ACCESS CONTROL MODELS FOR BUSINESS PROCESSES

Vahid R. Karimi and Donald D. Cowan
Cheriton School of Computer Science,University of Waterloo, Waterloo, Ontario, Canada

Keywords: Access control models and policies, Business processes, Patterns, RBAC, REA.

Abstract: A business model describes certain operations of an enterprise, and an important aspect of business operations deals with the specification of access control policies, which are used to constrain the business operations by adding what should, could, or must be. We describe the use of patterns for presenting access control models and policies. Our goal is to specify access control policies such that they are based on access control models and have the capability of policy languages, thereby making the foundational blocks of these policies and operational models identical. Thus, the integration of these policies into operational models is straightforward. To show our approach, we use Role-based Access Control (RBAC), a well-known access control model, and also select a business process model whose foundational building blocks are *Resources*, *Events*, and *Agents* (REA). We make three main contributions: 1) the use of the same foundational building blocks and similar models to describe business processes and access control models, 2) access control policies that are based on an access control model, and 3) access control policies that are rule-based and akin to policy languages. As a result, such models are more understandable, and their future modifications are more straightforward.

1 INTRODUCTION

The separation of access control policies from applications has several benefits: it makes possible the change of access control policies independent of applications, enables analysis of these policies separately, and provides the ability to document them clearly. Despite these benefits, this approach creates a problem: integrating these policies into applications (Ray et al., 2004). Our presentation is intended to provide some guidelines for this integration but not necessarily for their inclusion in applications.

The Resources, Events, Agents (REA) (Geerts and McCarthy, 2006) approach for describing business processes was initially introduced by McCarthy and has been extended over the years. REA has been chosen to represent business processes for two reasons: REA includes rules for constructing business models, making this process more straightforward than other representational notations. Moreover, REA describes both static and dynamic aspects of business processes by using UML class and activity diagrams. Both types of diagrams are important, providing a more comprehensive approach than others that mainly use a variation of activity diagrams. (A dynamic REA model has not been discussed in this paper.) *Exchange* and *conversion* models represent two broad categories of REA business processes. Any number of exchanges and conversions can be combined to build a larger model. Sales and loans exemplify exchange models, whereas production of new items and repair of existing items represent conversion models. A basic REA exchange model describes *agents* participating in *events* at which *resources* are exchanged. Figure 1(a) shows an example in which an employee *receive*s a client's cheque and *provide*s an account modification; similarly, the client *provide*s a cheque and *receive*s an account modification.

In our work, the word "pattern" is used to mean "an idea that has been useful in one practical context and will probably be useful in others," or similarly, "groups of concepts that represent a common construction in business modeling" (Fowler, 1997). For instance, Figure 1(b) shows a policy pattern and can be described as a rule-based statement: e.g., if the *kind* of *account* is savings with a *foreignCurrency* and the *kind* of modification is a withdrawal, then a *maximumAmount* for a withdrawal is permitted. Therefore, the rule-based nature of these types of specifications makes them akin to policy languages. Figure 1(b) can subsequently be combined with Figure 1(a) in which resource (account) and event (accountModifying), their common entities, are merged together. **Note:** The class *policy* in Figure 1(b) and in other figures of this paper is included to emphasize that policies exist. Only one policy class to de-

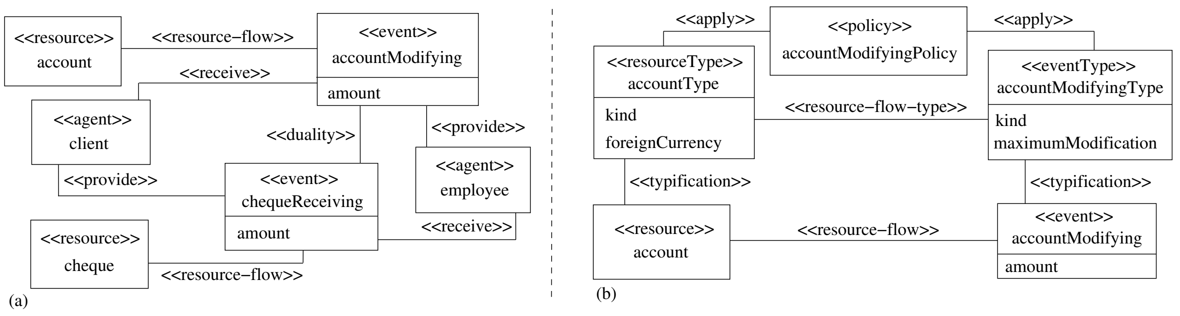

Figure 1: (a) A REA exchange model, (b) a policy.

fine access control policies exists for each application. For instance, for Figure 1(b), policies can be defined by the association between accountType and accountModifyingType classes.

