Towards An Attribute Based Constraints Specification Language

Khalid Zaman Bijon*, Ram Krishnan† and Ravi Sandhu*
*Institute for Cyber Security & Department of Computer Science
†Institute for Cyber Security & Department of Electrical and Computer Engineering
University of Texas at San Antonio

***Abstract*—Recently, attribute based access control (*ABAC*) has received considerable attention from the security community for its policy flexibility and dynamic decision making capabilities. In *ABAC*, authorization decisions are based on various *attributes* of entities involved in the access (e.g., users, subjects, objects, context, etc.). In an *ABAC* system, correct attribute assignment to different entities is necessary for ensuring appropriate access. Although considerable research has been conducted on *ABAC*, so far constraints specification on attribute assignment to entities has not been systematically studied in the literature. In this paper, we propose an attribute-based constraints specification language (*ABCL*) for expressing a variety of constraints on values that different attributes of various entities in the system can take. *ABCL* can be used to specify constraints on a single attribute or across multiple attributes of a particular entity. Furthermore, constraints on attributes assignment across multiple entities (e.g., attributes of different users) can also be specified. Finally, we demonstrate the usefulness of *ABCL* in practical usage scenarios including banking domains.**

Keywords: attribute based access control, constraints, language

I. Introduction

Over the last few years, attribute based access control (*ABAC*) has been emerging as a dominant form of access control due to its policy-neutral nature (that is, an ability to express different kinds of access control policies including DAC, MAC and RBAC) and dynamic decision making capabilities. Generally, *ABAC* regulates permissions of users or subjects to access system resources dynamically based on associated authorization rules with a particular permission. Thus, a user is able to exercise a permission on an object if the attributes of the user and object have a configuration satisfying the authorization rule specified for that permission. Hence, proper attribute assignment to these entities is crucially important in an *ABAC* system in order to avoid unauthorized accesses.

In this paper, we focus on constraint specifications on attribute assignment to the entities in *ABAC* as a mechanism to determine which entity should get which attribute values. By entities, we refer to users, subjects and objects which are common in access control systems. A user is an abstraction of human being, a subject is an instantiation of a user in the system and can refer to a particular session much like in RBAC and an object is a resource in the system. In general, constraints are an important and powerful mechanism for laying out higher-level access control policies of an organization. While *ABAC* is policy-neutral, it is also complex to manage. Thus it should have proper constraint specification and enforcement mechanisms in order to effectively configure required access control policies for an organization.

Constraint specification in *ABAC* is more complex than in RBAC since there are multiple attributes (unlike a single *role* attribute in RBAC) and attributes can take different structures (e.g., atomic or single-valued attributes such as *security-clearance* and *bank-balance* and set-valued attributes such as *role* and *group*). Constraints may exist amongst different values of a set-valued attribute (e.g. mutual exclusion on group memberships) and also on values across different attributes. For instance, suppose that an organization wants only their vice-presidents to get both a top-secret clearance and membership in their board-members email group. The *ABAC* system should have mechanisms to specify such constraints. In this case, there are three attributes for each user namely *role*, *clearance* and *group*. If the *role* attribute of a user is not 'vice-president', then his *clearance* and *group* attributes cannot have both the value of 'top-secret' and 'board-member-emails'. Note that these constraints are not concerned about users' access to objects directly. Instead, they focus on high-level requirements that a security architect would specify, which may indirectly translate into enabling or disabling accesses. This is much like separation of duty constraints in RBAC such as a particular employee cannot take both 'programmer' and 'tester' *roles* for the same project. Such a constraint eventually prevents the employee from simultaneously working on both developing code and testing it for a given project.

In general, the more expressive power a model has, the harder it is (if at all possible) to carry out many types of security analysis. It has already been shown that the safety problem of an *ABAC* system with infinite value domain of attributes is undecidable [22]. Nevertheless, *ABAC* is the leading mechanism that overcomes the limitations of discretionary access control (DAC) [17], mandatory access control (MAC) [16] and role-based access control (RBAC) [10]. NIST recognizes that *ABAC* allows an unprecedented amount of flexibility and security that makes it a suitable choice for large and federated enterprizes over other existing access control mechanisms [2]. Given that *ABAC* is known to be hard to analyze, constraint specification on attribute values is a powerful means to ensure that essential high-level access control requirements are met in a system that utilizes *ABAC*.

Our Contributions. We develop an attribute based constraint specification language (*ABCL*) for specifying constraints on attribute assignments. *ABCL* provides a mechanism to represent different kind of conflicting relations amongst attributes in a system in the form of relation-sets. Relation-sets contain different attribute values and *ABCL* expression expresses constraints on attribute assignments based on these values. There is considerable literature, such as [4], [7], [9], [11], [14], [18], [20], on the utility of attributes in managing various aspects of security in a system. Our work is the first investigation on how attributes themselves could be managed based on their intrinsic relationships. We also present an *ABCL* configuration for specifying powerful constraints in usage scenarios such as banking domains.

