A Metamodel for Hybrid Access Control Policies

Jamal Abd-Ali[1*], Karim El Guemhioui[1], Luigi Logrippo[1, 2]

[1] Université du Québec en Outaouais, Informatique et ingénierie, Gatineau, Québec, Canada.
[2] University of Ottawa, Electrical Engineering and Computer Science, Ottawa, Ontario, Canada.

* Corresponding author. Email: Jamal.Abd-Ali@uqo.ca.
Manuscript submitted September 15, 2014; accepted June 10, 2015.
doi: 10.17706/jsw.10.7.784-797

Abstract: Modelling is a proven technique to communicate and illustrate complex specifications in a wide range of disciplines. Access control (AC) specification is not an exception in this regard. Actually, it is characterized by the sensitivity and criticality of its contents where clarity and formalism are yet essential desired goals. In a metamodelling approach where textual languages and visual models are two equivalent forms of specifications, we propose an AC metamodel, setting the stage for its derived textual language. Our metamodel is characterized by its formal semantics, its modularity and refinement method, and its integration means for concurrent application of multiple reusable AC models. These characteristics enable AC specification with better readability, clarity, unambiguity and properties verification support.

Key words: Access control, metamodelling, formal semantics.

1. Introduction

Access Control (AC) is concerned with the restriction of the activities of legitimate and authenticated users of a system. More specifically, AC aims to restrict access to objects in order to protect their integrity and prevent their undesired use or disclosure. A principle underlying an AC specification can be applied repeatedly in the same organization. Each AC principle can be captured in a model that encompasses its invariant concepts and its decision logic in terms of these concepts and their associations. For example, we can mention the principle of granting access privileges based on the subject's role (i.e., position or mission) in the organization; this principle is embodied in the well-known Role Based Access Control model (RBAC) [1]. This model involves the following concepts: Subject, Object, Role, Permission, and their associations. In this model, the subjects are associated with their roles, and each role is associated with its appropriate access permissions on objects. Several other AC models are well-known in a variety of activity domains; amongst others we can mention Bell LaPadula (BLP) [2], BIBA [3], and Chinese Wall (CW) [4]. Although these models are seldom used in their 'pure' form, they are at the basis of many practical models.

An AC model allows the specification of an AC principle in a way that eases the formulation of its application. The model is reused each time we apply its underlying AC principle in a repeatable way. Thus, an AC model spares us the effort of specifying repeatedly its underlying logic; this logic is specified once for all when the AC model is elaborated. Nevertheless, many AC specification languages whose syntax and semantics are based on AC models (e.g., XACML [5], Ponder [6]) are self-contained and don't take advantage of any formally specified model.

AC modelling has an unexploited potential to contribute to the formal specification of AC policies. However two shortcomings can be noticed: 1) not all the AC models are formally specified, and therefore

they cannot endow the specification languages they underlie with the formal semantics they do not have themselves; 2) since AC specification can apply more than one AC principle concurrently (e.g., RBAC and Chinese Wall at the same time), we need a modelling approach that is not tailored to a single AC model. In the remainder of this paper, AC specifications (respectively policies) that apply more than one AC model are called hybrid specifications (respectively policies). In addition, we call integration the process of handling concurrently multiple AC models. As a response to the above two shortcomings, we propose a set of formally specified AC models, then we present an integration model that supports the integration of multiple AC models. In this paper, we consider four well-known AC models and explain how to define their formal specifications, including their AC decision logic. We show how our integration model provides a uniform integration means applicable to any set of formally specified AC models. Based on these models, we elaborate a textual specification language that draws its syntax and grammar from their structural elements, and draws its semantics from their underlying decision logic.

Although UML [7] is not the only modelling language that suits our approach, we will follow its notation and terminology for the sake of its large adoption as a de facto modelling language standard.

In Section 2, we introduce the concept of AC metamodelling and present four AC metamodels; in Section 3, we detail our approach to formally specifying our models. Section 4 introduces the integration metamodel. Section 5 considers related work. Section 6 concludes the paper with a review of its contribution.

2. Access Control Metamodels and Instances

The concepts captured in AC models pertain to two groups: 1) those related to the AC domain in general like "Object", "Subject" or "AccessMode", and 2) those specific to each particular AC model, like "Role" in the RBAC model. An instance of an AC model is itself a model at a lower abstraction level, created by replacing the AC model elements with their specific corresponding instances; e.g., in Fig. 1, Sam is an instance of Subject, and CEO is an instance of Role. According to the metamodelling paradigm [8], a model can have many instances at a lower abstraction level; each specifying a particular application (utilization) of that model. Such a model is called a metamodel and it defines a language to specify lower abstraction level models called its instances. For example, an instance of the RBAC model would be one that specifies the utilization of the RBAC model in a specific organization with its specific role hierarchy, its specific employees as subjects, and its specific objects.

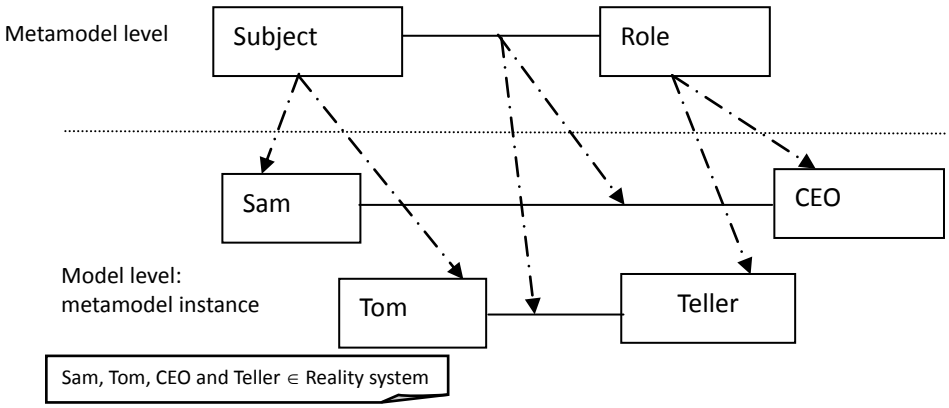

Fig. 1. Metamodelling levels.