We use patterns as the basis for representing an access control model. The benefits of patterns are commonly acknowledged in software engineering. For instance, patterns can enable us to reuse and prevent us from reinventing the same solutions over and over; in addition, one can take advantage of the guidelines on how and when to use patterns (Blaha and Rumbaugh, 2005). As an additional contribution, we use the REA model and present an access control model (i.e., RBAC as an example of a well-known access control model) for it. This capability is possible because all the components needed to express an access control model exist in our approach. To clarify this statement, we would like to point to a description from Ferraiolo et al. (Ferraiolo et al., 2007): almost any access control model can be described by the elements of *users*, *subjects*, *objects*, *operations*, *permissions*, and the relationships between these notions. A comparison between these elements and the building blocks of our model follows: *users*, *objects*, *operations*, and *permissions* of an access control model correspond to the ontological concepts of *agents*, *resources*, *events*, and attributes of *agents* or some combinations of *agents*, *events*, and *resources*, respectively. The *subject* also corresponds to an application built by using such ontological building blocks. Furthermore, the terms *agents*, *resources*, *events* appear to be more intuitive than *users*, *objects*, and *operations*, at least for describing business processes.

Some researchers (Finin et al., 2008) describe a synergy between efforts in the development of access control models (e.g., RBAC) and policy languages (e.g., eXtensible Access Control Markup Language (XACML)) as valuable. Our presentation has the same goal because policies are based on an access control model, and in addition, the rule-based nature of these policies resembles the declarative approach of policy languages.

The question of whether an access control metamodel for specifying access control policies exists has been asked (Ferraiolo and Atluri, 2008). Our presentation describing both access control models and business models with the same foundational building blocks is directed toward solving this problem.

2 BACKGROUND

Geerts and McCarthy (Geerts and McCarthy, 2006) introduced policy-level specification for REA models. Their general policy patterns, shown in Figure 2, are illustrated using a plane and flight scenario in Figure 3.

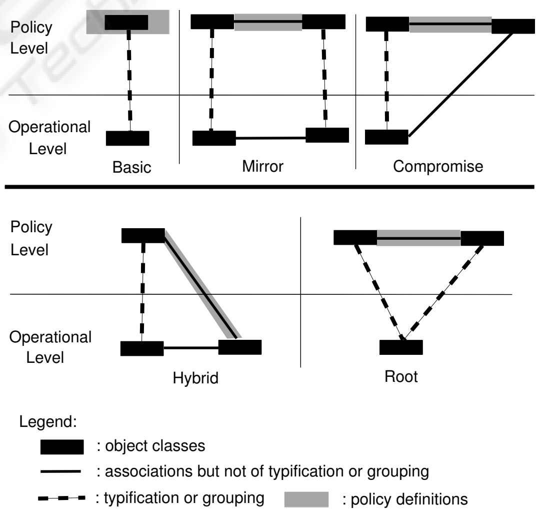

Figure 2: Patterns for policy-level specifications.

Figures 2 and 3 can be described in the following manner (Geerts and McCarthy, 2006): The *Basic* pattern includes flight, flightType, and a single typification association between them. A policy is defined by a flightType attribute: based on a scheduled departure of a flightType, a policy can state that a flight should take off at a certain time (which can be different from the flight's actual departure time). In these two figures, a typification relationship denotes an "is a-kind-of" association, whereas a grouping association represents an "is a-member-of" association.

The *Mirror* pattern is presented among planeType, flightType, plane, and flight (i.e., two typification or grouping associations and two other associations). A policy can express the type of plane to be used for the type of flight. In the *Compromise* pattern, a variation of Mirror, one typification or grouping is compromised such that it consists of one typification or grouping and two other associations–one of which is at the policy level. Compromise includes routeType, cancelationType, and flight; and a cancelationType can be defined per routeType. The *Hybrid* pattern, another variation of Mirror, includes plane, fleet, and serviceOperator and consists of one typification or grouping and two other associations–none of which is at the policy level. This pattern is used when one class has a small number of instances on which the policy is validated (e.g., a list of people who can provide service to a fleet).

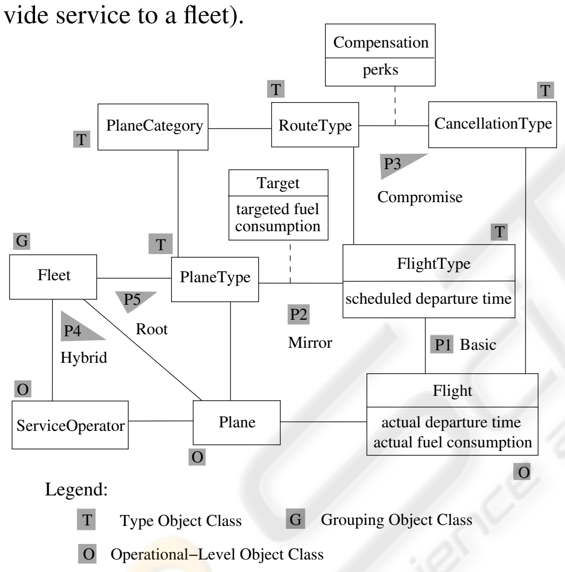

Figure 3: A flight example for policy-level specifications.