II. Related Work

Attribute Based Access Control. There is a sizable literature on *ABAC* in general. Damiani et al [4] described an informal framework for *ABAC* in open environments. Wang et al [20] proposed a framework that models an *ABAC* system using logic programming with set constraints of a computable set theory. Flexible access control system [9] can specify features of *ABAC* policies and provide a language that permits the specification of general constraints on authorizations. Yuan et al [21] described *ABAC* in the aspects of authorization architecture for web services. Lang et al [11] provided informal configuration of DAC, MAC, and RBAC through *ABAC*. These authors seek to develop an access control system either for open systems such as web, internet, etc., or to overcome the limitations of conventional access control models by utilizing attributes. Park et al [13] categorized attributes according to their mutability during execution of operations and developed a mechanism in which attributes of entities can be updated as a side-effect of an access. More recently, Jin et al [10] proposed an attribute based access control model in which they provide an authorization policy specification language and formal framework using which DAC, MAC and RBAC policies can be expressed. This literature focusses on *ABAC* in general and not much on constraints in *ABAC*.

Constraints. Several authors have focussed on issues in constraints specification in access control systems primarily in RBAC. Constraints in RBAC are often characterized as static separation of duty (SSOD) and dynamic separation of duty (DSOD). These two issues were addressed back to late 1980's when Clark et al [3] introduced SSOD and Sandhu [15] DSOD. A number of attempts have been initiated afterwards to identify numerous forms of SSOD and DSOD policies [5], [19] and to specify them formally [6], [8] in RBAC systems. The RCL-2000 language for specifying these policies in a comprehensive way was proposed by Ahn et al [1].[1] More recently, Jin et al [10] proposed an attribute based access control model in which they provide an authorization policy specification language that could also specify constraints on

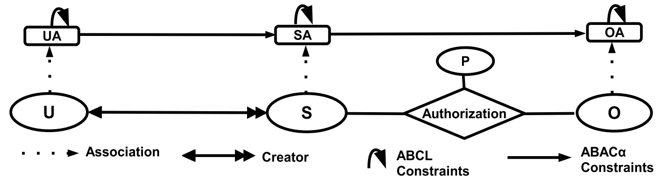

Fig. 1: *ABAC model with $ABAC_\alpha$ and* ABCL *Constraints*

attribute assignment. However, their constraints specification focuses on what values the attributes of subjects and objects may take given that users are currently assigned with particular attribute values. This is much like constraints on what roles can be activated in a user's session in RBAC given that a user is pre-assigned to a set of roles. Thus, prior work on constraint specification does not address *ABAC* comprehensively.

Attribute Based Encryption. This body of literature concerns a particular cryptographic enforcement mechanism for attribute based access control systems. Sahani et al [14] introduced the concept of Attribute Based Encryption (ABE) in which an encrypted ciphertext is associated with a set of attributes, and the private key of a user reflects an access policy over attributes. The user can decrypt if the ciphertext's attributes satisfy the key's policy. Goyel et al [7] improved expressibility of ABE which supports any monotonic access formula and Ostrovsky [12] enhanced it by including non-monotonic formulas. Several other attempts examines different variants of ABE. Basically, all these authors focus on improving secure encryption process by utilizing attributes.

III. Motivation and Scope

Attributes can capture identities, security clearances and classifications, roles, as well as location, time, strength of authentication, etc. As such *ABAC* supplements and subsumes rather than supplants currently dominant access control models including DAC, MAC and RBAC. figure 1 shows a typical *ABAC* model structure that contains users (*U*), subjects (*S*), objects (*O*) and different permissions (P). There are also user attributes (*UA*), subject attributes (*SA*) and object attributes (*OA*) associated with users, subjects and objects respectively. A subject is the representative of a user in the system. Each permission is associated with an authorization policy that determines whether a subject should get that permission on an object. An authorization policy compares the necessary subject and object attributes and any subject, associated with required attributes, can get the access. Hence, proper attribute assignment to the entities is crucially important in *ABAC*.

As discussed in related work, recently, an *ABAC* model called $ABAC_\alpha$ [10] proposed a policy specification language that could specify policies for authorizing a permission as well as constraints on attribute assignment. The constraints of $ABAC_\alpha$ are shown in the top row of figure 1 (horizontal solid lines with a single arrow-head). These constraints apply to values a subject attribute may get from its owner (user) when it is created, or an object attribute may get when the object is created or operated-on by a subject. $ABAC_\alpha$ constraints apply only when specific events such as a user modifying

[1]Several aspects of *ABCL* have been inspired by RCL-2000, including the use of conflict sets and the *oneelement* and *allother* operators. In dealing with general attributes rather than just the single attribute of role *ABCL* goes beyond RCL in many aspects.

a subject's attributes occur. In other words they are event specific. They relate the user attributes to the subject or the subject to the object depending on the event in question. *ABCL* constraints on the other hand are event independent and are to be uniformly enforced no matter what event is causing an attribute value to change. They are specified as restrictions on a single set-valued attribute or restrictions on values of different attributes of the same entity. *ABCL* constraints are depicted in the top row of figure 1 as arcs with a single arrow-head.

