Characterising Access Control Conflicts *

João Moura
CENTRIA - Centre for Artificial Intelligence
Universidade Nova de Lisboa, Portugal

Abstract

The emergence of technologies such as service-oriented architectures and cloud computing has allowed us to perform business services more efficiently and effectively. Access control is an important mechanism for achieving security requirements in such information systems, which are described by means of access control policies (ACPs).

However, these security requirements cannot be guaranteed when conflicts occur in these ACPs. Furthermore, the design and management of access control policies is often error-prone due not only to the lack of a logical and formal foundation but also to the lack of automated conflict detection and resolution.

We use meta-model $\mathcal{M}^P$ to describe ACPs as logic programs, identify their basic conflict types and characterise them in terms of Default Logic and Strong Equivalence of logic programs. This characterisation allows for the automatic identification of such conflicts among other reasoning tasks.

Introduction

We begin by clarifying the term $policy$ in the context of access control which is somewhat ambiguous in the literature. Next we describe different aspects of access control policies that are of general interest to us and serve as motivation for our approach. Still in this Section we describe the state of the art relevant for us in the area of logic based access control, particularly meta-models for access control, keeping in mind the way it reflects in answer set programming (Gelfond and Lifschitz 1988) in particular. We also present an overview of the relevant work in the context of Logic Programs as well as, towards the logical characterisation of conflicts in access control, an introduction to strong equivalence in the context of logic programs due to (Lifschitz, Pearce, and Valverde 2000) and a relativised version of this notion due to (Eiter, Fink, and Woltran 2007). Before we also introduce equilibrium logic and the logic of here and there because they are necessary to the definition of strong equivalence.

*The work of João Moura was supported by the grant SFRH/BD/69006/2010 from Fundação para a Ciência e Tecnologia (FCT) from the Portuguese MEC - Ministério do Ensino e da Ciência. He would also like to thank Carlos Damásio for his important contribution as well as the anonymous reviewers.

In the following section we start by identifying and characterising different access control conflicts. We then discuss the interplay of exceptions in access control with default negation in logic programming and open the path for introducing strong negation in future work.

After this, there is a section where we discuss conflict resolution methods and end with conclusions and future work.

Motivation The need for characterizing conflict factors is not only due to the non-monotonic nature of access control in ASP but also to the fact that some policies are distributed and as such they can derive conflicting conclusions. The characterizations we present have the potential to improve the use of ASP in an access control mechanism by adding another level of policy assurance.

We present next an example of an access control policy where a user is represented by its credentials, each serving different purposes and each with different attributes. The example contains only one user and one credential for space reasons. Each of the aforementioned attributes are to be trusted for an entity. Subscriptions are available for different resources. There are rules for deciding whether a user is authenticated, whether a credential is valid or to check if the policy derives access to a resource for a given purpose.

Example 1 presents a very simple access control policy in the form of a positive logic program (apart from the choice loop that could, in the context of this positive program, be replaced by $selectCred(X) \vee nselectCred(X) \leftarrow credential(X).$).

Several limitations arise from directly implementing ACPs as positive logic programs. One comes from the fact that it is important to have a meta-model to describe and possibly interchange this policy. Others from the facts that more expressive power is needed and features such as default knowledge and exceptions are desirable and can be included, thus making it non-monotonic. Negative authorisation is also described in the literature as being a necessary feature. Other features such as separation-of-duty and the existence of purposes are also desirable.

Example 1 *The following is an example of an access control policy in ASP* [1]*:*

[1] Example 1 was taken and then adapted from http://asptut.gibbi.com/ (Eiter et al. June 2006)

```
allow(download,Resource) :-
     public(Resource).
allow(download,Resource) :-
     authenticated(User),
     hasSubscription(User,Subscription),
     availableFor(Resource,Subscription).

authenticated(User) :-
     valid(Credential),
     attr(Credential,name,User).

valid(Credential) :-
     selectCred(Credential),
     attr(Credential,type,T),
     attr(Credential,issuer,CA),
     trustedFor(CA,T).

hasSubscription("Joao",law_basic).
hasSubscription("Joao",computer_basic).

availableFor("nmr12.pdf",computer_basic).

trustedFor("New University",id).
trustedFor("PT Government",ssn).

% resources r(at_1,...,at_n);
credential(cr01).
attr(cr01,type,id).
attr(cr01,name,"Joao").
attr(cr01,issuer,"New University").


% Choice loop used to decide
% if a credential is used or not
selectCred(X) :- credential(X),
         not nselectCred(X).
nselectCred(X) :- credential(X),
         not selectCred(X).
```

This program has two answer sets, from which we filter predicates $selectCred$, $nselectCred$, $valid$, $authenticated$ and $allow$, namely:

Answer Set 1:

```
{ selectCred(cr01), valid(cr01),
  authenticated("Joao"),
  allow(download,"nmr12.pdf") }
```

and Answer Set 2:

```
{ nselectCred(cr01) }
```

Access Control Policies

For a long time now, logic programming and rule-based reasoning have been proposed as a strong basis for policy specification languages. However, the term policy has never been given a unique meaning. In fact, it is used in the literature in an ambiguous and broad sense that encompasses at least the following types:

Access Control Policies are policies that pose constraints on the behaviour of a system. They are typically used to control permissions of users/groups while accessing resources and services.

Trust Management these policy languages are used to collect user properties in open environments, where the set of potential users spans over the entire web and by definition is a priori partially unknown.

Action Languages are used in the specification of reactive policies to execute actions like event logging, notifications, etc. Authorisations that involve actions and side effects are sometimes called provisional.