Finally, the *Root* pattern is defined among fleet, planeType, and fleet and applies to cases in which one operational-level object participates in more than one typification or grouping (e.g., the typification of plane and planeType and the grouping of plane and fleet). This pattern consists of two typification or grouping associations and another association at the policy level (e.g., a plane should be of a specific planeType to be a member of a fleet).

3 OUR APPROACH

We adapt the general policy patterns. By this adaptation, we represent both access control policies and an access control model (i.e., RBAC). Our adaptation is based on an examination of several access control policy expressions and a review of the RBAC model. Figure 4 (Ferraiolo et al., 2001) shows RBAC.

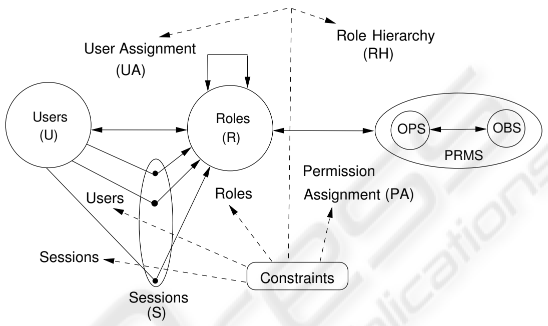

Figure 4: RBAC with permissions as operations on objects.

We describe modeling every element of Figure 4: a) roles and user assignments and, in addition, the way roles are assigned to users, which is not described by RBAC, b) permissions and permission assignments (RBAC does not describe permission assignment to roles.), c) role hierarchies, and d) constraint descriptions of users, roles, permissions, role hierarchies, and static and dynamic separation of duties.

A description of our use of patterns to model all the cases above follows shortly. We adapt the general policy patterns, shown in Figures 2 and 3, as a starting point because these patterns differ from ours for the following reasons:

a) We extend these patterns in various ways.
b) The building blocks of described patterns are classes and are not identified as the foundational building blocks of REA (i.e., resources, agents, events). We show specifically the foundational building blocks and their types needed for building access control policies: one of our intentions is to demonstrate that access control policies and business processes can be built by the same building blocks.
c) Our adapted patterns represent a well-known access control model and, in addition, describe access control policies when they are combined. No specific rules exist to structure policy definitions using the patterns of Section 2, but certain combinations of our adopted patterns represent access control policy definitions.
d) We explicitly illustrate the modeling of roles and user assignments, permissions, role hierarchies, and their constraints. This modeling is important

because not only does RBAC comprise these components but also any access control model consists of some of these elements.

3.1 Metadata and Powertype

In our approach and patterns, we use the concept of type, such as an event type or a resource type, in the sense of metadata (i.e., the data about data) or a metaclass (i.e., a class that describes another class) to be consistent with the current UML modeling approach. Figure 5 (Blaha and Rumbaugh, 2005) shows a CarModel and a PhysicalCar.

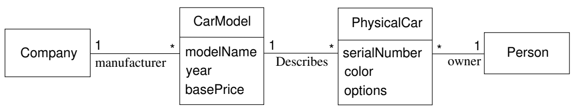

Figure 5: Metadata and data.

Figure 5 represents a CarModel class as metadata or a metaclass in relation to the PhysicalCar class because a CarModel, such as a 2005 Toyota Camry, describes many PhysicalCars with various serialNumbers and colors (i.e., a car model is a car type.) Data and metadata are relative to each other; otherwise, “models are inherently metadata, since they describe the things being modeled (rather than being the things)” (Blaha and Rumbaugh, 2005).

In our upcoming presentation of patterns, we do not differentiate between policy and operational levels, for the same reason that models are intrinsically metadata, and such differentiation between policy and operational levels is not necessary. To be consistent with the current UML modeling approach, we use an event type and a resource type as metaclasses relative to their resource and event counterparts as described above. Furthermore, our use of type and typification is also related to the concept of a power type.

Type and Power Type. Power type is an advanced technique and is mainly used in the analysis phase (OMG, 2009). A power type describes a class whose instances are subclasses of another class; therefore, power types are metaclasses with an extra condition applied to their instances such that the instances are also subclasses of a superclass (Martin and Odell, 1998). Figure 6 (OMG, 2009) shows an example.

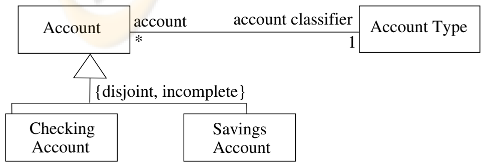

Figure 6: Types and Power Types.

3.2 A Running Example

Figure 7 shows a small running example that is related to a banking application and is described by Chandramouli (Chandramouli, 2000).

Convention. For the rest of this paper for readability purposes, we do not use capital letters or underscores to join words in class names or their attributes; therefore, “loan officer” and “day of week” are used instead of loanOfficer and dayOfWeek. The same convention holds when tables are presented.

4 PATTERN USE TO REPRESENT ACCESS CONTROL MODELS

Groups are usually a collection of users but do not usually represent a collection of permissions (Sandhu et al., 1996); therefore, a role and a group of users are not considered identical concepts. In general, policies apply to group and type entities rather than an individual entity. A group can have only one member, and a policy can apply to a group with only one member.

