Metamodelling with Formal Semantics with Application to Access Control Specification

Jamal Abd-Ali, Karim El Guemhioui and Luigi Logrippo
Computer Science and Engineering Department, Université du Québec en Outaouais, Gatineau, QC, Canada

Keywords: Metamodelling, Access Control.

Abstract: The visual aspect of metamodelling languages is an efficient lever to deal with the complexity of specifying systems. In many application domains, these systems are generally characterized by the sensitivity and criticality of their contents, hence precision and formalism are essential goals. This paper considers the domain of access control specification languages and proposes a metamodelling paradigm with capabilities for specifying both semantics and structuring elements. We describe how to specify semantics of domain specific systems at the metamodel and model levels. The paradigm defines reusable rules allowing mapping the models, including their semantics, to first order logic programs. It represents a methodical approach to elaborate domain specific languages endowed with visual aspects and means of reasoning on formal specifications. The paradigm is applicable to a wide range of systems. We show in this paper its application in the area of decision systems.

1 INTRODUCTION

In specific domains, there are specific concepts that range from simple to rich, and even complex. Domain specific metamodels capture such concepts, and their instances (i.e. models) are interpreted in the light of these underlying metamodelling concepts. Formally specifying these concepts as part of a metamodel allows the creation of metamodel instances, augmented with formal specifications of concepts that may express their semantics.

In many domains, like access control (AC) and decision system specification, metamodelling has an unexploited potential to contribute to the formal specification of systems. However two shortcomings can be noticed: (1) the metamodels are often not formally specified, and therefore they cannot endow the specification languages they underlie with the formal semantics they do not have themselves; (2) When metamodels are formally specified, the semantics of their elements are often inexistent.

As a response to these two shortcomings, we propose a metamodelling pattern that supports the specification of metamodels with integrated formal semantics conveyable to their instances. Such metamodels are very important as domain specific languages that join the graphical aspect to the formal semantics. The formalism allows ambiguity free systems specifications that suit automated verification and processing which are vital in many critical application domains like AC, transportation, and intensive care monitoring.

As an illustration of this pattern we propose an elaboration of formally specified AC metamodels. Then we integrate these metamodels in a unique AC metamodel that, according to our metamodelling pattern, allows the definition of a textual language. This language draws its syntax and grammar from the structural elements of the metamodel, and draws its semantics from formally specified decision logic carried by these elements.

Although UML (OMG, 2010) 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. However, for figures that do not describe metamodels we use a free illustrative style.

In Section 2, we introduce the concept of AC metamodelling through examples of 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.

354 Abd-Ali J., El Guemhioui K. and Logrippo L..
Metamodelling with Formal Semantics with Application to Access Control Specification.
DOI: 10.5220/0005272903540362
In *Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development* (MODELSWARD-2015), pages 354-362
ISBN: 978-989-758-083-3
Copyright © 2015 SCITEPRESS (Science and Technology Publications, Lda.)

2 ACCESS CONTROL METAMODELS

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. For example, we can mention the principle of granting access privileges based on the subject's role (i.e., position or mission) in an organization. This principle is embodied in the well-known Role Based Access Control model (RBAC) (Sandhu et al., 1996). 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) (Bell & LaPadula, 1976), BIBA (BIBA, 1977), and Chinese Wall (CW) (Brewer, 1989). Although these models are seldom used in their 'pure' form, they are at the basis of many practical models.

An AC model spares us the effort of specifying repeatedly the underlying AC logic; this latter 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 (OASIS, 2013), Ponder (Damianou et al., 2001)) are self-contained and don't take advantage of any formally specified model.

Furthermore, since AC specification can enforce several AC principles 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 enforcing concurrently multiple AC models.

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 Figure 1, Sam is an instance of Subject, and CEO is an instance of Role. According to the metamodelling paradigm (Kleppe, Warmer & Bast, 2002), 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.

Figure 1 shows an example of a metamodel and a model, instance of that metamodel. This very simple metamodel represents the principle that a subject must be associated to a role. It consists of the 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.