Action languages typically are sorted into two classes: action description languages and action query languages. Examples of the former include STRIPS, PDDL, Language A (a generalisation of STRIPS), Language B (an extension of A) and Language C (which adds indirect effects also, and does not assume that every fluent is automatically "inertial"). There are also the Action Query Languages P, Q and R. There are conversions of these to ASP particularly, action language C.

Business Rules are statements about how a business is done. These are used to formalise and automate business decisions as well as for efficiency reasons. They can be formulated as reaction rules, derivation rules, and integrity constraints.

In the next sections, we present an overview of the existing approaches for Access Control Policies which is the only type we consider in this paper. Henceforth, we will use the term Policy as being an Access Control Policy.

In (Kolovski 2007; Bonatti et al. 2009), the reader can find good introductory surveys to logic-based ACPs. (Kolovski 2007) is limited to the presentation of a DL-based formalism to represent XACML policies, which is not formally characterized, while (Bonatti et al. 2009) considers a more general overview, including XACML.

Hierarchies, Inheritance and Exceptions For a long time now, computer security models have supported some forms of abstraction regarding the authorisation elements, to formulate security policies concisely. For example, users can be organised in groups. The authorisations granted to a user group is applicable to all of its member users, and authorisations concerning a class of objects apply to all of its member objects. This is typically modelled via an authorisation hierarchy derived from the hierarchies of subjects, resources and operations (basic hierarchies).

The authorisation hierarchy can be exploited to formulate policies in a incremental and top-down fashion. Starting with an initial set of general authorisations that can be progressively refined with more specific authorisations that in turn introduce exceptions to the general rules. A benefit that come together with this is that policies may be expressed concisely and allow easy management. Exceptions make inheritance a defeasible inference in the sense that inherited authorisations can be retracted (or overridden) as exceptions are introduced. As a consequence, the underlying logic must be non-monotonic.

Exceptions require richer authorisations. It must be possible to say explicitly whether a given permission is granted

or denied. Then authorisations are typically extended with some form of sign for granted permissions and some form of negation for denials. It may easily happen that two conflicting authorisations are inherited from two incomparable authorisations, therefore a policy specification language featuring inheritance and exceptions must necessarily deal with conflicts. A popular conflict resolution method – called denial takes precedence – consists of overriding the positive authorisation with the negative one (i.e. in case of conflicts, authorisation is denied), but this is not the only possible approach. In (Al-Kahtani and Sandhu 2004) the analysis includes user authorisation, conflict among rules, conflict resolution polices, the impact of negative authorisation on role hierarchies and an enforcement architecture.

Recent proposals have worked towards languages and models that are able to express, in a single framework, different inheritance mechanisms and conflict resolution policies. Logic-based approaches, so far, are the most flexible and expressive.

The $\mathcal{M}^P$ model

Over the years, research in access control has proposed a number of different models and languages in which terms authorisation policies can be defined. Despite the variety of proposed access control models described in the literature, most of the existing access control models are based on a small number of primitive notions.

In (Barker 2010) the authors describe the interpretation, syntax and semantics that are adopted in their proposed access control meta-model $\mathcal{M}^P$, which attempts to identify the aforementioned small number of primitive notions and that we will use throughout this paper.

In order to define $\mathcal{M}^P$, a prior version called meta-model $\mathcal{M}$ (Barker 2009) is extended to accommodate data subjects, data controllers, denials of access, the notion of purpose, contextual accessibility criteria and the flexible specification of permitted recipients of a data subject's personal data. For that, the following core (interpreted) relations of the $\mathcal{M}^P$ model (defined with respect to their many-sorted language) are used:

- PCA, a 4-ary relation, $K_{ds} \times K_{du} \times C \times P$.
- ARCA, a 5-ary relation, $K_{ds} \times A \times R \times C \times P$.
- ARCD, a 5-ary relation, $K_{ds} \times A \times R \times C \times P$.
- PAR, a 3-ary relation, $K_{du} \times A \times R$.
- PRM, a 3-ary relation, $K_{ds} \times R \times M$.

The semantics of the n-ary tuples in PCA, $ARCA$, $ARCD$, PAR, and PRM are, respectively, defined thus:

- $(k_{ds}, k_{du}, c, p) \in PCA$ if-and-only-if a data user $k_{du} \in K_{du}$ is assigned to the category $c \in C$ for the purpose $p \in P$ according to the data subject $k_{ds} \in K_{ds}$.
- $(k_{ds}, a, r, c, p) \in ARCA$ if-and-only-if the permission (a, r) is assigned to the category $c \in C$ for the purpose $p \in P$ according to the data subject $k_{ds} \in K_{ds}$.
- $(k_{ds}, a, r, c, p) \in ARCD$ if-and-only-if the permission (a, r) is denied to the category $c \in C$ for the purpose $p \in P$ according to the data subject $k_{ds} \in K_{ds}$.
- $(k_{du}, a, r) \in PAR$ if-and-only-if a data user $k_{du} \in K_{du}$ is authorised to perform the action $a \in A$ on the resource $r \in R$.
- $(k_{ds}, r, m) \in PRM$ if-and-only-if the data subject $k_{ds} \in K_{ds}$ controls access to the resource $r \in R$ and k_{ds} asserts that the meta-policy $m \in M$ applies to access on the resource r.