The central concept in *ABCL* is conflicting relations on attribute values which can be used to express notions such as mutual exclusion, preconditions, and obligations, amongst attribute values. For instance, suppose a banking organization utilizes a set-valued user (customer) attribute called *benefit* whose allowed values are {'bf_1', 'bf_2', ..., 'bf_6'}. Say that the bank wants to specify the following constraints: (a) a client cannot get both *benefits* 'bf_1' and 'bf_2', (b) a client cannot get more than 2 *benefits* from the subset {'bf_1', 'bf_3', 'bf_4'}, and (c) for 'bf_6' a client first needs to get 'bf_3'. Here, the first policy represents mutual exclusion conflict between 'bf_1' and 'bf_2', the second one is a cardinality constraint on mutual exclusion and the last one is a precondition constraint. A number of other conflicts among attributes may also exist.

Figure 2 gives a hierarchical classification of the attribute conflict-relationships based on two parameters: the number of entities and number of attributes allowed in a conflict relation. For example, each constraint in level 0 is concerned with conflicts among values of a single user attribute and it applies to each user independently. Level 1 allows constraints across different attributes of a single user. In level 2, constraints evaluate conflicting values of each attribute individually but across multiple users and in level 3 it can be across different attributes across multiple users. For instance, in above banking example, if a constraint restricts both *benefits* 'bf_1' and 'bf_2' from offering to a client simultaneously, the constraint falls in level 0. Here, the constraint concerned with the conflict between two values of single attribute *benefit*. Again, suppose, there is a *felony* attribute that represents the clients' felony history. If any value of *felony* restricts a client to get any *benefit*, this constraint falls in level 1. In this case, a conflict among certain values of two different attributes *benefit* and *felony* are addressed. Section V-A shows examples of several other constraints those fall in different levels of the relationship hierarchy. In the following sections, we present *ABCL* formalization and discuss them for user attributes in an *ABAC* model. However, *ABCL* is capable of expressing attribute assignment constraints of other entities as well, e.g. subject and objects. For simplicity and lack of space we focus exclusively on user attributes.

IV. ATTRIBUTE BASED CONSTRAINT LANGUAGE (*ABCL*)

We now formally present the elements of *ABCL*. *ABCL* consists of three basic components: the attributes of different entities in an *ABAC* model, a few basic sets and functions to capture different relationships amongst attributes, and a language for specifying constraints using basic sets and functions.

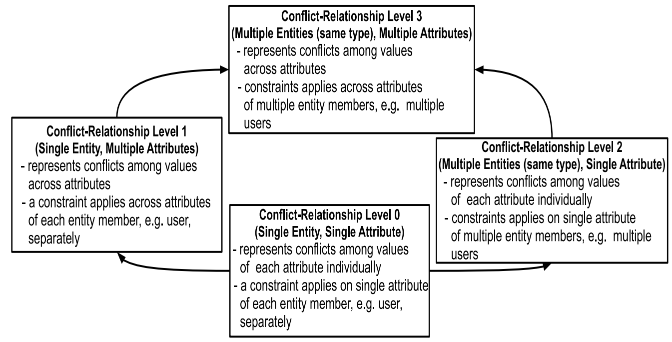

Fig. 2: *Attributes Relationship Hierarchy*

A. Basic Components of the ABAC Model

For the purpose of this paper, we use the basic framework of the $ABAC_\alpha$ model [10] as a representative *ABAC* model for *ABCL*. However, note that *ABCL* is not tailored for $ABAC_\alpha$ and can be similarly applied to other *ABAC* models.

A brief overview of $ABAC_\alpha$ is provided in table I. Like most access control models, $ABAC_\alpha$ consists of familiar basic entities: users (*U*), subjects (*S*) and objects (*O*). Each of these entities is associated with a respective set of attribute functions or simply *attributes* (*UA*, *SA* and *OA* respectively). Two types of attributes are considered in $ABAC_\alpha$: set-valued and atomic-valued. For example, *role* is a set-valued attribute since a user may take multiple *roles* in an organization. However, *security-clearance* is an atomic-valued attribute since a user takes only a single value for security clearance such as 'top-secret' or 'secret'. As shown in table I, an *attribute* is a function from the respective entity to a set of values that it can take (the **Range** of the *attribute*). The **Range** could be set or atomic-valued depending on the type of the attribute. A special attribute called **SubCreator** is used to keep track of the user that created a particular subject. Note that a user can create any number of subjects. The permissions that a subject can exercise on an object depends on the attribute values of the subject and object and the attribute-based authorization rule expressed for that permission in the system. Since *ABCL* is only concerned about constraints on what values these attributes can take and not on authorization rules for subject operations on objects or subject creation and other operations, the overview of $ABAC_\alpha$ provided in table I suffices for our purpose.