Fig. 1 shows an example of a metamodel and a model, instance of the upper level metamodel. This very simple metamodel represents the principle that a subject must be associated to a role. It consists of the metamodelling elements: Subject, Role, and their association. At a lower abstraction level, the model, instance of the metamodel, is composed of Sam and Tom as instances of the element Subject, and CEO and

Teller as instances of Role. Dotted arrows indicate instantiation, the arrows pointing toward the instances. Furthermore, the associations between, respectively, Sam and CEO, and Tom and Teller, are instances of the higher-level association between their metamodelling elements Subject and Role.

Finally, Sam, CEO, Tom, Teller, and their respective associations represent themselves a model of some reality resulting from the application of RBAC in a specific organization. From this perspective, RBAC constitutes a metamodel that defines a language for the specification of lower level abstraction models called its instances. Of course, the modelling principles illustrated in this example extend to all the AC models introduced in Section 1; and therefore these latter ones will be more accurately called AC metamodels in the remainder of this paper.

2.1. AC Metamodels and Decision Systems

In addition to elements representing concepts and their associations, our AC metamodels encompass the logic behind AC decisions in response to access requests. Decisions are issued by applying logical rules expressed in terms of AC metamodel elements, hence our introduction of the concept of decision system. A decision system is a system that returns a decision in response to a query. Specific decisions are instances of the metamodelling element Decision which is specified according to the application domain. In the AC domain, Decision represents the type of any possible AC response to an access request. In our approach, an AC metamodel instance is also a decision system. Every AC metamodel has a special element named DecisionHandler whose mission is to encapsulate the decision logic allowing to determine the AC decision in response to an access request. Thus, a DecisionHandler instance is itself a decision system. Each AC metamodel specializes the DecisionHandler according to its specific decision making needs, as shown in Fig. 2. The element ACmetaModelElement is a generalization of any element of the AC metamodels other than DecisionHandler. It is shown in this view to indicate that every DecisionHandler specifies its decision logic in terms of other AC metamodel elements. In the following AC metamodels illustrations, and for the sake of simplicity and readability, we will not show all the associations between DecisionHandler and the remaining AC metamodel elements. These associations are represented in the association "uses" in Fig. 2.

In the remainder of this section we present four AC metamodels: Chinese Wall, Bell LaPadula, BIBA, and RBAC. In the remainder of this paper, we adopt the following convention: metamodelling elements start with an uppercase letter, while their instances start with a lowercase; subject represents an instance of the element Subject.

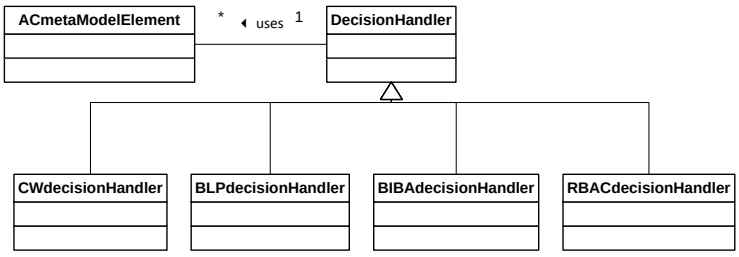

Fig. 2. Decision handler specializations in AC metamodels.

2.2. Kernel Access Control Elements

We introduce here AC elements that represent shared and fundamental elements of our AC metamodels. These elements are called kernel AC elements and they are defined below and illustrated in Fig. 3.

Object: represents the concept of an object as a resource of a system.

ObjectsGroup: represents a set of objects sharing some properties.

Subject: represents the concept of a subject as an entity that may request access to an object.

SubjectsGroup: represents a set of subjects sharing some characteristics.

AccessMode: specifies the different access modes as actions that a subject may perform on an object. In this paper, we only consider the instances of AccessMode representing the following actions: Read, Write, and Execute.

AdditionalAttribute: represents a construct associated to a metamodel element to support the specification of some property of that element. AdditionalAttribute has a name and a type that suits the specification of a property.

Query: represents an access request on an object by a subject. The object aimed by the access request is called the targeted object.

Query associations to respectively Object, Subject, and AccessMode (see Fig. 3) indicate that we respectively associate to an instance of Query: (1) An instance of Object representing the targeted object; (2) An instance of Subject representing the subject requesting the access; (3) An instance of AccessMode representing the requested action by the subject. The Query attribute isCurrent, of Boolean type, allows identifying the current Query instance as the query to process. This attribute is needed when we have to take into account other queries than the current one, since previous queries of a subject may influence an AC decision.

EnvironmentalAttribute: similar to AdditionalAttribute, but used to describe some property of any relevant entity of the environment that is not represented in the AC metamodel elements. It is used to hold information related to access request events such as time, place, emergency level of a context, temperature, etc. Hence, the association between EnvironmentalAttribute and Query (see Fig. 3), since an AC decision may depend on the state of the environment.