For representing hierarchies of categories, the following definition is included as part of the axiomatization of $\mathcal{M}^P$ (where '_' denotes an anonymous variable):

$$contains(C, C) \leftarrow dc(C, \_),$$
$$contains(C, C) \leftarrow dc(\_, C),$$
$$contains(C', C'') \leftarrow dc(C', C''),$$
$$contains(C', C'') \leftarrow dc(C', C'''); contains(C''', C'').$$

Authorisation may then be defined in $\mathcal{M}^P$ terms as:

$$par(K_{du}, A, R) \leftarrow prm(K_{ds}, R, c), \\ pca(K_{ds}, K_{du}, C', P), \\ contains(C, C'), arca(K_{ds}, A, R, C, P).$$

In this instance, a closed policy is specified as being enforced by all data subjects, and contains is a definition of a partial ordering of categories that are elements in the transitive-reflexive closure of a directly contains (dc) relation on pairs of category identifiers $dc(c_i, c_j)$, such that: $\Pi \models dc(c_i, c_j)$ iff the category $c_i \in C$ ($c_i \neq c_j$) is senior to the category $c_j \in C$ in a category hierarchy defined in Π and there is no category $c_k \in C$ such that $[dc(c_i, c_k) \wedge dc(c_k, c_j)]$ holds where $c_k \neq c_i$ and $c_k \neq c_j$. Although the partial ordering of categories is often a feature of access control models, it should be clear that other relationships between categories may be easily defined within the $\mathcal{M}^P$ model.

We point out that this meta-model is formally well defined and is essentially based on the use of just five key interpreted relations (the pra, pca, $arca$, $arcd$ and prm relations) and two proper axioms that define par and $contains$.

For further details we refer the reader again to (Barker 2009; 2010).

Example 2 *The following is the translation of Example 1 to $\mathcal{M}^P$:*

```
arca(public,download,Resource,all,any):-
    prm(public,Resource,MetaPolicy ).

arca(public,download,Resource,all,
        SubscriptionType):-
    pca(ds,User,authenticated,_),
    pca(ds,User,hasSubscription,
       SubscriptionType),
    par(SubsriptionType,availableFor,
         Resource).

pca(DS,User,authenticated,_) :-
     pca(DS,DU,Credential,valid),
     prm(ds,attr(cr01,name,User),
       metapolicy1).
```

```
pca(DS,DU,Credential,valid) :-
    par(DU,selectCred,Credential),
    prm(DS, attr(CR,type,Type),MP),
    prm(DS, attr(CR,issuer, Issuer),MP),
    par(Type,trustedFor,Organization).

par("Joao",hasSubscription,
    law_basic).
par("Joao",hasSubscription,
    computer_basic).

par(computer_basic,availableFor,
    "nmr12.pdf").

par(id,trustedFor,"New University").
par(ssn,trustedFor,"PT Government").

par(du,credential,cr01).

prm(ds,attr(cr01,type,id),metapolicy1).
prm(ds,attr(cr01,name,"Joao"),
     metapolicy1).
prm(ds,attr(cr01,issuer,
     "New University"),metapolicy1).

%Credential selection
par(DU,selectCred,Credential) V
par(DU,notselectCred,Credential):-
      par(DU,credential,Credential).
```

Equilibrium logic and the logic of here-and-there

We recall the basic concepts of equilibrium logic, an approach to non-monotonic reasoning developed by (Pearce 1997) as a generalisation of the answer-set semantics for logic programs. We give only the most relevant aspects here. For more details, the reader is referred to (Pearce 1997; 2006; Pearce, de Guzman, and Valverde 2000a; 2000b; Lifschitz, Pearce, and Valverde 2000).

Equilibrium logic is based on the non-classical logic of here-and-there, which is denoted by HT. The language of HT is given by the class of propositional formulas as described above, and the axioms and rules of inference of HT are those of intuitionistic logic together with the axiom schema

$$(\neg\varphi \supset \psi) \supset (((\psi \supset \varphi) \supset \psi) \supset \psi)$$

which characterises the three-valued here-and-there logic of Heyting and Gödel (for this reason, HT is sometimes also known as Gödel's three-valued logic). The standard version of equilibrium logic has two kinds of negation, intuitionistic negation, $\neg$, and strong negation, $\sim$. The authors also show how strong negation can be added but for reasons that we explain later as well as for simplicity, here we present a restricted version containing only the first type of negation.

The model theory of HT is based on the usual Kripke semantics for intuitionistic logic, which is given in terms of Kripke frames of form $\langle W, \leq\rangle$, where W is a set of points, or worlds, and $\leq$ is a partial-ordering on W, except that Kripke frames for HT are restricted to those containing exactly two worlds, say H (here) and T (there), with H $\leq$ T. As in ordinary Kripke semantics for intuitionistic logic, we can imagine that in each world a set of atoms is verified and that, once verified here, an atom remains verified there.

In view of the restricted nature of Kripke frames for HT, it is convenient to define the semantics of HT in terms of HT-interpretations, which are ordered pairs of form $\langle I_H, I_T\rangle$, where I_H and I_T are sets of variables such that $I_H \subseteq I_T$. For an HT-interpretation $I = \langle I_H, I_T\rangle$, a world $w \in \{H, T\}$, and a formula φ, the truth value, $v_I(w, \varphi) \in \{0, 1\}$, of φ in w under I is given as follows:

1. if $\varphi = \top$, then $v_I(w, \varphi) = 1$;
2. if $\varphi = \bot$, then $v_l(w, \varphi) = 0$;
3. if $\varphi = p$, for some variable p, then $v_l(w, \varphi) = 1$ if $p \in I_w$, and $v_l(w, \varphi) = 0$ otherwise;
4. if $\varphi = \neg\psi$, then $v_l(w, \varphi) = 1$ if, for every world u such that $w \leq u, v_I(u, \psi) = 0$, and $v_I(w, \varphi) = 0$ otherwise;
5. if $\varphi = (\varphi_1 \wedge \varphi_2)$, then $v_I(w, \varphi) = min(\{v_I(w, \varphi_1), v_I(w, \varphi_2)\})$;
6. if $\varphi = (\varphi_1 \vee \varphi_2)$, then $v_I(w, \varphi) = max(\{v_I(w, \varphi_1), v_I(w, \varphi_2)\})$; and
7. if $\varphi = (\varphi_1 \supset \varphi_2)$, then $v_l(w, \varphi) = 1$ if, for every world u such that $w \leq u$, $v_I(u, \varphi_1) \leq v_I(u, \varphi_2)$, and $v_I(w, \varphi) = 0$ otherwise.