The patterns presented here are not complex and are therefore consistent with the general presentation of patterns in the literature, e.g., (Fowler, 1997). Similar to the pattern literature, we describe patterns uniformly. This description includes “context,” “problem,” and “solution”–as three main elements–based on a common definition that a pattern describes a solution to a problem that happens over and over in a context. The “resulting context” is adopted from Hruby’s form (Hruby, 2006). The pattern description consists of the following elements:

Name. A short descriptive name for a pattern

Context. A description of the situation in which a pattern applies

Problem. A brief explanation of the problem that a pattern attempts to solve

Solution. An explanation of the way a pattern solves a described problem; we intend to describe the solution in enough detail to be clear, but also to be general enough to allow the solution to apply broadly.

Example. An example of the solution

Alternative Example. A secondary example that may not be as common as the main example

Resulting Context. A clarification and the consequences of the solution in a broader aspect

4.1 Modeling Roles & User Assignments

Name. *Role Modeling and User Assignments* Pattern

Context. Several research papers describe various approaches for discovering roles in an organization, but

Banking Case Study (Chandramouli, 2000)

The banking application is used by tellers, customer service reps, loan officers, accountants, and accounting managers.

Policies. The existing policies are as follows:

P1) A teller can modify customer deposit accounts.
P2) A customer service rep can create and delete customer deposit accounts and also has a teller's permissions.
P3) A loan officer can create and modify loan accounts.
P4) An accountant can create general ledger reports.
P5) An accounting manager can modify ledger posting rules and also has the permissions of an accountant.

Hierarchical Role Relationships.

H1) A customer service rep role ranks higher than a teller role.
H2) An accounting manager role ranks higher than an accountant role.
H3) Customer service rep, loan officer, and accounting manager roles rank at the same level.
H4) A branch manager ranks the highest.

Static Separation of Duties. A single user cannot hold the following pair of roles:

1) customer service rep and accounting manager 2) loan officer and accounting manager
3) teller and accountant 4) teller and loan officer 5) accountant and loan officer

Dynamic Separation of Duties. One individual cannot hold the following two roles at the same time: customer service rep and loan officer

Figure 7: A banking example.

our intention is not to determine possible roles. We assume that these roles are already known and want to describe these roles using various existing entities and their attributes.

Problem. How can we model the roles of users? The concept of roles in organizations is significant because activities and tasks are associated with roles. How are these roles assigned to individuals? In addition, how can different roles of an individual, if they arise, be modeled?

Solution. The *role modeling and user assignments* pattern (Figure 8) extends *Basic* and *Root* patterns (Section 2) to make possible the modeling of roles and user assignments to a certain level of detail. Based on our adaptation, the entities of this pattern can only be agents because our goal is to represent roles. Not only the attributes of types (i.e., agent types), as in the *Basic* pattern, but also the attributes of an agent can determine an agent's role in some detail.

In addition, the *role modeling and user assignments* pattern also uses a similar approach to the *Root* pattern (Figures 2 and 3) by including the "grouping" relationship to enable an agent to hold more than one role, depending on affiliation with different groups. (REA models a group, such as an organization unit or a team, by the "grouping" relationship.) By this inclusion and extension, an agent can participate in a typification and grouping relationship, which is similar to the *Root* pattern. The association between an agent type and agent grouping that exists in a *Root* pattern is also present in the UML class diagram of Figure 8(a); as a result, roles can change in relation to various groups. In addition, using various combinations of agent attributes, we will be able to describe certain details of roles and their functions. REA Grouping and aggregation are related concepts. One possible common approach for assigning roles to users is based on the value of user attributes. An extension to this approach can include describing a role with a certain level of detail based on the value of attributes, e.g., a teller's role is permitted to be active between 9 and 5.

Example. Figure 8(b) shows an example in which attributes are present ("day of week performed" attribute; e.g., Monday to Friday, or "daily start time" and "daily end time" attributes.)

Table 1 includes agent, agent attribute, and role columns and represents possible policy content, as shown in Figure 8(b). In this table, one or two attributes of an agent determine the agent's role or the existing constraints on the role. Several attributes can contribute to the determination of roles by a similar approach (to simplify, only one or two attributes are chosen.) The banking example (Figure 7) does not contain the roles shown in rows two and three of this table, but these roles exist to explain that a richer description of policies is possible with just the use of attributes. The second row of the table illustrates a situation in which, based on the status and responsibility attributes, a temporary teller role is recognized. Finally, the next-to-the last row illustrates the possibility of describing temporal aspects such that a teller role is defined to be valid only for a permitted time period (e.g., Monday to Friday and 9:00 to 5:00). The introduction of temporal aspects is very important be-