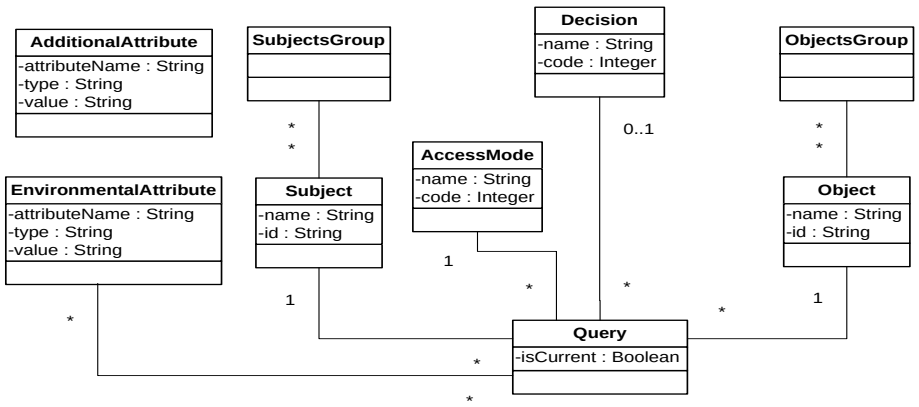

Fig. 3. Kernel AC elements.

Decision: represents the concept of a decision as a response issued by the AC about an access request. In this paper, only the following instances of Decision are considered: Permit, Deny, Indeterminate, NotApplicable.

The kernel elements are part of every AC metamodel. Thus in a hybrid policy, the subjects, the objects, and the query content are part of every AC metamodel instance. Furthermore notice the key role of Query instance in the specification of the decision logic, regardless of the considered AC metamodel.

2.3. Chinese Wall AC Metamodel

According to the Chinese Wall (CW) AC principle, in order to avoid a conflict of interests and other undesired situations, when a subject has accessed some objects, we must prevent it from accessing some other particular objects.

Consequently, we separate the objects in subsets called classes. These classes constitute a conflict group. The Chinese Wall AC logic can be formulated as follows:

Considering the classes C1, C2, …, Cn of a conflict group CG;

If a subject s has accessed any object belonging to a class Ci of CG, s will not be allowed to access any

object belonging to another class Ck of CG, with k≠ i. In other words ((AccessedBy-s ∩ (C1 ∪ C2 …∪ Cn)) ⊆ Cx) must remain true for any subject s, where AccessedBy-s designates the set of objects s has accessed, and Cx designates a single class of the conflict group CG.

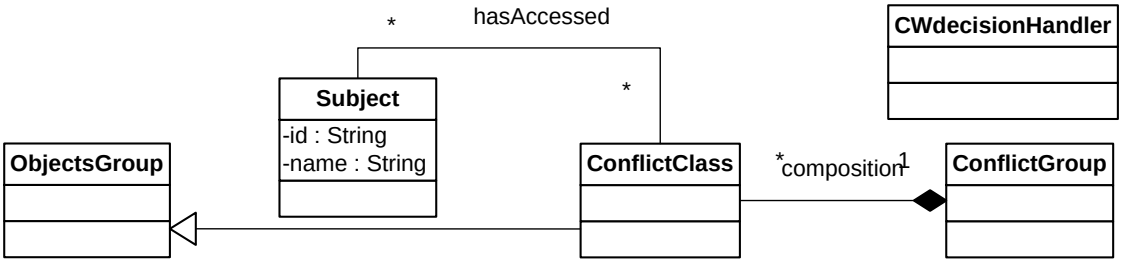

Fig. 4. Chinese Wall metamodel.

The CW metamodel shown in Fig. 4 illustrates the above mentioned concepts. ConflictClass denotes a class in a conflict group. The represented associations and their multiplicities reflect notable facts: Several conflict groups may coexist in the same CW model (instance of the depicted metamodel), but a conflict class may belong to only a single conflict group; furthermore, a subject cannot access more than one class in any conflict group. Nevertheless, a subject can access many classes pertaining to distinct conflict groups. Subject and ObjectsGroup are already introduced as kernel AC elements. CWdecisionHandler is the specialization of DecisionHandler for CW.

2.4. Bell LaPadula AC Metamodel

According to the BLP AC principle, the AC is based on the sensitivity levels of the targeted objects and the clearance levels of the subjects. A sensitivity level indicates to which extent an object is critical regarding its access risks, while a clearance level indicates to which extent we can trust a subject.

In addition, BLP associates objects with compartments which represent sets of objects having some common characteristics, and associates every subject with compartments that are of concern to it. According to BLP, an access request is accepted if the following two conditions are satisfied: (1) the clearance level of the subject is greater or equal to the sensitivity level of the targeted object; (2) the targeted object is associated with a compartment included in a compartment associated with the subject.

Thus the AC depends on comparison of couples, each couple indicating a clearance-sensitivity level and a compartment (level, compartment); such a couple is named a classification level.

We propose a metamodel of BLP, illustrated in Fig. 5, with the following elements.

Clearance-Sensitivity: an element indicating a level which stands for clearance or sensitivity depending on whether it is associated with an object or a subject.

Compartment: an element indicating a group of objects.

ClassificationLevel: this element represents the above introduced concept of classification level. An instance of this element is a couple referencing a level and a compartment.

The association multiplicities in the metamodel reflect the following:

A subject has one Clearance-Sensitivity instance, and zero or more compartments;

An object has one Clearance-Sensitivity instance, and zero or more compartments;

A subject or an object has one ClassificationLevel.

The reflexive association “LessThan” of the element ClassificationLevel, allows the specification of comparisons between instances of ClassificationLevel.

2.5. BIBA AC Metamodel

BIBA is another AC principle based on the above mentioned classification levels, but whose main concern is the information integrity level. This level is an indicator of the extent to which the information is not corrupted, not intentionally or unintentionally changed, and trustworthy. A subject with a given integrity level indicating to which extent it is trustworthy, is prevented from reading information with a lower integrity level; it may however modify such information. The rationale being that some information may compromise the level of trust of some higher integrity level information. To model BIBA, the only new concept needed is the integrity classification level, represented by the element BIBAclassificationLevel as illustrated in Fig. 6. The elements Subject and Object were already introduced as kernel AC elements.