We say that φ is true under I in w if $v_I(w, \varphi) = 1$, otherwise φ is false under I in w. An HT-interpretation $I = \langle I_H, I_T\rangle$ satisfies φ, or I is an HT-model of φ, if-and-only-if $v_I(H, \varphi) = 1$. If φ possesses some HT-interpretation satisfying it, then φ is said to be HT-satisfiable, and if every HT-interpretation satisfies φ, then φ is HT-valid. An HT-interpretation is an HT-model of a set T of formulas if-and-only-if it is an HT-model of all elements of T. Finally, an HT-interpretation $\langle I_H, I_T\rangle$ is said to be total if $I_H = I_T$, and non-total otherwise (i.e., if $I_H \subset I_T$). It is easily seen that any HT-valid formula is valid in classical logic, but the converse does not always hold. For instance, $p \vee \neg p$ and $\neg\neg p \supset p$ are valid in classical logic but not in the logic of here-and-there, because $I = \langle\emptyset, \{p\}\rangle$ is not an HT-model for either of these formulas.

We say that two theories are equivalent in the logic of here-and-there, or HT equivalent, if-and-only-if they possess the same HT-models. Two formulas, φ and ψ, are HT equivalent if-and-only-if the theories $\{\varphi\}$ and $\{\psi\}$ are HT-equivalent.

Equilibrium logic is characterised in terms of a particular minimal-model construction in HT. Formally, an equilibrium model of a theory T is a total HT- interpretation $\langle I, I\rangle$ such that

(i) $\langle I, I\rangle$ is an HT-model of T, and

(ii) for every proper subset J of I, $\langle J, I\rangle$ is not an HT-model of T. $\langle I, I\rangle$ is an equilibrium model of a formula φ if-and-only-if $\langle I, I\rangle$ is an equilibrium model of $\{\varphi\}$.

A formula φ is a brave consequence of a theory T, symbolically $T \mid\sim_b \varphi$, if-and-only-if some equilibrium model of T satisfies φ. Dually, φ is a skeptical consequence of T ,

symbolically $T \sim_s \varphi$, if-and-only-if all equilibrium models of T satisfy φ. The basic reasoning tasks in the context of equilibrium logic are the following decision problems:

- Decide whether a given theory T possesses some equilibrium model.
- Given a theory T and a formula φ, decide whether $T \sim_b \varphi$ holds.
- Given a theory T and a formula φ, decide whether $T \sim_s \varphi$ holds.

The first task is called the consistency problem; the second and third tasks are respectively called brave reasoning and sceptical reasoning. For the time being, we only need to fully consider the first but we will use the other two notions further ahead in the document.

The following two propositions are straightforward and will also be useful later on:

Proposition 1 *For any HT-interpretation $I = \langle I_H, I_T \rangle$ and any propositional formula φ, the following relations hold:*

1. $v_I(T, \varphi) = 1$ *if-and-only-if* $v_{l_T}(\varphi) = 1$*;*
2. $v_I(H, \varphi) = 1$ *implies* $v_I(T, \varphi) = 1$*; and*
3. $v_I(H, \varphi) = 1$ *if-and-only-if* $v_{l_H}(\varphi) = 1$*, if φ is an expression (i.e., a formula without $\supset$) that does not contain negation.*

Notice that the first part of this proposition states that φ is true under $I = \langle I_H, I_T \rangle$ in the world T if-and-only-if φ is true under I_T in classical logic. The second part is a direct consequence of the notion of an HT-interpretation, namely, in view of the condition $I_H \subseteq I_T$, which holds for each HT-interpretation $\langle I_H, I_T \rangle$. The third part states that formulas without negations and implications can be evaluated by pure classical means, i.e., in both worlds separately.

Proposition 2 *A total HT-interpretation $\langle I, I \rangle$ is an HT-model of φ if-and-only-if I is a model of φ in classical logic.*

Strong Equivalence of Logic Programs

Towards the logical characterisation of conflicts in access control, we present first an introduction to the here-and-there logic and then to strong equivalence in the context of logic programs due to (Lifschitz, Pearce, and Valverde 2000).

Strong Equivalence Theorem A program π is unary if, in every rule of π, the head is an atom and the body is either $\top$ or an atom. In the statement of the theorem, formulas and rules are identified in the sense of nested logic programs(Lifschitz, Tang, and Turner 1999) without strong negation and with propositional formulas. Accordingly, programs become a special case of theories, and we can talk about the equivalence of programs in the logic of here-and-there.

Theorem 1 *For any programs π_1 and π_2, the following conditions are equivalent:*

(a) *for every program π, programs $\pi_1 \sqcup \pi$ and $\pi_2 \sqcup \pi$ have the same answer sets,*

(b) *for every unary program π, programs $\pi_1 \sqcup \pi$ and $\pi_2 \sqcup \pi$ have the same answer sets,*

(c) *π_1 is equivalent to π_2 in the logic of here-and-there.*

The fact that (b) implies (a) shows that the strong equivalence condition we are interested in (for every π, $\pi_1 \sqcup \pi$ is equivalent to $\pi_2 \sqcup \pi$) does not depend very much on what kind of program π is assumed to be: it does not matter whether π is required to belong to the narrow class of unary programs or is allowed to be an arbitrary program with nested expressions. The fact that (a) is equivalent to (c) expresses the correspondence between the strong equivalence of logic programs and the equivalence of formulas in the logic of here-and-there.