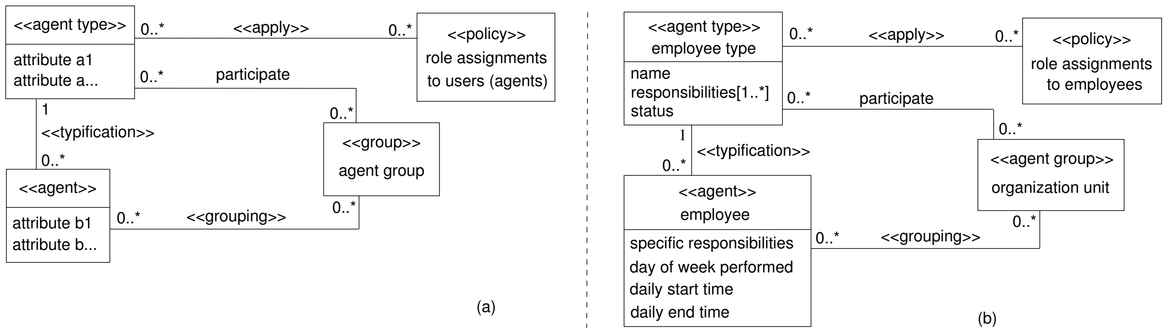

Figure 8: (a) Pattern for modeling roles and user assignments, (b) an example.

Table 1: Entities and attributes of the role modeling and user assignments pattern in a table format.

pattern name	user (agent)	agent type attribute	agent group	role
role modeling	employee	responsibilities	-	teller
and user	employee	status, responsibilities	-	temporary teller
assignment	employee	day of week performed	-	teller (when)
pattern	employee	-	branch	teller

cause they enable the creation of dynamic and rich models. The last row shows bank branch as an agent group.

Resulting Context. More expressive models such as temporal RBAC (TRBAC) (Bertino et al., 2000) exist; with TRBAC, temporal aspects of roles are specified such that some roles are active at certain times (e.g., as in the previous teller example). TRBAC achieves the expression of temporal aspects of roles by their activation and deactivation. It may be the case, as we have described previously, that some features of these extended models can be expressed by the original RBAC model and by the use of temporal attributes whenever they exist. On this note, we have also assumed that certain attributes such as “responsibilities” or “the time of an event” exist, an assumption that we believe is quite realistic.

Alternative Example. An alternative representation can use the power type concept.

4.2 Modeling Permissions

Name. *Permission Modeling* Pattern

Context. Permission is one of the main components of any access control model and constitutes an element of an access control policy. Describing the processes of an organization not only includes an explanation of what needs to be performed but also includes permissions and restrictions in performing these activities. If an authorization is described using the same entities and models as those used for business processes, then the integration of these authorizations into business processes will be more straightforward.

Problem. How can role permission be modeled? In general, how can permission as an element of an access control model be described?

Solution. Permissions can be represented as operations on objects, as is shown by the right side of Figure 4. Operations translate to REA events, and objects are equivalent to REA resources shown in Figure 9(a). Therefore, the *Mirror* pattern presents a viable choice for modeling this setting. The following brief description of events and operations can be useful and justify the mapping of events to operations.

Events and Operations: Despite some differences, these two terms are essentially two different views of the same thing; an event includes an important or interesting change of state whereas an operation is the agent of this change (Martin and Odell, 1998). In other words, the operation view considers the mechanism for this change, and the event view is concerned about the result of the operation (Martin and Odell, 1998). For instance, *verify* (order) represents an operation, whereas (order) *verified* describes an event occurrence.

The *permission modeling* pattern clarifies and modifies the *Mirror* pattern for modeling permissions. The clarification is based on the identification of event, event type, resource, and resource type as entities of this pattern, as shown in Figure 9(a). Operations (OPS), objects (OBS), two typification associations, and two other associations are also presented in this figure. An extension to this pattern includes the use of attributes of operations (i.e., events), objects (i.e., resources), or both. Therefore, a permission can

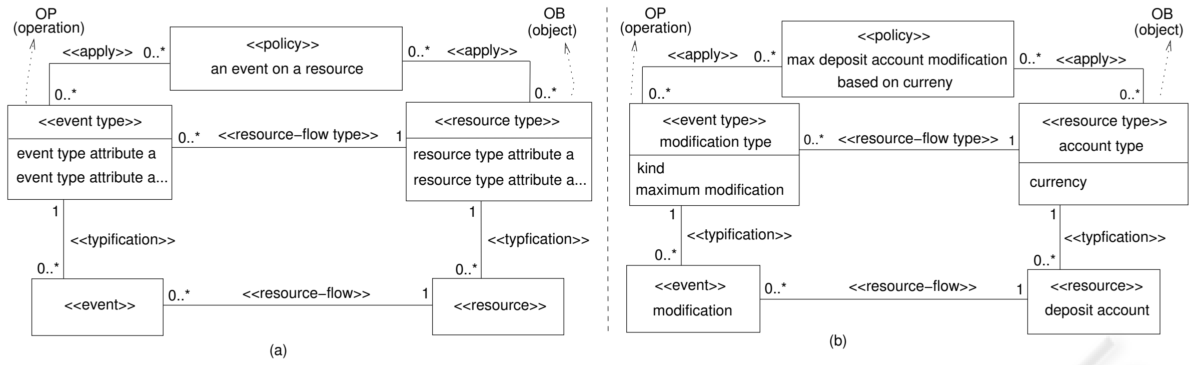

Figure 9: (a) Pattern for modeling permissions, (b) an example.

