Secure interoperability with O2O contracts

Céline Coma-Brebel, Nora Cuppens-Bouhlahia, Frédéric Cuppens

Chapter 1

Secure interoperability with O2O contracts

Céline Coma, Nora Cuppens-Boulahia and Frédéric Cuppens

IT/TELECOM Bretagne, 2 rue de la Châtaigneraie, 35512 Cesson Sévigné, France

The evolution of markets and the high volatility of business requirements put an increasing emphasis on the ability for systems to accommodate the changes required by new organizational needs while maintaining security objectives satisfiability. This is even more true in case of collaboration and interoperability between different organizations and thus between their information systems. Usual solutions do not anticipate interoperability security requirements or do it in a non satisfactory way. In this chapter, we propose a contract and compatibility principles within a formal framework O2O[1] to achieve a secure interoperation. Contracts are used to explicitly represent the rules that determine the way interaction between organizations must be controlled to satisfy secure accesses to resources. Compatibility relations make it possible to derive interoperability security policies. We specify all the wheelwork of interoperation between organizations which might manage their security policies using access control model RBAC[2] and/or OrBAC.[3]

Furthermore, as interoperation may lead to a lot of exchanges of information before and during the interoperability session, in particular those related to credentials and security policies, we propose to ensure privacy protection to use the O2O licence administrative view and an XML block based access control technique to obfuscate some of the information exchanged.

1.1. Introduction

In traditional approaches of data processing and resource access controlling, the model of an organization is rather static. This is sufficient for managing securely the behavior of an organization within settled ranges of actions; it is unsatisfactory when the environment of the organization changes following the occurrence of different events like those generated by interoperability sessions.

Environment changes become an established fact of current organizations. They need to be agile, flexible and securely interoperable since they act in an increasingly dynamic environment which tends to be collaborative but unfortunately possibly hostile. To meet these requirements, synthetic knowledge is needed to let organizations continuously manage accesses, ensure the integrity of exchanges and the continuity of services.

Most of the recent works argue for the use of ontologies as a mean for providing this synthetic knowledge of common domains. Moreover, as this cooperation must be secure for each of the party involved within it, it is necessary to provide a context-aware tailored

mapping process that takes into account this security aspect and derives the security policies. We claim that this derivation process needs to use: (1) ontological mapping, this means mapping structure-oriented entities of the two organizations having to interoperate. Accordingly, a correct meaning can be assigned to these structuring entities which could seem a priori different. Examples of these entities are roles or activities pertaining to these organizations, especially the environmental conditions under which they evolve, and (2) compatibility relations between the grantor organization offering the service and the grantee organization requesting an access to the service, this means mapping deontic-oriented entities of the organizations having to interoperate. These relations will be used to specify interoperability policies. In this case, deontic entities are primarily permissions and prohibitions rules.

Moreover, these interoperability security policies must not be derived on the fly. This can lead to a lack of fairness, inconsistencies and breaches of security. Also, these policies are usually established after a phase of policy negotiation, which can be time consuming. This lapse of time may in some cases be longer than the interoperability session and result in a loss of efficiency.

In this chapter, we propose an approach to anticipate the definition of interoperability security policies and thus shorten the policy negotiation steps. Our work is based on the O2O approach[1] as a framework for a secure interoperability where the authority sphere defining security policy and the management sphere administering this security policy are well identified. We define interoperability contracts to control the aforementioned derivation process of security policies. This process is characterized by a duality between the maximization of the security and the insurance of the cooperation.