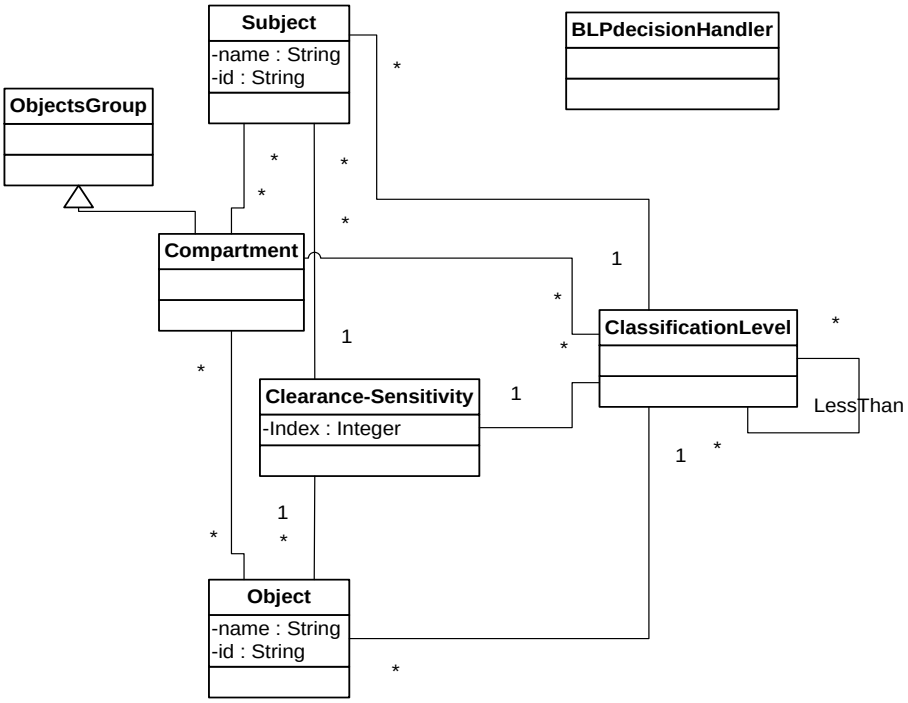

Fig. 5. Bell Lapadula metamodel.

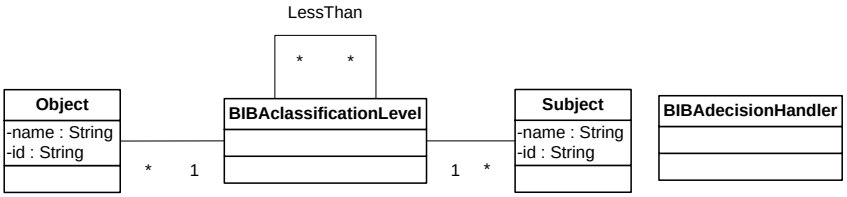

Fig. 6. BIBA metamodel.

2.6. RBAC AC Metamodel

RBAC is the best known AC model. According to RBAC, the AC decision relies on two main rules: (1) the position of a subject defined by its role allows specifying its access privileges, so applying RBAC implies specifying the roles and the access privileges for each role; (2) roles are structured in hierarchies in which a role has at least all the privileges of its children roles.

To describe RBAC, we have elaborated the metamodel illustrated in Fig. 7. Its specific elements are:

Role: This element represents the concept of position in an organization;

Permission: This element represents the access privileges associated with a role. When an instance **p** of Permission is associated to an instance r of Role, it means that the role **r** has the permission **p** to access the

objects associated to **p** with the access modes indicated by the instances of AccessMode associated to **p**.

Role associations to itself and to Subject: The reflexive association "subRole" of the element Role, allows the specification of a role hierarchy. The association between Subject and Role allows specifying that a subject takes on a particular role.

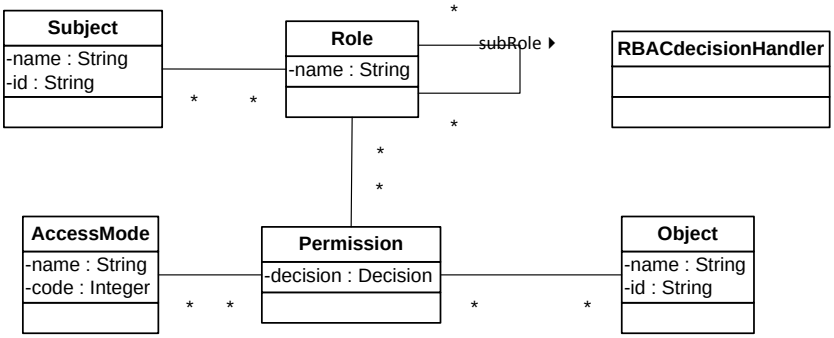

Fig. 7. RBAC metamodel.

3. Formal Specification of AC Metamodels

Our representation of AC metamodels relies on a subset of UML. Such modelling representation is neither complete nor ambiguity-free. Each of the metamodels we have presented identifies the involved concepts and their associations, but without formally determining how an instance of the metamodel returns an AC decision in response to an AC request. First-Order Logic (FOL) is known to be suitable for relating individuals to their types, for specifying relationships between individuals, and for expressing a decision logic based on the aforementioned relations. Therefore, we will formally specify our metamodelling elements, their attributes, and their associations, using dedicated and reusable predicates. Thereafter, we will elaborate the formal specification of the decision logic in terms of these predicates.