Table 2: Entities and attributes of permission modeling pattern.

pattern name	event type	event type attribute	resource type	resource	resource type attribute
permission	modification	-	account	deposit account	-
modeling	create	-	account	loan account	-
pattern	modification	maximum	account modification	deposit account	currency

specify the maximum amount of modification to a deposit account if foreign currency is involved. The use of attributes with this pattern describes constraints on operations and objects.

Example. Figure 9(b) shows a permission and also uses currency and maximum modification attributes. Our running example includes a simpler permission: the ability to modify a deposit account regardless of the maximum amount of modification and the presence of a foreign currency. Similarly, a power type representation is possible for an account as shown previously in Figure 6.

Table 2 presents the entities and attributes used for describing permissions. The first two rows are cases of the banking example, and the last row shows an additional scenario in which the attributes of event and resource types are also used.

Resulting Context. Using various combinations of events, resources, and their attributes, we will be able to describe permissions with a certain level of detail.

4.3 A Core Access Control Model by Combining Patterns

Name. *Core or Base Access Control Model* Pattern

Context. A core or base access control model represents the basic functions of an authorization model. It is desirable to describe authorization functions similar to the explanation of business models. For instance, RBAC_0 represents a base access control model; excluding the constraints and role hierarchies from Figure 4 creates such a model.

Problem. How can a core access control model for business processes be obtained using the same entities and similar models of business processes? How can access control policies be described such that these policies are based on an access control model? The synergy between an access control model and access control policies previously mentioned (section 1) is related to the latter question.

Solution. The combination of *role modeling and user assignments* and *permission* patterns creates a base model of authorization. Pattern combination has two benefits: the combination of patterns enables us to create an access control model (i.e., RBAC), and access control policies can be described uniformly.

Example. Figure 10 shows the combination of these patterns, and this combination presents an example of a core RBAC model in which an employee role (i.e., teller) is determined in conjunction with one permission for this role.

Table 3 shows the policies of the banking example from Figure 7. The columns of this table represent the operation (event) type, object (resource) type, and role used to define various access control policies. In general, we use a table whose columns are REA entities or their attributes, or which are defined by these entities and attributed (e.g., role) to express access control policies. Figure 10(b) uses a UML class diagram and presents the first row of Table 3; the other rows can be presented similarly. The policies of this table can be described by an access control model: this table presents policies based on the core RBAC.

Resulting Context. To summarize, the combination

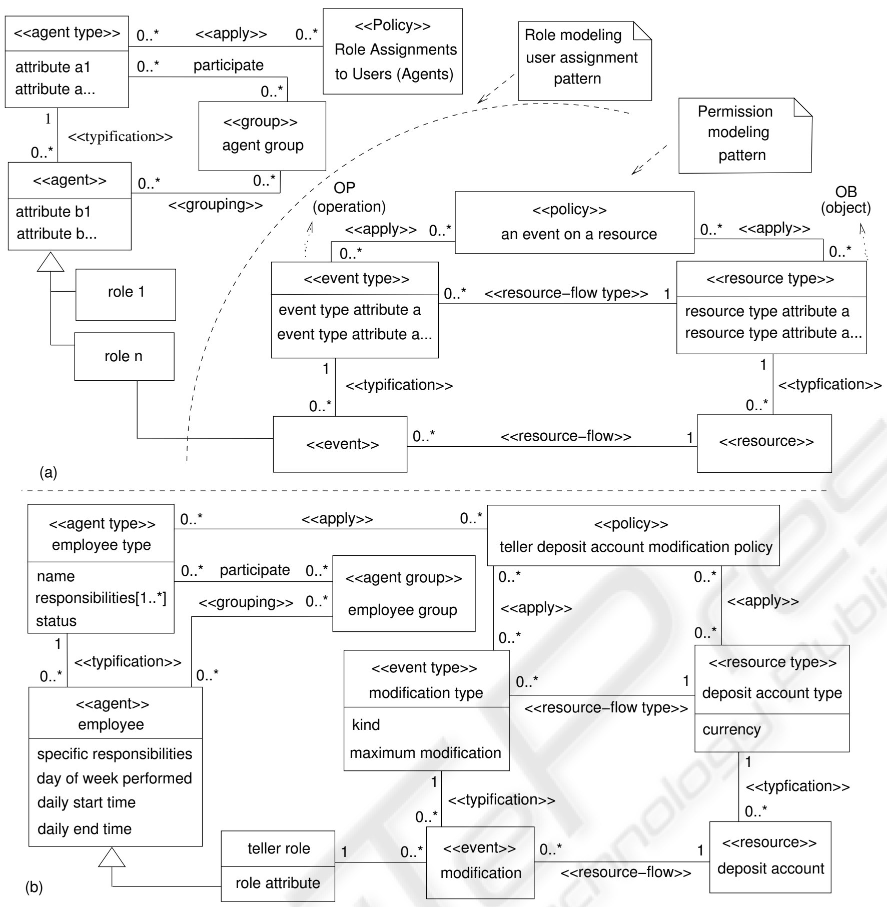

Figure 10: (a) A combination of two presented patterns, (b) an example.