For specifying *ABCL* constraints, we specify additional derived functions for convenience. For each attribute, we define $\textbf{assignedEntities}_{U,att}$ (table II) that identifies the set of users that are assigned a particular value of that attribute. Similar functions can also be declared for subjects and objects.

B. Basic Sets and Functions of ABCL

Attribute conflict can occur in several ways. *ABCL* recognizes two types of conflict: values that have conflict with other values of the same attribute (referred to as single-attribute conflict) and values having conflict with the values of other attributes (referred to cross-attribute conflict). Note that single-attribute conflict is applicable only for set-valued attributes (e.g. mutually exclusive roles) while cross-attribute conflict applies to both atomic and set-valued attributes.

TABLE I. Basic sets and functions of *ABAC*

U, S and O represent finite sets of existing users, subjects and objects.
UA, SA and OA represent finite sets of user, subject and object attribute functions.
P represents a finite set of permissions.
For each att in $UA \cup SA \cup OA$, **Range**(att) represents the attribute's range, a finite set of atomic values.
SubCreator: $S \rightarrow U$. For each subject it gives the creator.
attType: $UA \cup SA \cup OA \rightarrow \{$set, atomic$\}$. Given an attribute name, this function will return its type as either set or atomic.
Each attribute function maps elements in U, S and O to atomic or set values.

$$\forall ua \in UA.\ ua: U \rightarrow \begin{cases} \textbf{Range}(ua) \text{ if } \textbf{attType}(ua)\text{=atomic} \\ 2^{\textbf{Range}(ua)} \text{ if } \textbf{attType}(ua)\text{=set} \end{cases}$$

$$\forall sa \in SA.\ sa: S \rightarrow \begin{cases} \textbf{Range}(sa) \text{ if } \textbf{attType}(sa) = \text{atomic} \\ 2^{\textbf{Range}(sa)} \text{ if } \textbf{attType}(sa)\text{=set} \end{cases}$$

$$\forall oa \in OA.\ oa: O \rightarrow \begin{cases} \textbf{Range}(oa) \text{ if } \textbf{attType}(oa)\text{=atomic} \\ 2^{\textbf{Range}(oa)} \text{ if } \textbf{attType}(oa)\text{=set} \end{cases}$$

TABLE II. Derived Functions from Basic *ABAC* Sets

For each $att \in UA$
assignedEntities$_{U,\,att}$: **Range** $(att) \rightarrow 2^U$ where
assignedEntities$_{U,att}(attval)=\{u \mid attval \in$ **Range**$(att) \wedge u \in U \wedge (att(u)=attval$ if **attType**(att)=atomic or $attval \in att(u)$ if **attType**(att)=set$)\}$

TABLE III. Declared *ABCL* Conflict Sets

1. Expression for declaring sets that represent conflicts among the values of a single attribute
For each $att \in UA$ and **attType**(att)=set there are zero or more ***Attribute_Set***$_{U,att}$ = $\{avset_1, avset_2, ..., avset_t\}$, where $avset_i$= $(attval, limit)$ in which $attval \in 2^{\textbf{Range}(att)}$ and $1 \leq limit \leq |attval|$.
2. Expression for declaring sets that represent value conflicts across multiple attributes
For each $Aattset \subseteq UA$ and $Rattset \subseteq UA$ there is zero or more ***Cross_Attribute_Set***$_{U,Aattset,Rattset}$=$\{$**attfun**$_1$, ..., **attfun**$_u\}$, where **attfun**$_i(att)=(attval, limit)$ in which $att \in Aattset \cup Rattset$ and ($attval \in 2^{\textbf{Range}(att)}$ if **attType**(att)=set or $attval \in$ **Range**(att) if **attType**(att)=atomic) and $0 \leq limit \leq |$attval$|$.

In order to specify these two types of conflict, *ABCL* facilitates the specification of two type of sets that may contain conflicting values for single and cross-attribute conflicts respectively and a formal language for precisely specifying constraints based on these conflicts. We discuss these sets in this subsection and the language in the following.