Relativised Notions of Strong and Uniform Equivalence In what follows, we revise the notions of relativised strong equivalence (RSE) and relativised uniform equivalence (RUE) due to (Eiter, Fink, and Woltran 2007).

Definition 1 *Let P and Q be programs and let A be a set of atoms. Then,*

(i) *P and Q are strongly equivalent relative to A, denoted $P \equiv_s^A Q$, if-and-only-if $P \cup R \equiv Q \cup R$, for all programs R over A;*

(ii) *P and Q are uniformly equivalent relative to A, denoted $P \equiv_u^A Q$, if-and-only-if $P \cup F \equiv Q \cup F$, for all (non-disjunctive) facts $F \subseteq A$.*

Observe that the range of applicability of these notions covers ordinary equivalence (by setting $A = \emptyset$) of two programs P , Q, and general strong (resp. uniform) equivalence (whenever $Atm(P \cup Q) \subseteq A$). Also the following relation holds: For any set A of atoms, let $A' = A \cap Atm(P \cup Q)$. Then, $P \equiv_e^A Q$ holds, if-and-only-if $P \equiv^{A'} Q$ holds, for $e \in \{s, u\}$.

They show that RSE shares an important property with general strong equivalence: In particular, they state that it appears that for strong equivalence, only the addition of unary rules is crucial. That is, by constraining the rules in the set in Definition 1 to unary rules does not lead to a different concept.

Conflict types in Access Control and their Characterisation

Prohibition is essential to achieve the security requirements of modern information systems. However, defining prohibition in access control model will give rise to conflicts. A policy conflict occurs when the objectives of two or more policies cannot be simultaneously met. (Wang et al. 2010) have summarised three types of policy conflicts in their model, defined as modality, redundancy and potential conflicts.

Modality Conflict

Modality conflicts are inconsistencies in the policy specification which may arise when two or more policies with opposite modalities refer to the same authorisation subjects, actions and objects.

Simply put, a modality conflict occurs when there are both allow and deny decisions with the same authorisation subject, action and objects.

Definition 2 *Let π be an access control policy. We say that there is a modality conflict if:*

$$\pi \models_b Allow(X) \wedge Deny(X)$$

In ASP and $\mathcal{M}^P$ terms this means having both arca and arcd predicates in the same model. Due to the introduction of an arcd predicate to represent $\neg$ arca, it is possible to have an equilibrium model (an answer set) containing both arca and arcd predicates refering to the same authorisation subjects, actions and objects.

Redundancy Conflict

It is known that assigning priorities to access control policies can solve modality conflicts. However, this method can lead to the emergence of policies that never apply, which can be called redundant policies. Even though a redundancy conflict has no influence in the enforcement of the access control policies, it should be identified and dealt with because a redundant policy often reflects a mistake that was made while describing security requirements.

Definition 3 (Redundancy in $\mathcal{M}^P$) *Let π be an access control policy and Π a set of access control policies. Assuming that the priority of Π is higher than the priority of π (i.e. $\Pi \prec \pi$), we have a redundancy conflict in $\mathcal{M}^P$ terms when, for π and Π, $arca$ literals are derived:*

$$\pi \models_c arca(d_s, a, r, c, p) \text{ and } \Pi \models_c arca(d_s, a, r, c, p)$$

or $arcd$ literals are derived:

$$\pi \models_c arcd(d_s, a, r, c, p) \text{ and } \Pi \models_c arcd(d_s, a, r, c, p)$$

Thus, an access control policy π is a redundant policy if a permission or a prohibition, having the same authorisation subject, authorisation action, and authorisation object as π, is always ($\models_c$ denotes cautious consequence)[2] derived from the set of access control policies with higher priority than the priority of π[3].

Theorem 2 (Redundancy in terms of RSE) *Let Π be a set of access control policies and π be a policy such that $\Pi \prec \pi$. A redundancy conflict occurs when:*

$$\pi \cup \Pi \equiv^A_{s(c)} \Pi$$

Thus, if $\pi \cup \Pi$ is strongly equivalent (over A) to Π, where A is the language of $\mathcal{M}^P$.

The characterisation of redundancy in terms of strong equivalence has been noted in the past e.g. in (Eiter et al. 2004).

Definition 4 (Partial Redundancy in $\mathcal{M}^P$) *Let π be an access control policy and Π a set of access control policies. Assuming that the priority of Π is higher than the priority of π (i.e. $\Pi \prec \pi$), we have a partial redundancy conflict in $\mathcal{M}^P$ terms when for π and Π, $arca$ literals are derived:*

$$\pi \models_b arca(d_s, a, r, c, p) \text{ and } \Pi \models_b arca(d_s, a, r, c, p)$$

or $arcd$ literals are derived:

$$\pi \models_b arcd(d_s, a, r, c, p) \text{ and } \Pi \models_b arcd(d_s, a, r, c, p)$$

Thus, an access control policy π is a partially redundant policy if a permission or a prohibition, having the same authorisation subject, authorisation action, and authorisation object as π, can always be derived ($\models_b$ denotes brave reasoning) from the set of access control policies with higher priority than the priority of π.

Theorem 3 (Partial Redundancy in terms of cautious RSE) *Let Π be a set of access control policies and π be a policy such that $\Pi \prec \pi$. A redundancy conflict occurs when:*

$$\pi \cup \Pi \equiv^A_{s(c)} \Pi$$

Thus, if $\pi \cup \Pi$ is strongly equivalent (over A) to Π, where A is the language of $\mathcal{M}^P$.