Table 3: A table representation of policies for the banking example.

			Permission	
policies	user type (agent type)	role	operation type (event type)	object type (resource type)
P1	employee	teller	modify	deposit accounts
P2	employee	customer service rep	modify, create, delete	deposit accounts
P3	employee	accountant	create	general ledger reports
P4	employee	accounting manager	modify	ledger posting rules
P5	employee	loan officer	create, modify	loan accounts

of a *role modeling and user assignments* pattern with a *permission modeling* pattern enables us to express a large range of access control policies. This combination creates a core (base) access control model.

Furthermore, as Table 3 shows, it is possible to describe access control policies based on an access control model. For instance, a rule-based policy based on the first row of this table states that if an agent is an employee who fills the role teller, then a permission can exist for this role to allow the modification of deposit accounts.

4.4 Modeling Constraints and Role Hierarchies

In general, constraints constitute a majority of authorization specifications because many such conditions occur in reality. For instance, constraints are an important part of RBAC and are sometimes considered "to be the principle motivation behind RBAC (Sandhu et al., 1996)." Constraints are also very significant components of any access control policies. In terms of RBAC, constraints can be on roles, permissions,

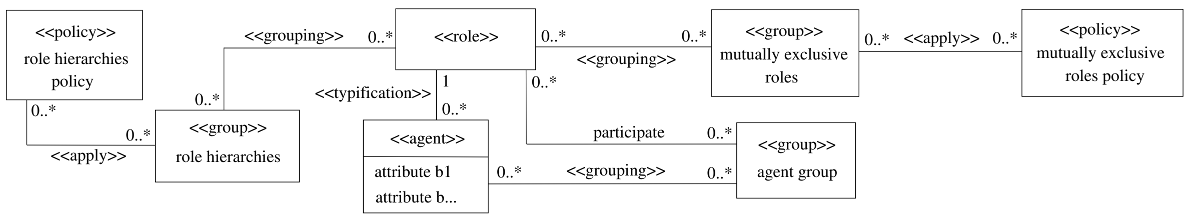

Figure 11: Role hierarchies and mutually conclusive roles.

role hierarchies, and sessions. A major category of constraints is related to the separation of duties.

Constraints on Agents, Roles, and Permissions. the role modeling and user assignments pattern can model constraints on agents such as by using "daily start time" or/and "day of week performed" attributes or constrain roles using role attributes (Figure 8), as described in Section 4.1. Similarly, the permission modeling pattern can model constraints on permissions (i.e., operations on objects) using an operation (e.g., a "maximum adjustment" attribute) and/or an object (e.g., the "currency" attribute of Figure 9). With constraints, it is possible to specify that a pilot (i.e., a role) based on "skill" or other attributes is authorized to fly (an operation or an event) certain type of planes (objects or resources).

Role Hierarchies and Mutually Exclusive Roles. Both role hierarchies and mutually exclusive roles can be viewed as cases in which constraints on roles exist. A variation of the role modeling and user assignments pattern (Section 4.1) is used to describe both these cases.

Name. A variation of *role modeling and user assignments* pattern

Context for Role Hierarchies. Role hierarchies in organizations express the line of authorities or responsibilities and exemplify certain constraints or relationships among roles. Role hierarchies represent a partially ordered set and are written as $RH \subseteq R \times R$: the set of roles with a binary relationship of inheritance between roles. Role hierarchies are usually visualized by a diagram in which, by definition, the roles with higher permissions are located at higher positions of the diagram.

Context for Separation of Duties. Separation of duties is a well-known principle and has two broad categories: static and dynamic; the dynamic one contains several variations (Simon and Zurko, 1997). The static separation of duty and the basic form of dynamic separation of duty represent constraints on roles and sessions, respectively. The static separation of duty represents a strong exclusion of roles in which no user can ever take both roles A and B if these two roles are exclusive (e.g., that a purchaser and an approver of an order must be different people is a common example provided frequently in the literature). The assumption for mutually exclusive roles indicates that these roles cannot gain the permission of their counterpart through using a third role or a combination of roles.

Problem. How are role hierarchies modeled? How can this constraint on roles be added to the previously presented core access model? In general, how can an ordered set be represented in this context? Since roles and agents are closely related, agents must be included in this presentation. In addition, how can the static separation of duties be modeled?

Solution. Figure 11 presents role hierarchies and mutually exclusive roles in which an agent is also shown.

Example. In Figure 11, if a role is a teller, then the role hierarchies group contains teller, customer service rep, and branch manager. Similarly, whenever a role is that of a teller, the mutually exclusive roles group includes teller, accountant, and loan officer roles.