The rest of this chapter is organized as follows. In section 1.2 we carry out a survey of existing research works related to secure interoperability and emphasize on their weaknesses. In section 1.3 we recall the two basic bricks we use to express, derive and enforce context-aware interoperability policies: the contextual security model OrBAC[4] and the interoperability framework O2O. In section 1.4 we explain the steps for interoperability establishment. After limiting the scope of interoperability in section 1.6, we state formally the different types of compatibility relations between the grantor and the grantee organizations to establish interoperability security policy rules. This formal statement is based on the ontological mapping and the definition of compatibility constraints. The process guarantees that the interoperability security policy is consistent with the local policy of the grantor organization and the security policy of the grantee organization. Section 1.9 shows how our O2O model applies to manage security of a Virtual Private Organization (VPO). Some privacy requirements and enforcement are discussed in section 1.11. We present a peer-to-peer example to illustrate our approach in section section 1.12. Section 1.13 concludes this chapter.

1.2. Usual Approaches for Interoperability

Actually, there exist three ways to deal with interoperability. The first approach is Federation of Identity Management (FIM). The second kind of interoperability related works is security policy oriented and the third way to manage secure interoperability makes use of ontological approaches.

1.2.1. *Federated Identity Management*

The most famous FIM technologies are Liberty Alliance[5] and Microsoft Passport (Live Identity).[6] FIM generally covers at the same time user management and access management. Several identity services like Single Sign-On,[7] access control and federation of identities, bring together the necessary components to provide a specific pertaining service to a given identity. However, FIM bases trust relationships on reliability of the identities of each participant. This identity is a set of information known about the participants which could be stored across multiple management systems. Certification of identities relies on a relevant trusted authority. So, FIM could have problems to set up the trust relationships. Furthermore, in most of FIM systems, negotiation exchanges are centralized and participants have to be identified by FIM. This choice limits the use cases of interoperability as FIM relies only on the user identities when dealing with security. Consequently, one of the major problems of FIM is the lack of granularity when assigning privileges. For example, if two organizations B and C belong to a same alliance A, all users of B have the same privileges when they access to C. Our approach provides finer grained access control than one can do with FIM.

1.2.2. *Negotiation policy*

Because most of access control models often propose a preestablished policy and cannot take into account new requesters, trust negotiation is a good solution for dynamic policies. In this case, the decision to grant an access is primarily based on exchanges of credentials and accesses to the information they convey. Automated Trust Negotiation,[8] TrustBuilder,[9] and Trust-χ[10] are examples of these approaches. In this case, specific languages have to be defined.

ATN

Automated Trust Negotiation (ATN) is an approach to regulate the exchange of sensitive credentials by using access control policies. The languages used in this case do not make clear separation between the security policy specification and credentials that are used to implement this policy. They are largely inspired by RBAC (Role based Access control) model philosophy.[11] The RBAC model does not take into account the organization dimension, which is essential in the case of interoperability to ensure the confinement principle. Moreover, as the concept of role is prevalent in this model, this is not sufficient to express fine grained access control. Our proposal is context-aware and goes beyond the role based

approaches.

TrustBuilder

TrustBuilder is is one of the most significant proposed system for negotiation of trust in dynamic coalitions. TrustBuilder rests on ABAC model (attribute-based access control model).[12] That is, resources access control policies are written as a declarative specification of the attributes needed in order to gain access to these resources. Each participant of negotiation declares his disclosure sequences. These disclosures sequences allow Trustbuilder to construct disclosure tree via the Trustbuilder strategies (Simple Strategy, Relevant Strategy, ...). With these disclosure trees, participants can disclose credentials in order to establish trust and to gain access to a resource. Each node of these trees are labeled with a credential. So, when a participant wants to access to a resource, he just has to disclose gradually requested credential with respect to the resource sensitivity. So this approach incorporates policy disclosure; only policies that are relevant to the current negotiation may be disclosed by the concerned parties. These policies specify what combinations of credentials one can present in order to gain access to a protected resource of the accessed service. In this way, it is possible to focus the negotiation and base disclosures on need-to-know. Since these policies may contain sensitive information, their disclosure can also be managed by a strategy. Furthermore, TrustBuilder project develops a prototype[9] whose goal is to create scalable, reusable trust negotiation components. But again, the security policy are not explicitly stated, the focus is mainly on the negotiation aspects.