Note that a partial redundancy occurs when there $always$ exist some answer sets (over a set A) that are the same for $\pi \cup P$ and $\Pi \cup P$ for any program P, but not all of them are always the same i.e.

$$\exists a \in AS(\pi \cup P), b \in AS(\Pi \cup P) \; s.t. \; (a \cap b \neq \emptyset) \wedge (a \neq b)$$

Potential Conflict

Notice that the above two types of conflict are inconsistencies related to the authorisation subject, action and object. There is another type of conflict between two policies having overlaps in their condition expression. It is the case that there is no modality conflict and redundancy conflict between the two polices, but when their associated conditions are simultaneously satisfied, the two policies result in a modality conflict or redundant conflict.

According to the definition, when some policies have the same condition literals, where a condition is a conjunctive formula $P_1 \wedge P_2 \wedge \ldots \wedge P_n$ and each P_i represents a generic well-formed formula. We can infer the existence of potential conflicts among these policies. Consequently, potential conflicts are highly pervasive in access control systems.

Definition 5 *A potential conflict occurs between two policies π_i and π_j in $\mathcal{M}^P$ terms if:*

1. *π_i derives a permission (in the form of an $arca$ literal) and π_j derives a prohibition (in the form of an $arcd$ literal) and,*
2. *There are overlaps such as: $condition(\pi_j) \cap condition(\pi_i) \neq \emptyset$, and*
3. *There is no policy π_k in the policy set such that: $condition(\pi_i) \wedge condition(\pi_j) \rightarrow condition(\pi_k)$ and π_k derives a prohibition when $priority(\pi_i) \prec priority(\pi_j) \prec priority(\pi_k)$ or π_k derives a permission when $priority(\pi_j) \prec priority(\pi_i) \prec priority(\pi_k)$.*

Theorem 4 *Let π be a policy, there is a potential conflict if:*

$$\pi \cup \{\bot \leftarrow allow(x) \wedge deny(x)\} \equiv_s \{\bot \leftarrow \top\}$$

and the policy is safe if:

$$\pi \cup \{\bot \leftarrow allow(x) \wedge deny(x)\} \equiv_s \pi$$

[2] We consider here the well-known concepts of brave and cautious (sceptical) consequence.

[3] Note that the meta-model $\mathcal{M}^P$ does not allow directly the specification of order between policies but this can be easily done in ASP.

Potential conflicts have also been analysed in the extension of Lobo's PDL with ordered disjunction by (Bertino 2005), where logic programming with ordered disjunction was used as a way to prioritize action execution in case conflicting actions were triggered by the policy.

Default Negation as a Cause of Conflicts

Throughout this section we will present examples of programs which we start by formulating as sets of default rules and then present their translation into ASP and their underlying characterisation. Knowledge is represented in default logic (Reiter 1987) by a default theory $\langle D, W\rangle$ consisting of a set of defaults D and a set of formulas W. Each default rule like:

$$\frac{A : B_1, \ldots, B_n}{C}$$

where A is a prerequisite, $B_1 \ldots B_n$ are justification and C is a conclusion, is represented in LP as:

$$C \leftarrow A, not\ \neg B_1, \ldots, not\ \neg B_n.$$

There is work in the literature about intuitionistic interpretations of default logic (Cabalar and Lorenzo 2004). In (Woo and Lam 1992), default rules are used to provide semantics to closed and open policy bases for the case where a policy is represented as a 4-tuple $A = (P^+, P^-, N^*, N^-)$ in which can be fitted explicit approvals and denials as well as undetermined decisions.

Incoherences Some contradictions in a default theory cause the non-existence of extension. We call such contradictions incoherences. A default theory is incoherent if it has no default extension, otherwise it is coherent.

Incoherences may be categorised into the different sorts which are described in the following examples. We also show the way they can be reflected as access control conflicts translated to incoherent ASP programs and the way to capture them with the $\mathcal{M}^P$ meta-model due to (Barker 2010):

Definition 6 (Default rule with exception.) *$T = \langle D, W\rangle$, where $D = \{\frac{:B}{\neg A}\}$ and $W = \{A\}$. In T incoherences occur between W and the consequents of applicable defaults.*

Example 3 *The following is an example of an ASP program reflecting this conflict:*

$$\begin{aligned} P1 = \quad & \{allow\_analysis(pedro, code). \\ & candidate(pedro). \\ & \neg allow\_analysis(X, code) : - \\ & \qquad not\ employee(X).\} \end{aligned}$$

Where the following is its translation to $\mathcal{M}^P$:

$$\begin{aligned} & arca(ds, read, code, pedro, analysis). \\ & pca(ds, pedro, candidate, analysis). \\ & arcd(ds, read, code, pedro, analysis) \leftarrow \\ & \qquad not\ pca(ds, pedro, employee, analysis). \end{aligned}$$

Considering the $\mathcal{M}^P$ meta-model, Example 3 presents a contradiction because its only answer set contains $arca(ds, read, code, pedro, analysis)$ and $arcd(ds, read, code, pedro, analysis)$.Mp

Definition 7 (Omission of mandatory choice) *$T = \langle D, W\rangle$, where $D = \{\frac{:B}{C}, \frac{:D}{\neg C}\}$ and $W = \emptyset$. In T, incoherences occur in the consequents of applicable defaults.*

Example 4 *The following is an example of an ASP program reflecting this conflict:*

$$\begin{aligned} P2 = \{ \quad & allow\_entrance(X, premises) : - \\ & \qquad not\ \neg employee(X). \\ & \neg allow\_entrance(X, premises) : - \\ & \qquad not\ \neg candidate(X).\} \end{aligned}$$

Where the following is its translation to $\mathcal{M}^P$:

$$\begin{aligned} & arca(ds, enter, premises, X, \_) : - \\ & \qquad not\ \neg pca(ds, X, employee, \_). \\ & arcd(ds, enter, premises, X, \_) : - \\ & \qquad not\ \neg pca(ds, X, candidate, \_). \end{aligned}$$

Considering the $\mathcal{M}^P$ meta-model, Example 4 presents a contradiction because its only answer set contains arca and arcd predicates with the same arguments.