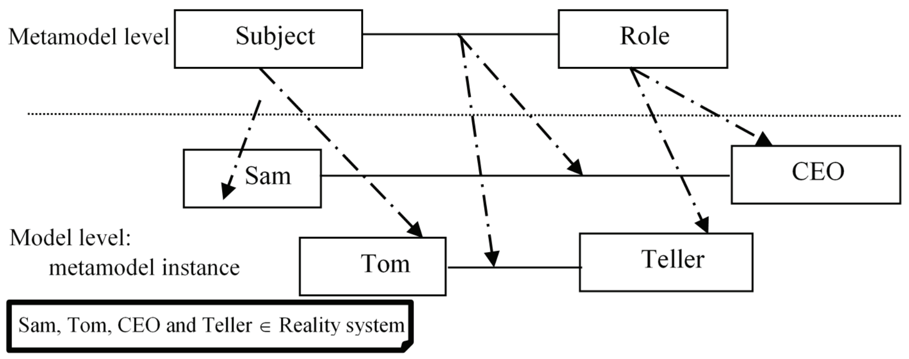

Figure 1: Metamodelling levels.

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 this Section; 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 Figure 2.

Figure 2The 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. This explains the association “uses” between DecisionHandler and ACmetamModel. 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 by the association “uses” in Figure 2.

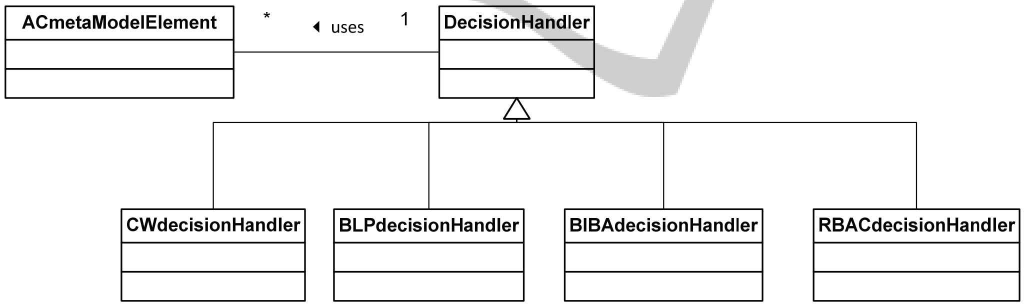

Figure 2: DecisionHandler specialisations in AC metamodels.

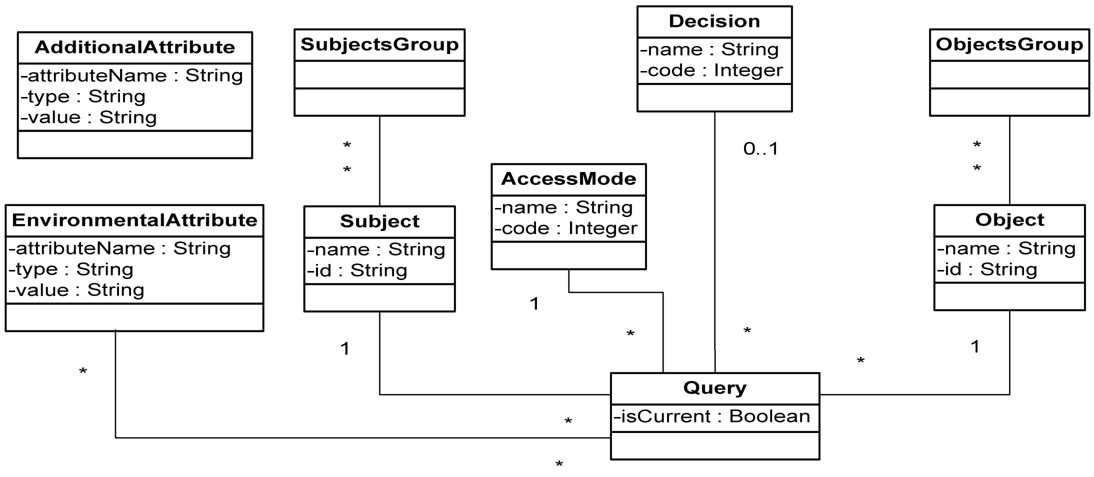

Figure 3: Kernel AC elements.