Item 1 and 2 in table III provide the mechanism for declaring sets for single-attribute and cross-attribute conflicts respectively. As shown in item 1, each ***Attribute_Set*** contains a set of values of an attribute that may have a particular type of conflict (mutual exclusion, precondition, inclusion, obligation, etc.). A separate ***Attribute_Set*** for each such conflict could be specified. As previously mentioned, the semantics of the constraints stated with respect to an ***Attribute_Set*** will be discussed in the next subsection. Each element of an ***Attribute_Set*** is an ordered pair (*attval*, *limit*) where *attval* contains the values that have some form of conflict and *limit* specifies the cardinality, that is the number of values in attval for which the conflict applies. The interpretation of *limit* could also be different, e.g. at least, exactly, at most, etc. The ***Attribute_Set*** declaration and initialization for the banking example of section III are as follows (the syntax for these expressions is shown in table IV).

Attribute_Set$_{U,benefit}$ *UMEBenefit*
UMEBenefit=$\{avset_1, avset_2\}$ where
$avset_1$=({'bf$_1$','bf$_2$'}, 1) and $avset_2$=({'bf$_1$','bf$_3$','bf$_4$'}, 2)
Attribute_Set$_{U,benefit}$ *PreconditionBenefit*
PreconditionBenefit=$\{avset_1\}$ where
$avset_1$=({'bf$_3$', 'bf$_6$'}, 1)

Here, $avset_1$ in *UMEBenefit* could indicate that the values 'bf$_1$' and 'bf$_2$' of the *benefit* attribute conflict with each other. Similarly, $avset_2$ could indicate that the *benefit* cannot take 2 or more of the values in the set {'bf$_1$', 'bf$_3$', 'bf$_4$'}. Note that the *limit* of *UMEBenefit* indicates that the number of elements from *attval* should be less than or equal to the value of *limit*. While, in *PreconditionBenefit* the number of elements from *attval* should be at least equal to *limit*.

As mentioned earlier, there could also be conflicts amongst values across different attributes of a user. Let us say in the banking example of section III, there is another user attribute called *felony* and its range is {'fl$_1$', 'fl$_2$', 'fl$_3$'}. The bank seeks to restrict a user to *benefit* 'bf$_1$' if she has ever committed felony 'fl$_1$' or 'fl$_2$'. This is a mutual exclusive conflict relation among the values of *benefit* and *felony*. These relations are represented as another type of relation-set called ***Cross_Attribute_Set*** which is formally defined in table III item 2. Each ***Cross_Attribute_Set*** is declared for two arbitrary sets of user attributes which are determined at declaration time. These two sets of attributes are represented as *Aattset* and *Rattset* and combination of certain values of the attributes in *Aattset* as a group has specific type of conflicts with certain values of each attribute in *Rattset*. In other words, values of the attributes of *Aattset* together restrict the values of each attribute in *Rattset*. Each element of a ***Cross_Attribute_Set*** is a function called **attfun** that returns the values of the attributes of *Aattset* and *Rattset* as an ordered pair (*attval*, *limit*) where *attval* represents the values and *limit* is the cardinality. ***Cross_Attribute_Set*** declaration and initialization for the banking example are as follows (the syntax for these expressions is shown in table IV).

Cross_Attribute_Set$_{U,Aattset,Rattset}$ *UMECFB*
Here, *Aattset*= {*felony*} and *Rattset*= {*benefit*}
UMECFB={**attfun**$_1$} where
attfun$_1$(*felony*)=(*attval*, *limit*)
where *attval*={'fl$_1$', 'fl$_2$'} and *limit*=1
attfun$_1$(*benefit*)=(*attval*, *limit*)
where *attval*={'bf$_1$'} and *limit*=0

Using the set above, one can state if at least one value from {'fl$_1$','fl$_2$'} is assigned to *felony* of a user, 'bf$_1$' should not be assigned to the *benefit* attribute of that user.

ABCL also has two nondeterministic functions, *oneelement* and *allother*. The *oneelement*(X) returns one element x_i from set X and in a constraint expression it is written as **OE**(X). Multiple occurrences of **OE**(X) in a single *ABCL* expression selects the same element x_i from X. The *allother*(X) returns a subset of elements from X by taking out one element with **OE**(X). We usually write *allother* as **AO**. These two functions are related