Our metamodelling elements will be endowed with logic; this will be achieved by attaching special attributes holding logical expressions (clauses) to classes in the AC metamodel. Such expressions allow the definition of rules applicable to the metamodel elements, and can be used to evaluate the truth values of other predicates. We respectively call such special attributes and their type "Clause attributes" and "Clause". An instance of a Clause is a clause as logical expression. The left hand side of a clause is a predicate (called head), while its right hand side (called body) is a conjunction of predicates that entails the truth of the head. For example, consider the following clause:

$$\text{Human(X)} \leftarrow \text{Walk(X), Talk(X).} \quad (1)$$

Clause (1) indicates that, for an individual X when the predicates Walk(X) and Talk(X) evaluate to true, then Human(X) is true. Human is the head of the clause, while the body consists of the logical conjunction of the two predicates Walk and Talk. The comma separator denotes the logical AND operator. When a predicate or a logical conjunction of predicates are followed by a dot this means that they are set to true. Furthermore, our modelling language is a subset of UML limited to classes, attributes, associations, generalization, and specialization, but augmented with the type Clause as a predefined type of Clause attributes. We propose mapping to FOL for this subset of UML. It is in most cases a direct one to one mapping that we can present with a concise introduction and simple examples as follows.

We distinguish FOL clauses with the special font "Courier new", and when confusion is possible, we use "/*" and "*/" to delimitate comments. For better readability, we also write in bold the clauses heads.

The declaration of a Class X having a list of attributes atti with their respective types Yi, is mapped to IsType(X)and DeclaredAttribute(X, atti, Yi). In addition every Clause attribute is mapped without changes except the replacement of "Self" by the current instance when the class is instantiated. When the Clause

Attribute is static the mapping will be generated once for all instances of the Class X.

A named Association LName between Classes X and Z is mapped to:

DeclaredLink(X, LName, Z).

At the implementation level, additional rules must be added to handle unnamed associations, and association directions like:

DeclaredLink(Z, LName, X)← DeclaredLink(X, LName, Z).

DeclaredLink(X, Z) ← DeclaredLink(X, LName, Z).

Class X inheritance from Class Z is mapped to:

IsOfType(O, Z)← IsOfType(O, X).

DeclaredAttribute(X, Attname, Attype)← DeclaredAttribute (Z, Attname, Attype).

The same applies to inherited association with the predicate DeclaredLink.

For the sake of simplicity and readability, we limit ourselves to main mappings, omitting some obviously similar mappings like those of rules to handle unnamed associations and association directions. Based on the aforementioned mapping of AC metamodels, the next step is to map the creation of instances of classes and instances of their association, taking into consideration the Clause attributes.

Hence, an instance X1 of a Class X having the attributes noted atti set to the values Vi is mapped to:

IsOfType(X1, X) and for every attribute atti Attribute(X1, atti, Vi).

We Check whether: DeclaredAttribute(X, att_i, Y_i), IsOfType(V_i ,Y_i).

And, for every Clause attribute, we map a new clause replacing "Self" by the current instance X1. When the Clause attribute is static the generation is made once per class.

An association linking two class instances X1 and X2 with eventually an association name Lname is mapped to Link(X1, Lname, X2) in addition to other FOL rules to handle eventual association direction and name. We also check whether the association is declared.

As illustrating example we specify the FOL mapping of kernel AC elements and the CW metamodel in terms of predicates as follows in Table 1 and Table 2.

Table 1. Excerpt of Kernel Elements Specification

```
/*Kernel elements declaration e.g. Subject and Object:  */
    IsType(Subject), IsType(Object).
/* Associations between elements must be declared, e.g.: */
DeclaredLink(Subject, Query), DeclaredLink(Object, Query), DeclaredLink(AccessMode, Query).
/* We also define reusable predicates like those used to identify query related elements, like the subject requesting access, the targeted object and the access mode:  */
QueryingSubject(S) ← IsOfType(S,Subject), IsOfType(Q,Query), Link(S,Q), CurrentQuery(Q).
/* Similar clauses are needed for QueriedObject and QueryAccessMode. */
/*Another clause for current query identification  */
CurrentQuery(Q) ← IsOfType(Q,Query), Attribute(Q, isCurrent, true).
```

4. An integration Metamodel of AC Metamodels

In a real context, AC decisions depend on more than one principle, for instance, observing the level of trust in the subject, the need to access an object in order to accomplish a task, and the risk of disclosure of a private or critical information. An AC decision generally depends on more than one AC metamodel, and a complete AC control specification must state how to consider multiple AC decisions resulting from multiple

AC metamodel instances. We recall that such policies and specifications have been called "hybrid".

Our integration approach consists of clustering the DecisionHandler instances of a hybrid AC policy in order to apply to them combining algorithms (ComAl) in a multistage way, as explained in the remainder of this section. A combining algorithm consists of a specification of how to conclude with only one AC decision as output, in response to a set of multiple AC decisions as input. This entails that a ComAl is a decision system.

Table 2. Excerpt of the CW Metamodel Mapping

/* CWdecisionHandler Type declaration */ isType(CWdecisionHandler).
/* CWdecisionHandler inheritance of DecisionHandler */
IsOfType(X, DecisionHandler) ← IsOfType(X, CWdecisionHandler).

/* ConflictClass and ConflictGroup declaration */
IsType(ConflictClass).
IsOfType(K, ConflictClass) ← IsOfType(K, ObjectsGroup).
IsType(ConflictGroup).

/* Associations declaration */
DeclaredLink(Subject, hasAccessed, ConflictClass).
DeclaredLink(ConflictGroup, composition, ConflictClass).
DeclaredLink(CWdecisionHandler, uses, ConflictGroup).
/* The association name "uses" in the latter predicate, reflects the fact that non kernel elements instances of an AC metamodel are related to the DecisionHandler instance.