In the remainder of this section we present the elements shared by the following four AC metamodels: Chinese Wall, Bell LaPadula, BIBA, and RBAC, but for the sake of space limitation we only present the Chinese Wall metamodel. We adopt the following convention: metamodel elements start with an uppercase letter, while their instances start in lowercase. Thus subject and object represent respectively instances of the elements Subject and Object.

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 Figure 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 to by the access request is called the targeted object.

Query associations to respectively Object, Subject, and AccessMode (see Figure 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 Figure 3), since an AC decision may depend on the state of the environment.

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, after a subject has accessed some objects, it must be prevented 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:

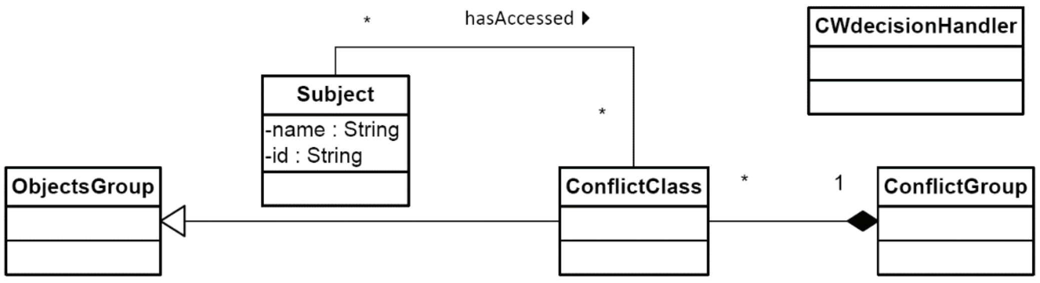

Figure 4: Chinese Wall metamodel.

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 $k \neq i$. In other words ((AccessedBy-s $\cap$ (C1 $\cup$ C2 …$\cup$ Cn)) $\subseteq$ Cx) must remain true for any subject s, where AccessedBy-s designates the set of objects s has accessed, and Cx designates a single ConflictClass instance which belongs to the conflict group CG.

The CW metamodel shown in Figure 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, although it should be blocked if it can lead to unwanted information propagation. Subject and ObjectsGroup are already introduced as kernel AC elements. CWdecisionHandler is the specialization of DecisionHandler for CW.

3 FORMAL SPECIFICATION OF AC METAMODELS AND DECISION LOGIC

We consider the metamodels whose representation relies on a subset of UML limited to classes, attributes and classes associations. Such modelling representation is neither complete nor ambiguity-free. Indeed, a metamodel is generally endowed with elements representing informally specified semantics of a domain. We propose percolating these semantics from metamodel elements to their instances by attaching formal semantics to metamodel elements. Our approach aims to express both modelling elements and their semantics in the same formal language that we call target language.

As a target language, we have adopted the First order logic since it is known to be suitable for relating individuals to their types, for specifying relationships between individuals, and for expressing rules like decision logic.

That said, we propose a metamodelling pattern based on two complementary folds: the first maps the modelling elements to First Order Logic (FOL) as a target language, the second attaches to each modelling element a special dedicated attribute that holds FOL clauses expressing semantics. Such attributes are called "Clause attributes" and they carry the semantics at both metamodel and model levels.

For every element of the considered UML subset and for every instantiation operation of this element, we use a dedicated predicate to formally specify it.

Furthermore, the Clause attribute type is called "Clause", and an instance of it 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:

```
          Human(X)  ← Walk(X), Talk(X).  (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 map this subset of UML to FOL. 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 delimit 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 respectivie types Yi, is mapped to `IsType(X)`and `Declared Attribute (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).
   DeclaredLinK(W, LName, X) ←
DeclaredLink(W, LName, Z).
```

This mapping reflects that: (1) every instance of X is instance of Z; (2) Z attributes and associations are also inherited by X. 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, atti, Yi), IsOfType(Vi ,Yi).`

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.

That said, we can now apply our metamodelling pattern to our AC metamodels, in order to specify their semantics consisting of AC decision logic. Every element of an AC metamodel is mapped to an FOL individual or predicates truth values relating individuals. At this stage, the semantic of the AC metamodel is not yet specified neither in the UML representation nor in the elements mapping. Hence, we use the Clause attribute of DecisionHandler, and we assign it a clause representing the decision logic of the considered AC metamodel. The clause is expressed in terms of predicates applied to the individuals resulting from the mappings of the AC metamodel elements. Thus, the semantic of the metamodel is formally specified as an AC decision logic.