TABLE IV. Syntax of Language

Declaration of the *Attribute_Set* and *Cross_Attribute_Set*:
<attribute_set_declaration> ::= <atribute_set_type> <set_identifier>
<attribute_set_type> ::= ***Attribute_Set***$_{U,\langle \text{attname} \rangle}$ | ***Attribute_Set***$_{S,\langle \text{attname} \rangle}$ | ***Attribute_Set***$_{O,\langle \text{attname} \rangle}$
<cross_attribute_set_type> ::= ***Cross_Attribute_Set***$_{U,\langle Aattset \rangle,\langle Rattset \rangle}$ | ***Cross_Attribute_Set***$_{S,\langle Aattset \rangle,\langle Rattset \rangle}$
| ***Cross_Attribute_Set***$_{O,\langle Aattset \rangle,\langle Rattset \rangle}$
<*Aattset*> ::= {<attname>, <attname>*}
<*Rattset*> ::= {<attname>, <attname>*}
<set_identifier> ::= <letter> | <set_identifier><letter> | <set_identifier><digit>
<digit> ::= *0|1|2|3|4|5|6|7|8|9*
<letter> ::= $a|b|c|....|x|y|z|A|B|C|...|X|Y|Z$
Constraint Expressions:
<statement> ::= <statement> <connective> <statement> | <expression>
<expression> ::= <token> <atomiccompare> <token> | <token> <atomiccompare> <size>
| <token> <atomiccompare>|<set>| | <token> <atomiccompare> <set> | <token>
<token> ::= <token> <setoperator> <term> | <term> | |<term>|
<term> ::= <function> (<term>) | <attributefun> (<term>) | **OE** (<relationsets>).<item>
| **OE** (<term>) | **OE** (<set>) | **AO** (<term>) | **AO** (<set>) | <attval>
<connective> ::= $\wedge$ | $\Rightarrow$
<setoperator> ::= $\in$ | $\cup$ | $\cap$ | $\notin$
<atomicoperator> ::= + | < | > | $\leq$ | $\geq$ | $\neq$ | =
<set> ::= *U*| *S*| *O*
<relationsets> ::= <set_identifier>
<attname> ::= ua_1 | ua_2 | ... | ua_x | sa_1 | sa_2 | ... | sa_y | oa_1 | ... | oa_z
<attval> ::= 'ua_1val_1' | 'ua_1val_2' | ... | 'ua_xval_r' | 'sa_1val_1' | 'sa_1val_2' | ... | 'sa_yval_s' | 'oa_1val_1' | ... | 'oa_zval_t'
<size> ::= ϕ | 1 | ... | N
<item> ::= *limit*| *attval*| **attfun**(<attname>).*limit*| **attfun**(<attname>).*attval*
<attributefun> ::= ua_1 | ua_2 | ... | ua_x | sa_1 | sa_2 | ... | sa_y | oa_1 | ... | oa_z
<function> ::= **SubCreator** | **assignedEntities**$_{U,\langle \text{attname} \rangle}$ | **assignedEntities**$_{S,\langle \text{attname} \rangle}$ | **assignedEntities**$_{O,\langle \text{attname} \rangle}$

by context, because for any set *S*, {**OE**(*S*)}∪**AO**(*S*)=*S*, and at the same time, neither is a deterministic function. An example use of **OE** is as follows.

Requirement: No user can get more than three benefits.
Expression: $|benefit(\mathbf{OE}(U))| \leq 3$

OE(*U*) means a single user from *U* and *benefit*(**OE**(*U*)) returns all benefits that are assigned to that user. This expression ensures that a single user cannot have more than three benefits. Later, we will see how **AO** is used in an *ABCL* expression.

C. Syntax of ABCL

The syntax of *ABCL* is defined by the grammar given in table IV in Backus Normal Form (BNF). The grammar contains declaration syntax for both type of relation-sets ***Attribute_Set*** and ***Cross_Attribute_Set*** and syntax for constraint expressions.

V. *ABCL* USEAGE SCENARIO

In this section, we present an extensive case study in which a large set of *ABCL* expressions is generated to capture various access control requirements of a banking organization.

A. Security policy specifications for Banking Organizations

We present *ABCL* constraints for several high-level security requirements in a banking organization. Due to the space limitation, we only show constraints for user attribute management in this context. In a banking organization, let us consider a finite set of existing users (U) in which a user is a human being and could be of different types, e.g. client, junior employee.

Table V shows different user attributes, their types and ranges in this system. Each user is assigned an attribute *id* which is a unique identifier. Attribute *uType* represents the type of a user and *orgType* represents the organization a user belongs to. There is a *role* attribute representing various job descriptions of a user such as 'customer', 'cashier', etc. The bank might provide a number of benefits i.e. bonus, cash back rate, etc, to the customers which is represented by the *benefit* attribute. Attribute *felony* represents if the user has any felony record and *loan* and *cCard* represent granted loans and credit cards to a user respectively. Suppose that the banking authority wishes to specify the following security policy requirements for user attribute management. The *ABCL* formalism for these requirements are given in following subsection. We also show the conflict-relationship level of each of these constraints.