Decision logic for issuing AC decision: ReturnedDecision is a Clause attribute of DecisionHandler. It is inherited by CWdecisionHandler, and it helps in specifying the returned decision by the instances of CWdecisionHandler. "Self" designates the current instance of the class CWdecisionHandler in a Clause attribute.
The decision will be Deny if the queriedObject belongs to a ConflictClass different from the one already accessed by the queryingSubject; both considered classes are associated to the same ConflictGroup instance. */

ReturnedDecision (Self, Deny) ←
Link(Self, uses, SomeConflictGroup),
Link(SomeConflictGroup, composition, Class1),
Link(SomeConflictGroup, composition, Class2),
Link(Self, uses,Class1), Link(Self, uses,Class2),
Link(QueriedObject, belongsTo,Class1),
Link(queryingSubject, hasAccessed, Class2),
¬Link(queriedObject,belongsTo,Class2),
QueryingSubject(queryingSubject),
QueriedObject(queriedObject).

/* The above clause can be read informally by saying that the current CWdecisionHandler instance (Self) will return a Deny decision if the following are all true:
-there is SomeConflictGroup associated to "Self";
-this SomeConflictGroup has two classes Class1 and Class2;
-Class1 and Class2 are associated to "Self";
- the queried object pertains to Class1;
-the querying subject has accessed Class2;
- the queried object does not belong to Class2

The latter clause must be generated for each CWdecisionHandler instance by replacing "Self" with the considered instance. */

To specify the integration of several AC metamodels of a hybrid AC policy, we propose an integration metamodel (IM) based on a tree structure of AC decision systems, called Ascending Decisions Tree (ADT). The nodes of the ADT are either DecisionHandler instances or nodes applying ComAls and called

ComAlNode(s). In an ADT, every leaf node is a DecisionHandler instance that issues its AC decision based on its metamodel decision logic. Whereas, a non-leaf node is a ComAlNode that issues its decision by applying its ComAl on the decisions of its direct children nodes. These children nodes could be themselves leaf nodes or non-leaf nodes.

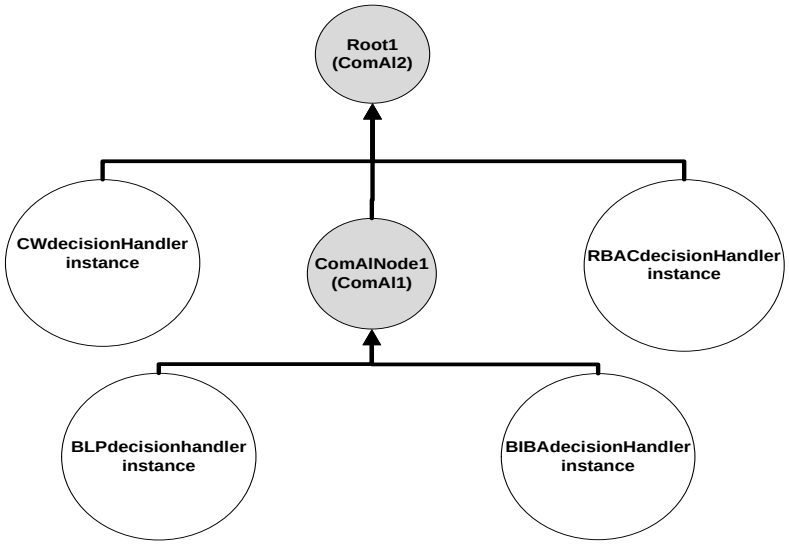

Fig. 8. An ADT integrating multiple AC metamodels instances.

The ADT has a unique root node and the decision this latter returns represents the decision of the whole tree carrying out the hybrid AC policy. Note that the root node concludes the application of several ComAls applied at many levels of the tree. Furthermore, we notice that any sub-tree of the complete ADT is itself an ADT and also a decision system. Thus the ADT uses a refinement and a divide and conquer approach. This approach allows structuring the specification of how to apply ComAls on separate sets of decision systems in order to always conclude with only one AC decision.

As an example of the application of the IM, we consider the hybrid AC policy illustrated in Fig. 8. According to the ADT representing the AC policy, we start by applying a ComAl, designated by ComAl1, to a BLPdecisionHandler instance and a BIBAdecisionHandler instance. Then another ComAl at the root node, designated by ComAl2, is applied to the decisions of the following elements: ComAlNode1, CWdecisionHandler instance, and RBACdecisionHandler instance. Thus, this ADT root node always returns a single AC decision based on four decisions issued by four DecisionHandler instances and the sequential application of two ComAl(s).

Furthermore, the IM comprehend the AC metamodels including their DecisionHandler elements in addition to the ComAlNode element. So the IM is sufficient to handle the specification of AC metamodel instances and their integration. IM is presented from different viewpoints: the main view that captures the structure of decision systems meta-elements, namely DecisionHandler and ComAlNode; and the views presented in Section 2 that encompass the kernel and the specific AC metamodels elements. Our IM is defined in a way that allows the description of a complex hybrid AC policy by structuring many DecisionHandler instances and ComAlNode instances in the ADT tree. Fig. 9 shows the IM main view with its structuring elements.

DecisionSystem: this element represents the concept of a decision system defined in Section 2 and whose instances are decision systems.

DecisionHandler is the already introduced element of AC metamodels whose instances are carrying the decisions of AC metamodels.

ComAlNode: this element represents any node of the tree that holds and applies a ComAl. Its association to DecisionSystem with a multiplicity of one or more means that every instance of ComAlNode is associated to one or more instances of DecisionSystem. A ComAlNode instance is itself a decision system since it returns a decision and consequently it is a specialization of DecisionSystem.

ACmetaModelElement: this element is introduced in Section 2 as a generalization of the element of the AC metamodels. It is shown in this view to indicate that the IM comprehend the remaining metamodels elements, in addition to DecisionHandler. It also recalls that a DecisionHandler depends on other elements of the same AC metamodel.