In addition, every instance of an AC metamodel has an instance of DecisionHandler whose Clause attribute is a copy of the DecisionHandler Clause attribute in which "Self" is replaced by the individual representing itself as the current instance of DecisionHandler.

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 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 hybrid AC policies in order to apply to them combining algorithms (ComAl) in a multistage way, as explained in the remainder of this section. A combining algorithm specifies how to generate only one AC decision as output, in response to a set of multiple AC decisions as input. Thus, a ComAl is a decision system.

To specify the integration of several AC metamodels of a hybrid AC policy, we propose an integration metamodel (IM) that encompasses in addition of the already explained metamodelling elements the ComAlNode and DecisionSystem elements as illustrated in Figure 5.

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

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

ComAlNode: this represents an element whose instance holds a ComAl specification as Clause attribute. It applies this ComAl to the decisions issued by its associated DecisionSystem with the association named child. This means that every instance of ComAlNode can issue a decision resulting from applying ComAl to one or more instances of DecisionSystem. This explain why a ComAlNode instance is itself a decision system, and consequently it is a specialization of DecisionSystem.

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 a tree we call Ascending Decision Tree (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 decision systems at a direct lower level

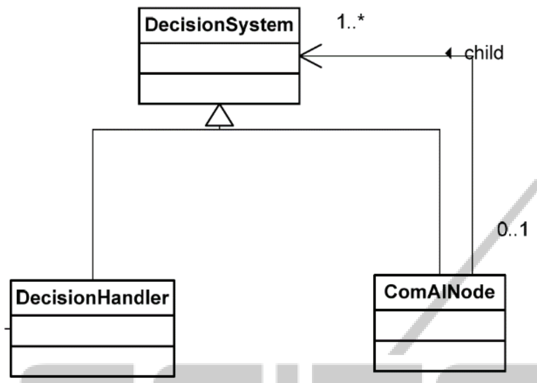

Figure 5: The IM metamodel main view.

5 RELATED WORK

Barker's papers (Barker, 2009), (Barker, 2012) 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's metamodel (Slimani & al., 2011) 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. (Graven et al., 2009), and Gelfond and Lobo (Gelfond & Lobo, 2008). 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 (Becker, 2007) 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.

Several attempts to extend UML to address AC design and implementation should be mentioned also. Epstein and Sandhu (Epstein & Sandhu, 1999) use UML to specify RBAC requirements. Shin and Ahn (Shin & Ahn, 2000) propose techniques using the UML notation to achieve RBAC modelling. Doan et al. provide support for incorporating mandatory AC into UML diagrams (Doan et al., 2004).

In his UMLsec approach to model secure systems with a UML extension, Jurjens (Jurjens, 2001) makes room for specifying AC with formally specified annotations attached to UML elements. SecureUML is proposed by Basin (Basin, Doser & Lodderstedt, 2006) 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.

(Pavlich-Mariscal, Demurjian & Michel, 2010) proposes 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 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 property verification, whereas Pavlich-Mariscal et al. focus on the generation of the code enforcing an AC policy in the context of an application.

Jajodia et al. (Jajodia et al., 2001) 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 CONCLUSIONS

Through metamodelling for the access control specifications domain, this paper has presented a metamodelling pattern capable of supporting the semantics of the modelled systems. The adopted mapping between modelling elements and FOL is by defining a two way mapping, and is domain independent. The proposed pattern promotes reutilization by offering a technique allowing to convey logic rules, like decision logic, from metamodel elements to their instances. This allows upgrading from illustrative metamodels to formal specification languages with semantic expressiveness capabilities and visual representation.

Applying this pattern, we have elaborated an access control 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 generation of IM instances with their corresponding synchronized textual specifications. This tool will also support syntax validation and property verification techniques.

ACKNOWLEDGEMENTS

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

REFERENCES

Barker, S. (2012) Logical Approaches to Authorization Policies. In: Artikis, A., Craven, R., Çiçekli, N. K., Sadighi, B., Stathis, K.(eds.) Logic Programs, Norms and Action. LNCS, vol. 7360, pp. 349-373. Berlin Heidelberg: Springer.