Definition 8 (Need for Action Rules) *$T = \langle D, W\rangle$, where $D = \{\frac{:\neg B}{A}\}$ and $W = \{A \rightarrow B\}$. In T incoherences occur between the justifications of used defaults and the consequences of (W and the consequents of used defaults).*

Example 5 *The following is an example of an ASP program reflecting this conflict:*

$$\begin{aligned} P3 = \{ \quad & candidate(X) : - not\ employee(X). \\ & employee(X) : - candidate(X).\} \end{aligned}$$

Where the following is its translation to $\mathcal{M}^P$:

$$\begin{aligned} & pca(ds, X, candidate, \_) : - \\ & \qquad not\ pca(ds, X, employee, \_). \\ & pca(ds, X, employee, \_) : - \\ & \qquad pca(ds, X, candidate, \_). \end{aligned}$$

Considering the $\mathcal{M}^P$ meta-model, Example 5 presents an incoherence because it has no answer sets. This is typically solved with the introduction of action languages such as the ones described in the introduction.

Incoherence can also be defined (in the sense of (Eiter, Fink, and Moura 2010)) in terms of odd negative loops such as the well known:

Definition 9 (Depth 1 negative loop) *$T = \langle D, W\rangle$, where $D = \{\frac{:\neg A}{A}\}$ and $W = \{\}$. In T incoherences occur because there is a loop between the justifications of used defaults and the consequents of used defaults).*

Example 6 *The following is an example of an incoherent ASP program reflecting this conflict:*

$$\begin{aligned} P4 = \{ \quad & employee(pedro). \\ & allow\_entrance(X) : - employee(X), \\ & \qquad not\ allow\_entrance(X).\ \} \end{aligned}$$

Where the following is its $\mathcal{M}^P$ description:

$$\begin{aligned} & pca(ds, pedro, employee, \_). \\ & arca(ds, enter, premises, X, \_) : - \\ & \qquad pca(ds, X, employee, \_), \\ & not\ arca(ds, enter, premises, X, \_). \end{aligned}$$

Considering the $\mathcal{M}^P$ meta-model, Example 6 presents an incoherence because it has no answer sets.

Classical Strong Negation as a Cause of Modality Conflicts Though we have not yet discussed strong negation in this paper, mostly because $\mathcal{M}^P$ presents a way of dealing with this through $arcd$ predicates, for contextualisation and motivation for future work, we still sketch what are some of its implications in access control.

Introducing strong negation ($\neg$) may lead to modality conflicts in the state of a single user with respect to a single role. The conflict is due to simultaneous positive and negative authorisations. The following are variations of the conflict:

- Case 1: Conflict among unrelated rules: an atom is derived by an applicable rule and its negation is derived by another applicable rule.
- Case 2: Conflict among related rules: an atom a is derived by an applicable rule r_1. That rule r_1 implies another rule r_2 to be applicable, which in turn derives the negation of the atom $\neg a$.

Conflict resolution methods

Access control policies are expressed by means of rules which enforce derivation of not only authorisations, access control and integrity constraint checking but also conflict resolution. To resolve rule conflicts, there must be a method for unambiguously choosing a decision. Most conflict resolution methods in practice choose one of the rules in conflict to take precedence over the others. (Other methods are possible, however, such as majority rules – choosing the decision of the majority of the rules in conflict).

We list several possible conflict resolution methods below. Note that it is sufficient to define them in terms of their behaviour when exactly two rules are in conflict, because the access control system can handle cases of more than two rules in conflict by following a simple algorithm that does paired matches of each Allow rule against each Deny rule. This algorithm issues an Allow decision if any Allow rule wins its matches against every Deny rule, and otherwise issues a Deny decision. Some of the possible conflict resolution methods for choosing rules to take precedence are:

- Specificity precedence: A rule that applies to a more specific entity takes precedence over a rule that applies to a more general entity.
- Deny precedence: Deny rules take precedence over Allow rules.
- Order precedence: Rules are totally ordered, so it is possible to explicitly state which rules take precedence over others.
- Recency precedence: Rules specified more recently in time take precedence over others. Note that recency precedence is equivalent to order precedence where order is determined by the time at which each rule was set.

These conflict resolution methods may be used in combination. It is possible to use different conflict resolution methods depending on whether conflicting rules differ in the principals they cover, the resources they cover, or both. For example deny precedence if conflicting rules differ in principals, but specificity precedence if conflicting rules differ in resources or in both resources and principals. It may also be necessary to resort to multiple conflict resolution methods when one method fails to resolve a conflict. For example, when conflicting rules cover groups, but those groups are peers of each other, specificity precedence cannot resolve the conflict.

It has been shown in the literature how to use prioritised logic programming to solve authorisation conflicts e.g. (Bai 2007), where the authors assign each rule a name representing its preference ordering, using a fixed point semantics to delete those less preferred rules, then using ASP to evaluate the authorisation domain to get the preferred authorisations.

In (Ahn et al. 2010) different combining algorithms have been identified: Permit-overrides, Deny-Overrides, First-Applicable, and Only-One-Applicable. as well as the way they can be implemented in ASP.

Two-dimensional conflicts Besides violations of direct manipulation in the presence of rule conflicts there is also the question of behaviour in the presence of a two-dimensional conflict. For example, two dimensional conflict occurs when two rules are in conflict and one rule is more specific in the principal dimension (e.g. in the role hierarchy) while the other is more specific in the resource dimension.

Conclusions and Future Work

We identified different types of basic conflicts that occur in access control programs and characterise them in terms of the notion of Relativised Strong Equivalence of logic programs. We also identify conflicts that occur when we introduce default negation and characterise them in terms of default logic while using meta-model $\mathcal{M}^P$ as well as answer set programming throughout that section give examples of those conflicts.