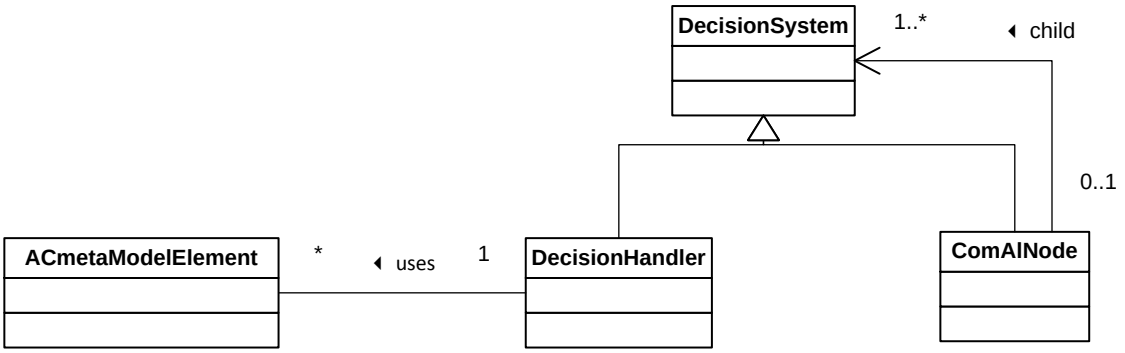

Fig. 9. The IM metamodel main view.

To summarize, the IM metamodel has the needed components to specify a hybrid AC policy as a tree of decision systems, with the capacity to specify each decision system node in terms of its specific modelling elements. Consequently, an AC policy can be specified visually as an ADT in which one can unfold or collapse:

- A node (as a decision system) to view or hide metamodel element instances or a ComAl specification;
- A sub-tree to view or hide a subset of the decision systems at a direct lower level.

5. Related Work

Barker's papers [9] and [10] are among the best known on the subject of AC modelling. Barker proposes a metamodel that identifies the AC primitives: subject, resource, and action augmented with an abstract element called "Category". Category can represent any AC concept which is not predefined as one of the above mentioned primitives; such concept should be specific to a certain AC metamodel. Category is a generic element that can represent the concepts of role, clearance, sensitivity, or any other AC concept. We mention also an extension to Barker metamodel [11] that adds to the metamodel an element allowing the specification of constraints on metamodel elements. Our work is compatible with Barker's metamodel or its extension; it rather allows the specification of each AC metamodel with a set of predefined elements including the underlying decision logic. Another aspect of comparison between our metamodel and Baker's work resides in the capacity to envision a structured use of combined algorithms (ComAls) to support hybrid AC policies.

We mention also other work based on logic to specify AC requirements, such as the contributions of Graven and al. [12], and Gelfond and Lobo [13]. These contributions have interesting features but they do not consider a modelling approach for depicting the AC concepts and relating them to language features. The SecPAL approach [14] proposes a general declarative framework for specifying AC requirements with an intuitive syntax similar to that of logic programming, but it neither relies on, nor proposes a modelling approach.

Besides, several attempts to extend UML to address AC design and implementation should be mentioned. Epstein and Sandhu [15] use UML to specify RBAC requirements. Shin and Ahn [16] propose techniques using the UML notation to achieve RBAC modelling. Doan et al. provide support for incorporating mandatory AC into UML diagrams [17].

In his UMLsec approach to model secure systems with a UML extension, Jurjens [18] makes room for specifying AC with formally specified annotations attached to UML elements. SecureUML is proposed by Basin [19] as a proof of concept for using a model driven approach to model secure systems; but SecureUML is limited to RBAC specification with no support for other policies.

In their work, Pavlich-Mariscal et al. [20] propose a UML extension in order to represent some AC concepts of RBAC, Bell LaPadula, and privileges delegation. The metamodel written in MOF is considered as a core metamodel that can be augmented, in future work, with any new AC metamodelling elements. They propose means to express only simple combining algorithms to deal with conflicts among multiple AC metamodels. In spite of the apparent similarity with their work, our research is more focused on the elaboration of a comprehensive integrating approach of AC metamodels and their logic mapping for properties verification, whereas Pavlich-Mariscal et al. focus is on the generation of the code enforcing an AC policy in the context of an application.

Jajodia *et al*. [21] propose a model encompassing hierarchies of roles, subjects and objects. The proposed model takes into account the history of granted accesses which can be used to express particular AC metamodels like CW. This model allows specifying conflicts between AC requirements, but with a limited expressivity based on precedence of denial or permission, or on an absolute priority to a specified authorization.

6. Conclusion

This paper has presented an integration metamodel IM as a comprehensive metamodel for hybrid AC policies. Although other metamodels were proposed in the literature, IM represents a realistic advance toward an AC specification language that allows formal verification of properties, promotes non-ambiguity, reduces complexity and supports readability and clarity. This is achieved by operating on four axes. First, the formal semantics carried by the FOL mapping provides the base for property verification using reasoning on FOL clauses. Second, the IM integration capability allows reducing the complexity of AC specification by separating the specification into well-structured and well-defined AC metamodel instances. Third, IM supports refinement and modularity with a visual representation based on an ADT tree structure that can be unfolded to selectively display progressively more detailed elements in AC metamodel instances or combining algorithms. Fourth, each AC metamodel allows the reutilization of its encapsulated AC decision logic and relevant elements.

We plan to develop an IM specification editing tool supporting the editing of IM instances with their synchronized corresponding textual specifications. This tool will also support syntax validation and property verification techniques.

Acknowledgment

This research was funded in part by the Natural Sciences and Engineering Research Council of Canada.

References

[1] Sandhu, R., Coyne, E., Feinstein, H., & Youman, C. (1996). Role-based access control models. *IEEE Computer*, *29(2)*, 38–47.