Req# 1: A user can get at most 5 *benefits*. (Relationship lev.0)
Req# 2: A user cannot hold the 'president' and 'vice-president' *roles* simultaneously. (Lev.0)
Req# 3: A user cannot get both *benefits* 'bf_1' and 'bf_2'. (Lev.0)
Req# 4: A user can get at-most 5 *loans* and *cCards*. (Lev.1)
Req# 5: If a user has *felony* records 'fl_1' and 'fl_2', she cannot get more than one *benefit* from {bf1, bf2, bf3}. (Lev.1)
Req# 6: If a user is a 'client', she cannot get certain *roles*, e.g. 'cashier', 'manager'. (Lev.1)
Req# 7: No more than 12 users can get a 'car' *loan*. (Lev.2)
Req# 8: *ids* of two users cannot get the same value.(Lev.2)
Req# 9: If a user has *felony* 'fl_1' and belongs to 'org_1', no users from 'org_1' can get *benefit* 'bf_1'. (Lev.3)

B. Formal ABCL *Specification for Banking Organization*

Table VI shows declaration and initialization of the *ABCL* sets for representing necessary relations among attributes

TABLE V. User Attributes (UA)

Attribute	attType	Range
id	atomic	{'id_1','id_2', ..., 'id_x'}
uType	atomic	{'client', 'junior', 'senior', 'leader'}
orgType	set	{'org_1', 'org_2', ..., 'org_{20}'}
role	set	{'customer', 'cashier', 'manager', 'president', 'vice-president'}
benefit	set	{'bf_1', 'bf_2', 'bf_3', ..., 'bf_{10}'}
felony	set	{'fl_1', 'fl_2', 'fl_3', ..., 'fl_8'}
loan	set	{'car', 'house', 'education'}
cCard	set	{'card_1', 'card_2', ..., 'card_{12}'}

TABLE VI. *ABCL* Sets Declaration and Initialization:

1. *Attribute_Set* Declaration and Initialization:

Attribute_Set$_{U,benefit}$ *UMEBenefit*
UMEBenefit={avset_1, avset_2}
avset_1=({'bf_1', 'bf_2'}, 1), avset_2=({'bf_2', 'bf_3', 'bf_4', 'bf_5'}, 2)
Attribute_Set$_{U,role}$ *UMERole*
UMERole={avset_1}
avset_1=({'president', 'vice-president'}, 1)

2. *Cross_Attribute_Set* Declaration and Initialization:

Cross_Attribute_Set$_{U, \{uType\}, \{role\}}$ *UMECTR*
UMECTR={**attfun**$_1$}
attfun$_1$(*uType*)=({'client'},1)
attfun$_1$(*role*)=({'cashier','manager','president', 'vice-precident'}, 0)
Cross_Attribute_Set$_{U, \{felony\}, \{benefit\}}$ *UMECFB*
UMECFB={**attfun**$_1$, **attfun**$_2$}
attfun$_1$(*felony*)=({'fl_1','fl_2'},2), **attfun**$_1$(*benefit*)=({'bf_1','bf_2','bf_3'},1)
attfun$_2$(*felony*)=({'fl_1'},1), **attfun**$_2$(*benefit*)=({'bf_2'}, 0)
Cross_Attribute_Set$_{U, \{felony, orgType\}, \{benefit\}}$ *UMECFOB*
UMECFOB={**attfun**$_1$}
attfun$_1$(*felony*)=({'fl_1'},1), **attfun**$_1$(*orgType*)=({'org_1'}, 1),
attfun$_1$(*benefit*)=({'bf_1'}, 0)

for specifying above security policies for the banking organization. *UMEBenefit* contains mutual exclusive values of the *benefit* attribute and *UMERole* represents mutual exclusive roles. Similarly, mutual exclusive conflicts of *uType* with *role*, *felony* with *benefit*, and *felony* and *orgType* with *benefit* attributes are represented by the ***Cross_Attribute_Set****s* *UMECTR*, *UMECFB*, and *UMECFOB* respectively. *ABCL* expressions for the above discussed security policies are:

Req# 1: $|benefit(\mathbf{OE}(U))| \leq 5$.

Req# 2: $|\mathbf{OE}(UMERole).attset \cap role(\mathbf{OE}(U))| \leq \mathbf{OE}(UMERole).limit$

Req# 3: $|\mathbf{OE}(UMEBenefit).attset \cap benefit(\mathbf{OE}(U))| \leq \mathbf{OE}(UMEBenefit).limit$

Req# 4: $|cCard(\mathbf{OE}(U)) + loan(\mathbf{OE}(U))| \leq 5$

Req# 5: $|\mathbf{OE}(UMECFB)(felony).attset \cap felony(\mathbf{OE}(U))| \geq \mathbf{OE}(UMECFB)(felony).limit \Rightarrow |\mathbf{OE}(UMECFB)(benefit).attset \cap benefit(\mathbf{OE}(U))| \leq \mathbf{OE}(UMECFB)(benefit).limit$

Req# 6: $|\ \mathbf{OE}(UMECTR)(uType).attset \cap uType(\mathbf{OE}(U))| \geq \mathbf{OE}(UMECTR)(uType).limit \Rightarrow |\mathbf{OE}(UMECTR)(role).attset \cap benefit(\mathbf{OE}(U))| \leq \mathbf{OE}(UMECTR)(role).limit$