These characterisations enables the detection of conflicts to be done automatically by using automatic theorem provers such as (Zinn and Intelligenz) and most importantly the ones identified in (Cabalar and Lorenzo 2004) where it is stated that the relation they established between S4F and the logic of Here-and-There, allows using modal S4F provers for proving theorems in that intermediate logic. Because of the characterisation of Strongly Equivalent programs as programs that are equivalent in the logic of HT, we can use these theorem provers to perform reasoning and automatically identify the conflicts we characterised before in terms of Strong Equivalence and Relativised Strong Equivalence.

Overall, these characterizations are flexible enough to be extended to several types of conflicts, and can be used to detect which types of conflict are generated, as well as trace them back to the source (potentially identifying leaks in ACP).

Future Work Introducing strong negation ($\neg$) may lead to modality conflicts, even if this matter has been thoroughly

studied and partially solved in the literature through the introduction of paraconsistent semantics and by dealing with it syntactically.

We still need to investigate the possibility of having a characterisation in the logic of HT or in terms of (Relativised) Strong Equivalence for the conflict types that we identified as occurring with the introduction of default negation.

Research must be done next on conflict resolution methods, formally defining rule combining algorithms in $\mathcal{M}^P$. We also plan to study the implication of using paracoherent semantics such as Semi-Equilibrium models which was presented in (Eiter, Fink, and Moura 2010). It is necessary also to investigate the usage of Action Languages to solve problems that arise from the introduction of default negation.

References

Ahn, G.-J.; Hu, H.; Lee, J.; and Meng, Y. 2010. Representing and reasoning about web access control policies. In Ahamed, S. I.; Bae, D.-H.; Cha, S. D.; Chang, C. K.; Subramanyan, R.; Wong, E.; and Yang, H.-I., eds., *COMPSAC*, 137–146. IEEE Computer Society.

Al-Kahtani, M. A., and Sandhu, R. 2004. Rule-based rbac with negative authorization. In *Proceedings of the 20th Annual Computer Security Applications Conference*, ACSAC '04, 405–415. Washington, DC, USA: IEEE Computer Society.

Bai, Y. 2007. Logic program for authorizations. *World Academy of Science, Engineering and Technology issue 33*.

Barker, S. 2009. The next 700 access control models or a unifying meta-model? In *Proceedings of the 14th ACM symposium on Access control models and technologies*, SACMAT '09, 187–196. New York, NY, USA: ACM.

Barker, S. 2010. Personalizing access control by generalizing access control. In *Proceedings of the 15th ACM symposium on Access control models and technologies*, SACMAT '10, 149–158. New York, NY, USA: ACM.

Bertino, E. 2005. Pdl with preferences. In *Proc. of POLICY*, 213–222.

Bonatti, P. A.; Coi, J. L. D.; Olmedilla, D.; and Sauro, L. 2009. Rule-based policy representations and reasoning. In Bry, F., and Maluszynski, J., eds., *REWERSE*, volume 5500 of *Lecture Notes in Computer Science*. Springer. 201–232.

Cabalar, P., and Lorenzo, D. 2004. New insights on the intuitionistic interpretation of default logic. In de Mántaras, R. L., and Saitta, L., eds., *ECAI*, 798–802. IOS Press.

Eiter, T.; Fink, M.; Tompits, H.; and Woltran, S. 2004. Simplifying logic programs under uniform and strong equivalence. In *In LPNMR04*, 87–99. Springer.

Eiter, T.; Ianni, G.; Polleres, A.; and Schidlauer, R. June 2006. Answer set programming for the semantic web.

Eiter, T.; Fink, M.; and Moura, J. 2010. Paracoherent answer set programming. In Lin, F.; Sattler, U.; and Truszczynski, M., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, May 9-13, 2010*. AAAI Press.

Eiter, T.; Fink, M.; and Woltran, S. 2007. Semantical characterizations and complexity of equivalences in answer set programming. *ACM Trans. Comput. Logic* 8(3).

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. 1070–1080. MIT Press.

Kolovski, V. 2007. Logic-based access control policy specification and management.

Lifschitz, V.; Pearce, D.; and Valverde, A. 2000. Strongly equivalent logic programs. *ACM Transactions on Computational Logic* 2:2001.

Lifschitz, V.; Tang, L. R.; and Turner, H. 1999. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence* 25:369–389.

Pearce, D.; de Guzman, I. P.; and Valverde, A. 2000a. Computing equilibrium models using signed formulas. In *Proceedings of the First International Conference on Computational Logic*, CL '00, 688–702. London, UK: Springer-Verlag.

Pearce, D.; de Guzman, I. P.; and Valverde, A. 2000b. A tableau calculus for equilibrium entailment. In *In Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAUX 2000, LNAI 1847*, 352–367. Springer.

Pearce, D. 1997. A new logical characterisation of stable models and answer sets. In *In Proc. of NMELP 96, LNCS 1216*, 57–70. Springer.

Pearce, D. 2006. Equilibrium logic. *Ann. Math. Artif. Intell.* 47(1-2):3–41.

Reiter, R. 1987. *A logic for default reasoning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. 68–93.

Wang, Y.; Zhang, H.; Dai, X.; and Liu, J. 2010. Conflicts analysis and resolution for access control policies. In *Information Theory and Information Security (ICITIS), 2010 IEEE International Conference on*, 264 –267.

Woo, T. Y. C., and Lam, S. S. 1992. Authorization in distributed systems:a formal approach. *Security and Privacy, IEEE Symposium on* 0:33.

Zinn, C., and Intelligenz, L. F. K. Colosseum - an automated theorem prover for intuitionistic predicate logic based on dialogue games.