Trust-χ

Bertino et al. propose Trust-χ an XML-based framework for trust negotiations.[10] The architecture of the main component of the trust-X is symmetric and both parties are Servent (Server-Client). So it could be used by peer-to-peer systems. Trust-χ certificates are either credentials or declarations. A credential states personal characteristics of its owner, certified by a Credential Authority (CA), whereas declarations collect personal information about its owner that do not need to be certified. These declarations only help in better customized service. To specify certificates and policies trust-χ use χ-TNL.[13] A novel aspect of χ-TNL presented in this paper is the use of trust tickets and policy preconditions.

All participants in Trust-χ are considered equally. So each party has its system of negotiation and a view on the state of the negotiation process. All information about the user are collected in χ-profiles. So χ-profile contains all users' credential and declarations. All theses documents are XML-structured for a better homogeneity. Information could be constructed by entities which do not share a common ontology, but system could correctly interpret information with the use of namespaces combined with certificate type system.

To reduce the number of exchange in negotiation, Bertino *et al.* introduce the trust ticket. The trust ticket is produced at the end of well-pass negotiation, and proves that the provider can trust the requester for a given resource. The core of trust ticket is the sequence of certificates which allows the negotiation to be well-pass.

Trust-χ resource can be either a certificate or a service. A resource can be characterized

by a set of attributes. Policies for a resource could be defined as follows:
$p1 = (\{\}, peer\_node \leftarrow peer\_id())$;
$p2 = (\{p1\}, peer\_node \leftarrow DELIV)$;
One problem of this type of specification policy, is the loops created which make the specification incorrect. So, the system has to check if the chain of policies is well-formed.

In Trust-χ, the trust negotiation is composed of three phases, the first is the introduction phase, the second phase is the sequence generation phase and the third is caching of trust sequence. The second phase can be done by performing the policy evaluation phase (with negotiation tree building), by exchanging trust tickets and by using the sequence prediction module (based on similarity with previous negotiations).

So, Bertino *et al.* with the use of precondition policies make the administration not user friendly. Because, the policy designer has to be sure that the policy is well-formed. This leads to a real scalability problem in particular in peer-to-peer systems.

1.2.3. *Ontological approaches*

In the case of interoperability, each organization should inform the other organizations about some knowledge it wants to share with them, in order to help the set up or the continuation of the (potential) interoperability sessions. The requirement is that this knowledge must be understandable by the interoperability session's participants. An ontology provides an explicit conceptualization that describes the semantics of the data and provides a common sharing knowledge which can be easily communicated. Languages such as OWL[14] or DAML[15] have been developed to share a knowledge model by using ontologies. REI[16] and KAoS[17] are examples of models belonging to this category.

KAoS

KAoS is designed to specify and reason on security policies of interoperable environments, such as Grid or Web services. It was firstly based on DAML+OIL[15] a language that combines the features of both the DARPA Agent Markup Language DAML and Ontology Interchange Language OIL. Then KAoS language has been extended using the Web Ontology Language OWL.[14] KAoS manages positive and negative authorizations and obligations. To facilitate expression of ontologies, KAoS proposes as ontology entries more than one hundred reusable classes of Policy Ontologies. Theses classes are composed of actions, actors, groups, places, policies, etc. Here is an excerpt of a KAoS policy:

```
<rdfs:comment xml:lang="en"> An action </rdfs:comment>
 <owl:Class rdf:ID="ActionExample">
  <rdfs:subClassOf rdf:resource="#CommunicationAction" />
    <owl:Restriction>
       <owl:onProperty rdf:resource="#performedBy" />
       <owl:toClass rdf:resource="#MembersOfDomainA" />
    </owl:Restriction>
```

```
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
       <owl:onProperty rdf:resource="#hasDestination" />
       <owl:toClass rdf:resource="#notMembersOfDomainA " />
    </owl:Restriction>
  </rdfs:subClassOf>
 </owl:Class>

<rdfs:comment xml:lang="en"> An authorization</rdfs:comment>
   <policy:PosAuthorizationPolicy rdf:ID="Example">
   <policy:controls rdf:resource="#ActionExample" />
   <policy:hasSiteOfEnforcement rdf:resource="#ActorSite" />
   <policy:hasPriority>10</policy:hasPriority>
```