Req# 7: $|\mathbf{assignedEntities}_{U,\, loan}(\text{'car'})| \leq 12$

Req# 8: $id(\mathbf{OE}(U)) \neq id(\mathbf{OE}(\mathbf{AO}(\mathbf{OE}(U))))$

Req# 9: $|\mathbf{OE}(UMECFOB)(felony).attset \cap felony(\mathbf{OE}(U))| \geq \mathbf{OE}(UMECFOB)(felony).limit \wedge |\mathbf{OE}(UMECFOB)(orgType).attset \cap orgType(\text{OE(U)})| \geq \mathbf{OE}(UMECFOB)(orgType).limit \Rightarrow |\mathbf{OE}(UMECFOB)(benefit).attset \cap (benefit(\mathbf{OE}(U)) \cup benefit(\mathbf{OE}(\mathbf{AO}(U))))| \leq \mathbf{OE}(UMECFOB)(benefit).limit$

VI. Conclusion

Relationship constraints among attributes is an important factor for attribute assignment in *ABAC*. We have developed *ABCL* for specifying these constraints on attribute assignments. *ABCL* configurations for a banking organization provides its expressiveness for generating various constraints for fulfilling an organization's security requirements. In future, we plan to explore *ABCL* configurations for various RBAC constraints specification given in [1] and analyze *ABCL* enforcement complexities.

Acknowledgement. This work is partially supported by the National Science Foundation and AFOSR MURI projects.

References

[1] G. J. Ahn and R. Sandhu. Role-based authorization constraints specification. *ACM Trans. Inf. Syst. Secur.*, 3(4):207–226, Nov. 2000.
[2] V. C. Hu et al. Guide to attribute based access control (ABAC) definition and considerations (draft). *NIST Special Publication*, 2013.
[3] D. D. Clark and D. R. Wilson. A Comparison of Commercial and Military Computer Security Policies. In *Proc. of the IEEE S&P*, 1987.
[4] E. Damiani, S. D. C. Di Vimercati, and P. Samarati. New paradigms for access control in open environments. In *Proc. of the ISSPIT*, 2005.
[5] D. Ferraiolo, J. Cugini, and R. Kuhn. Role-based access control (RBAC): Features and motivations. In *Proc. of the 11th ACSAC*, 1995.
[6] V. D. Gligor et al. On the formal definition of separation-of-duty policies and their composition. In *Proc. of the IEEE S&P*, 1998.
[7] V. Goyal et al. Attribute-based encryption for fine-grained access control of encrypted data. In *Proc. of the ACM CCS*, 2006.
[8] T. Jaeger. On the increasing importance of constraints. In *Proc. of the ACM RBAC*, 1999.
[9] S. Jajodia et al. Flexible support for multiple access control policies. *ACM TODS*, 26(2):214–260, 2001.
[10] X. Jin, R. Krishnan, and R. Sandhu. A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC. In *DBSec*, 2012.
[11] B. Lang, I. Foster, F. Siebenlist, R. Ananthakrishnan, and T. Freeman. A flexible attribute based access control method for grid computing. *Journal of Grid Computing*, 7(2):169–180, 2009.
[12] R. Ostrovsky, A. Sahai, and B. Waters. Attribute-based encryption with non-monotonic access structures. In *Proc. of the ACM CCS*, 2007.
[13] J. Park and R. Sandhu. The UCON_{ABC} usage control model. *ACM Transactions on Information and System Security (TISSEC)*, 7(1), 2004.
[14] A. Sahai and B. Waters. Fuzzy identity-based encryption. In *Proc. of the EUROCRYPT*. 2005.
[15] R. Sandhu. Transaction control expressions for separation of duties. In *Proc. of the 4th ACSAC*, 1988.
[16] R. S. Sandhu. Lattice-based access control models. *IEEE Computer*, 26(11), 1993.
[17] R. S. Sandhu and P. Samarati. Access control: Principle and practice. *Communications Magazine, IEEE*, 32(9):40–48, 1994.
[18] C. Schläger, M. Sojer, B. Muschall, and G. Pernul. Attribute-based authentication and authorisation infrastructures for e-commerce providers. In *Proc. of the EC-Web*. 2006.
[19] R. T. Simon and M. E. Zurko. Separation of duty in role-based environments. In *Proc. of the IEEE CSFW*, 1997.
[20] L. Wang, D. Wijesekera, and S. Jajodia. A logic-based framework for attribute based access control. In *Proc. of the ACM FMSE*, 2004.
[21] E. Yuan and J. Tong. Attributed based access control (ABAC) for web services. In *Proc. of the IEEE ICWS*, 2005.
[22] X. Zhang, R. Sandhu, and F. Parisi-Presicce. Safety analysis of usage control authorization models. In *Proc. of the ASIACCS*, 2006.