[2] Bell, D., & LaPadula, L. (March 1976). *Secure computer systems: unified exposition and multics*

interpretation. Mitre Corporation: Bedford, MA.
[3] Biba, K. (1977). *Integrity considerartions for secure compouter systems*. The Mitre Corporation.
[4] Brewer, D. F., & M. J. CNash, (1989). The Chinese Wall security policy. *Security and Privacy*, 206-214.
[5] OASIS. (2013). eXtensible Access Control Markup Language XACML Version 3.0. OASIS Standard.
[6] Damianou, N., Dulay, N., Lupu, E., & Sloman, M. (2001). The ponder specification language. *Policies for Distributed Systems and Networks*.
[7] Object Management Group. (2010). OMG Unified Modeling Language, Version 2.3, 2010. OMG Document Number: formal.
[8] Kleppe, A., Warmer, J., & Bast, W. (2002). M*DA explained, The Model Driven Architecture: Practice And Promise*. Addison-Wesley.
[9] Baker, S. (2012). Logical approaches to authorization policies. *Logic Programs, Norms and Action*. 349-373.
[10] Barker, S. (2009). The next 700 access control models or a unifying meta-model?. *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies* (pp. 187–196).
[11] Slimani, N., Khambhammettu, H., Adi, K., & Logrippo, L. (2011). UACML: Unified access control modeling language. *Proceedings of the 2011 4th IFIP International Conference on New Technologies, Mobility and Security* (pp. 1-8).
[12] Graven, R., Lobo, J., Ma, J., Russo, A., Lupu, E. C., & Bandara, A. K. (2009). Expressive policy analysis with enhanced system dynamicity. *Proceedings of the 4th international Symposium on Information Computer, and Commuication Security* (pp. 239–250).
[13] Gelfond, M., & Lobo, J. (2008). Authorization and obligation policies in dynamic systems. *Proceedings of the 24th International Conference on Logic Programming* (pp. 22–36).
[14] Becker, M. Y., Fournet, C., & Gordon, A. D. (2007). Design and semantics of a decentralized authorization language, 3–15.
[15] Epstein, P., & Sandhu, R. (1999). Towards a UML based approach to role engineering. *Proceedings of 4th ACM workshop on Role-based Access Control* (pp. 135–143).
[16] Shin, M., & Ahn, G. (2000). UML-based representation of role-based access control. *Proceedings of the 9th IEEE International Workshops on Enabling Technologies* (pp. 195–200).
[17] Doan, T., Demurjian, S., Ting, T. C., & Ketterl, A. (2004). MAC and UML for secure software design. *Proceedings of 2004 ACM Workshop on Formal Methods in Security Engineering* (pp. 75–85).
[18] Jurjens, J. (2001). Towards development of secure systems using UMLsec. *Proceedings of 4th International Conference on Fundamental Approaches to Software Engineering* (pp. 187–200).
[19] Basin, D., Doser, J., & Lodderstedt, T. (2006). Model driven security: From UML models to access control infrastructures. *ACM Transactions on Software Engineering and Methodology*, *15(1)*, 39–91.
[20] Pavlich, M. J., Demurjian, S., & Michel, L. (2010). A Framework of composable access control features: Preserving separation of access control concerns from models to code. *Computers and Security*, *29(3)*, 350–379.
[21] Jajodia, S., Samarati, P., Sapino, M., & Subrahmaninan, V. (2001). *Flexible Support for Multiple Access Control Policies*.

Jamal Abd-Ali was born in Beirut, Lebanon. He received his bachelor degree in civil engineering from Université Saint-Joseph (Beirut-Lebanon) in 1987. He obtained a master degree in computer science from Université du Québec en Outaouais (Quebec, Canada) in 2006. He is currently enrolled in the Ph.D program of computer science and technologies in the same university.

He worked as a project manager till 2002 then he has been an adjunct professor in the Computer Science and Engineering Department at the Université du Québec en Outaouais (Quebec, Canada). His research interests are in formal methods with application to access control, and model driven engineering.

Mr. Jamal is a member of Action TI in Quebec, the Project Management Institute and the Order of of Engineers and Architects of Beirut. He has been awarded a doctoral scholarship from the National Sciences and Engineering Research Council of Canada.

Karim El Guemhioui was born in Tetuan, Morocco. He received his degree in civil Engineering from École Mohammadia d'Ingénieurs (Morocco) in 1982. He earned a MS in computer science in 1992, and a PhD in computer science and engineering in 1997, both from the University of Connecticut, USA.

He worked several years as an engineer since 1982. He worked for a couple of years at the Centre de Recherche en Informatique de Montréal (Québec, Canada) before becoming in 1998 a faculty member at the Université du Québec en Outaouais (Québec, Canada). He is currently the head of the engineering programs. His research interests include software engineering of distributed systems, model driven development, and semantic web technologies.

Dr. Karim is a member of the Ordre des ingénieurs du Québec and a Fubright Alumnus.

Luigi Logrippo was born in Bologna, Italy. He received his degree in law from the University of Rome (Italy) in 1961. He obtained a MSc in computer science from the University of Manitoba (Winnipeg, Manitoba, Canada) in 1969, and then a PhD in computer science from the University of Waterloo (Waterloo, Ontario, Canada) in 1974.

He started a career in computing in 1961. From 1961 to 1967 he worked for Olivetti, General Electric, and Siemens. From 1973 he was with the University of Ottawa for 29 years, where he was Chair of the Computer Science Department for 7 years. He is an emeritus professor of this university. In 2002 he moved to the Université du Québec en Outaouais in Gatineau (Québec, Canada), Département d'informatique et ingénierie. His research interests are in formal methods with application to telecommunications, security and privacy software, and legal implications.

Dr. Luigi is a member of the ACM, and of technical committees of the International Telecommunications Union.