Barker, S. (2009) The next 700 access control models or a unifying meta-model?. *In: Proceedings of 14th ACM Symposium on Access Control Models and Technologies (SACMAT'09).* pp. 187–196.

Basin, D., Doser, J., Lodderstedt, T. (2006) Model driven security: From UML models to access control infrastructures. *ACM Transactions on Software Engineering and Methodology.* Vol.15, pt. 1, pp. 39–91.

Becker, M. Y., Fournet, C. & Gordon, A. D. (2007) Design and semantics of a decentralized authorization language. In: *CSF.* pp. 3–15.

Bell, D. & LaPadula, L. (1976) Secure Computer Systems: Unified Exposition and Multics Interpretation. *Mitre Corporation:* Bedford, MA. (March 1976).

Biba, K. (1977) Integrity Considerartions for Secure Compouter Systems. *The Mitre Corporation.* (April 1977).

Brewer, D. F.C & Nash, M. J. (1989) The Chinese Wall security policy. In: *Security and Privacy 1989, Oakland, CA.* pp. 206-214.

Damianou, N., Dulay, N., Lupu, E. & Sloman, M. (2001) The Ponder specification language. In: *Workshop on Policies for Distributed Systems and Networks, Jan 2001.*

Doan, T., Demurjian, S., Ting, T.C. & Ketterl, A. (2004) MAC and UML for secure software design. In: *Proceedings of 2004 ACM workshop on Formal methods in security engineering (FMSE'04) 2004.* pp. 75–85.

Epstein, P. & Sandhu, R. 1999 Towards a UML based approach to role engineering. In*: Proceedings of 4th ACM workshop on Role-based Access Control (RBAC'99) 1999.* pp. 135–143.

Gelfond, M. & Lobo, J. (2008) Authorization and Obligation Policies in Dynamic Systems. In: Garcia de la Banda, M., Pontelli, E. eds. *ICLP 2008*. LNCS, vol. 5366, pp. 22–36. Heidelberg: Springer.

Graven, R., Lobo, J., Ma, J., Russo, A., Lupu, E.C. & Bandara, A.K. (2009) Expressive policy analysis with enhanced system dynamicity. In: *ASIACCS proceedings of the 4th international Symposium on Information Computer, and Commuication Security 2009.* pp. 239–250. New York: ACM.

Jajodia, S., Samarati, P., Sapino, M. & Subrahmaninan, V. (2001) Flexible support for multiple access control policies. Vol. 26, pt.2, pp.214–260 ACM TODS.

Jurjens, J. (2001) Towards development of secure systems using UMLsec. In: *Hussmann, H. (eds.) Proceedings of 4th International Conference on Fundamental Approaches to Software Engineering (FASE/ ETAPS'01) 2001.* volume of LNCS, vol. 2029, pp.187–200. Heidelberg: Spring.

Kleppe, A., Warmer, J. & Bast, W. (2002). *MDA Explained, The Model Driven Architecture: Practice And Promise.* Addison-Wesley.

OASIS (2013) eXtensible Access Control Markup Language XACML version 3.0. OASIS standard.

Object Management Group, (2010) Unified Modeling Language, version 2.3. OMG Document Number: formal/2010-05-03.

Pavlich-Mariscal, J., Demurjian, S. & Michel, L. (2010) A framework of composable access control features: Preserving separation of access control concerns from models to code. *Computers & Security*. Vol. 29, pt.3, pp.50–379.

Sandhu, R., Coyne, E., Feinstein, H. & Youman, C. (1996) Role-based access control models. *Computer*. Vol. 29, pt.2, pp.38–47.

Shin, M., &Ahn, G. (2000) UML-based representation of role-based access control. In: *Proceedings of 9th IEEE International Workshops on Enabling Technologies (WETICE'00) (2000)*. pp. 195–200.

Slimani, N., Khambhammettu, H., Adi, K. & Logrippo, L. (2011) UACML: Unified Access Control Modeling Language. In: *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference*. pp. 1-8. Paris: IEEE press.