Resulting Context. Role hierarchies are a partially ordered set; therefore, not every member (i.e., role) of this set can be compared with each other. For instance, the privileges of a loan officer and an accounting manager cannot be compared. RBAC mentions dynamic separation of duties whose support needs a dynamic model. In our future work, using a more detailed example than the ones provided by the banking case study, we intend to describe a dynamic model in which dynamic separation of duties is supported.

5 RELATED WORK

Ferraiolo and Atluri (Ferraiolo and Atluri, 2008) ask whether the movement in access control research has been toward a unifying model or a meta-model. They also ponder whether such a unifying framework is a practical approach in view of the existence of a wide range of policies. Finin et al. (Finin et al., 2008) present ROWLBAC for the integration of RBAC (as a representation of an access control model) with OWL, as a language describing access control policies.

Rule-based RBAC (RB-RBAC) (Al-Kahtani and Sandhu, 2002) presents an approach in which users are dynamically assigned to roles based on a finite set of rules. The attributes of users are the determining factors within the rules that assign roles to users. Several extensions to RBAC exist; temporal RBAC (TRBAC) has been mentioned previously. Bertino et al. [14] propose TRBAC, which enables periodic activation and deactivation of roles and allows roles to be either active or inactive within a certain time. They provide the syntax and semantics of TRBAC and show an example. Various policy languages exist. Currently, eXtensible Access Control Markup Language (XACML) (OASIS, 2005) is the XML standard access control policy language. Knowledgable Agent-oriented System (KAoS) (Tonti et al., 2003) enables specification and resolution of policy conflicts where policies are specified as an ontology. Hruby (Hruby, 2006) presents REA as a group of business patterns, organized into two categories: operational and policy.

6 CONCLUSIONS

We describe an approach in which both business processes and access control policies are specified using the same foundational building blocks and similar models. As a result, the two models can be easily integrated. Access control policies are described in two formats: the UML class diagram and a table format. It has also been explained that the policies have the advantage of being based on the RBAC model. In addition, we describe the RBAC model as a combination of patterns, explained uniformly as is common in the pattern community.

REFERENCES

Al-Kahtani, M. and Sandhu, R. (2002). A model for attribute-based user-role assignment. In *ACSAC'02, 18th Annual Computer Security Applications Conference*, pages 353–364. IEEE Computer Society.

Bertino, E., Bonatti, P., and Ferrari, E. (2000). TRBAC: A temporal role-based access control model. In *RBAC'00, Fifth Workshop on Role-Based Access Control*, pages 21–30. ACM.

Blaha, M. and Rumbaugh, J. (2005). *Object-oriented Modeling and Design with UML*. Pearson Prentice Hall, New Jersey, 2nd edition.

Chandramouli, R. (2000). Application of XML tools for enterprise-wide RBAC implementation tasks. In *RBAC'00*, pages 11–18. ACM.

Ferraiolo, D. and Atluri, V. (2008). A meta model for access control: Why is it needed and is it even possible to achieve? In *SACMAT'08, 13th Symposium on Access Control Models and Technologies*, pages 153–154. ACM.

Ferraiolo, D., Kuhn, D., and Chandramouli, R. (2007). *Role-Based Access Control*. Artech House, Boston, 2nd edition.

Ferraiolo, D., Sandhu, R., Gavrila, S., Kuhn, D., and Chandramouli, R. (2001). Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security*, 4(3):224–274.

Finin, T., Joshi, A., Kagal, L., Niu, J., Sandhu, R., Winsborough, W., and Thuraisingham, B. (2008). ROWLBAC: Representing role based access control in OWL. In *SACMAT'08*, pages 73–82. ACM.

Fowler, M. (1997). *Analysis Patterns: Reusable Object Models*. Addison-Wesley, Menlo Park, California.

Geerts, G. and McCarthy, W. (2006). Policy-level specifications in REA enterprise information systems. *Journal of Information Systems*, 20(2):37–63.

Hruby, P. (2006). *Model-Driven Design Using Business Patterns*. Springer-Verlag, New York.

Martin, J. and Odell, J. (1998). *Object-Oriented Methods: a Foundation, UML Edition*. Prentice Hall, New Jersey, 2nd edition.

OASIS (2005). *eXtensible Access Control Markup Language (XACML), Version 2.0*. Organization for the Advancement of Structured Information Standards.

OMG (2009). *Unified Modeling Language (UML) Superstructure, Version 2.2*. Object Management Group.

Ray, I., Li, N., France, R., and Kim, D. (2004). Using UML to visualize role-based access control constraints. In *SACMAT'04*, pages 115–124. ACM.

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

Simon, R. and Zurko, M. (1997). Separation of duty in role-based environments. In *CSFW'97, 10th Computer Security Foundations Workshop*, pages 183–194. IEEE Computer Society.

Tonti, G., Bradshaw, J., Jeffers, R., Montanari, R., Suri, N., and Uszok, A. (2003). Semantic web languages for policy representation and reasoning: A comparison of KAoS, Rei, and Ponder. In *ISWC'03, 2nd International Semantic Web Conference*, pages 419–437. Springer.