002.

[BWJ97] C. Bettini, X. Wang, S. Jajodia. Satisfiability of Quantitative Temporal Constraints with Multiple Granularities. In Proc. CP 1997.

[DD+01] N. Damianou, N. Dulay, E. Lupu, M. Sloman The Ponder Policy Specification Language in [Pol01], 2001.

[G81] D. Gries. The Science of Programming. Springer-Verlag, 1981.

[KH00] M. Kudo and S. Hada. XML document security based on provisional authorization. In proc. of the 7th ACM conference on Computer and communications security, pp. 87–96, 2000.

[JKS01] Provisional Authorizations. S. Jajodia, M. Kudo, V.S. Subrahmanian. In *E-Commerce Security and Privacy,* Anup Gosh (Ed.), pp. 133–159, Kluwer Academic Press, 2001.

[JS+01] S. Jajodia, P. Samarati, M.L. Sapino, and V. S. Subrahmanian. Flexible Support for Multiple Access Control Policies. *ACM Transactions on Database Systems,* 26(2):214–260, 2001.

[KFJ01] L. Kagal, T. Finin and A. Joshi. Trust-Based Security in Pervasive Computing Environments. in IEEE Computer, December 2001.

[KU+02] L. Kagal, J. Undercoffer, F. Perich, A. Joshi and T. Finin. A Security Architecture for Pervasive Computing Systems. Grace Hopper Celebration of Women in Computing 2002.

[LW94] B. H. Liskov and J. M. Wing. A Behavioral Notion of Subtyping. ACM Transactions on Programming Languages and Syetems, pages 1811-1841, November, 1994.

[LBN99] J. Lobo, R. Bhatia, and S. Naqvi. A Policy Description Language. In proc. of National Conference of the American Association for Artificial Intelligence, Orlando, FL, USA, 1999.

[Prz88] T. Przymusinski. On the declarative semantics of deductive databases and logic programs. In J. Minker, editor, *Foundations of deductive databases*, pages 193–216. Morgan Kaufmann, San Mateo, 1988.

[Pol01] Proceedings. Policies for Distributed Systems and Networks, International Workshop, POLICY 2001 Bristol, UK, Proceedings, M. Sloman, J. Lobo, E. Lupu (Eds.), Springer LNCS N.1995, 2001.

[Ull88] J. D. Ullman. *Database and Knowledge-base systems*. Computer Science Press, 1988.

[WL93] T.Y. C. Woo, S. S. Lam. Authorizations in distributed systems: A new approach. *Journal of Computer Security,* 2(2,3):107–136, 1993.

Appendix

Proof of Theorem 1

Termination is guaranteed by the fact that we only have a finite number of predicates and constants. Indeed, there are n sets $\bar{Q}_i$ whose number of elements can only grow by the steps in the algorithm, and since we have a finite number of ground atoms, each set can have only a finite number of pairs with different tuples. Moreover, the number of pairs with the same tuple is finite since the number of equivalent PO-formulas is finite, and hence only a finite number of them can be associated with each tuple.

Even if the algorithm uses a slightly different approach with respect to the standard datalog naive evaluation algorithm [Ull88], it is easily seen that it is equivalent. Indeed, it applies all possible ground rules starting from facts, and ending when the fixed-point is reached.

We have to show that the computed PO-formula for each atom in the model is a GPOS for that atom. Hence, we show that, given an atom $Q(\bar{a})$ in the model, the formula $\Psi(Q(\bar{a}))$ derived by the algorithm and an interpretation I for that formula, I satisfies $\Psi(Q(\bar{a}))$ iff the model of all I-grounded rules includes $Q(\bar{a})$. We proceed by induction on the number of steps in the main recursive loop of the algorithm, i.e., on the depth of the derivations. Consider first the case of a policy with only facts where $Q(\bar{a})$ is one of them. Then, there is a rule ($Q(\bar{a}) \ \leftarrow$) that we call r with an associated PO-formula $\phi(r)$ which is considered in the first iteration of the algorithm, deriving the pair $(\bar{a}, \phi(r))$. Since there cannot be other rules deriving $Q(\bar{a})$, the algorithm derives $\Psi(Q(\bar{a})) = \phi(r)$. Then, if I satisfies $\Psi(Q(\bar{a}))$, r is among the I-grounded rules and their model must include $Q(\bar{a})$. In the other direction, if the model of all I-grounded rules includes $Q(\bar{a})$, it means there is a rule among these rules that has $Q(\bar{a})$ as its head. Since by hypothesis r is the only such a rule, I satisfies $\phi(r)$ and hence also $\Psi(Q(\bar{a}))$. Suppose the theorem holds for all derivations of depth i and we prove it for depth $i + 1$. First assume I satisfies $\Psi(Q(\bar{a}))$. Then, it is easily seen that I satisfies ψ where $(\bar{a}, \psi)$ is derived by `Eval` in step 2b of the algorithm. We are interested in the case the derivation occurs at depth $i + 1$. Suppose r is the rule and $q_1, \ldots, q_m$ the pairs given as arguments to `Eval`. Then, if q_j is $(\bar{b}_j, \psi_j)$, $\psi = \psi_1 \wedge \ldots \wedge \psi_m \wedge \phi(r)$. Hence, I satisfies ψ_j for $j = 1 \ldots m$ and $\phi(r)$. Then, by induction, each atom $Q_j(\bar{b}_j)$ in the body of r is in the model of the set of I-grounded rules. But since I satisfies $\phi(r)$, also r with variables instantiated accordingly to $q_1, \ldots, q_m$ is among the I-grounded rules. Then $Q(\bar{a})$ must be in that model too. In the other direction, assume the model of all I-grounded rules includes $Q(\bar{a})$. Then, since this is the least Herbrand model[2] there exists at least a I-grounded rule r' with that atom as head and with all the atoms $Q_1(\bar{b_1}), \ldots, Q_m(\bar{b_m})$ in the body that are also in the model. By induction hypothesis, I satisfies their associated PO-formulas $\psi_1, \ldots, \psi_m$. Furthermore, since I satisfies $\phi(r')$, I also satisfies $\psi = \psi_1 \wedge \ldots \wedge \psi_m \wedge \phi(r)$, which is the formula returned by `Eval` together with the tuple $\bar{a}$ when evaluating the rule. By step 3, we know that this formula will appear, possibly in disjunction with other formulas in $\Psi(Q(\bar{a}))$. Then I satisfies $\Psi(Q(\bar{a}))$.

Proof of Theorem 2

We first show that all the VPOS that are candidate to be BPOS are considered in Step 2. Let Q be the ground atom, and $\Psi(Q)$ its GPOS. Consider an arbitrary VPOS V which is not considered in Step 2. By definition V logically implies $\Psi(Q)$. Hence, if $V_1, \ldots, V_m$ are the VPOS derived by step 1, there exists $1 \leq k \leq m$ such that $V \supset V_k$. Then, by monotonicity of addition over positive numbers, we derive $W(V) > W(V_k)$ from which we conclude V cannot be a BPOS. To conclude the proof we just observe that the min function in Step 4 guarantees that the returned VPOS are indeed BPOS.

[2] This fact is essential to rule out the case in which the set of I-grounded rules with head $Q(\bar{a})$ contains only a rule like $Q(\bar{a}) \leftarrow Q(\bar{a})$ or other 'cyclic' rules.