In KAoS, policies are administrated using a tool called KPAT (KAoS Policy Administration Tool) which is used also to deal with conflicts appearing in a KAoS security policy. In this case, KPAT calls a conflict Java Theorem Prover (JTP). Therefore, KAoS users can cumulate different classes of policy ontology and then apply JTP to manage the conflicts. Unfortunately, KAoS is restricted to the specification of security policies that do not require the use of variable parameters. It is not a fully context-aware language though some security policies may need to define security rules according to some imposed constraints. These constraints can, for example, depend on parameters that are known at the time of service deployment or execution. Thus, policies expressed with KAoS are not enough dynamic to fit all the interoperability requirements.

REI

REI extends the KAoS language by adding logical variables to increase its expressivity. Like KAoS, REI also manages inconsistency of the security rules. It includes the specification of meta-policies to solve conflicts.

Security policies expressed with REI are based on OWL-lite.[14] OWL-lite is an ontological language which allows REI to express security policies with constraints and obligations on environmental resources. But the use of OWL-lite occasions several restrictions in the REI policies expression. For instance, the use of disjunction and negation are not possible and cardinality is reduced to 0 or 1. Therefore, REI extends OWL-lite language by adding variables. This extension allows REI to be more flexible. Three sets compose REI core policies. The first policy set is composed of action combinations and conditions. In that case, predicates are structured as policy objects $PolicyObject(Action, Condition)$. These policy objects are permission, prohibition, obligation and dispensation. As prohibitions are negative permissions and dispensations are negative obligations, conflicts can appear in REI security policy specification. These conflicts are solved in metapolicies using override principles (negative over positive preference or vice versa). The second policy set is used to link an entity to a $policyObject$. These entities are agent or object. For

instance, the fact that a subject which is a professor has the permission to examine a student is expressed by $has(Var, right(examineStudent, professor(Var))$. The third policy set expresses actions, their effects and requisite preconditions. However, REI ontology only deals with constraints and description of local entities which is far from being enough in case of interoperability.

Thus, we should go further in this direction to take into account local, external and contextual information and derive a dynamic security policy on which we can reason. That is our proposal in this chapter.

1.3. Generic Interoperation Policies

1.3.1. *Contextual Security Policy: the OrBAC model*

The goal of our approach is to anticipate the management of security of potential interoperability sessions. To achieve this goal, we need an approach which allows both fine-grained access control and interoperation. The OrBAC model[4] is an access control model in which the policy designer can express the security policy of an information system at the organizational level, that means independently of the future implementation of this policy. So, the operational system security requirements could be expressed and then deployed over various security components, considered in OrBAC as sub-organizations of the information system organization. This deployment rests also on administrative responsibilities which could be granted to subjects assigned to different roles.

In OrBAC, the traditional access control triple (subject, action, object) is abstracted at the organizational level into the triple (role, activity, view). A role (respectively an activity and a view) is a set of subjects (respectively actions and objects) to which the same security rules apply. This reduces the number of security rules to define. Furthermore, we need to specify dynamic security rules to adapt the policy to the context and satisfy interoperability requirements. OrBAC is context sensitive, so that the policy could be expressed dynamically by taking into account environmental events and states. For instance, the following OrBAC security rule:

$$securityRule(\ network,\ permission(\ peer,\ access,\ resource,\ memberOfnetwork))$$

means that in organization *network*, a *peer* has the permission to *access* to a *resource* in a context where this *peer* is member of *network*.

Moreover, using the OrBAC model we can specify a decentralized security policy administration that complies with four principles: (1) Organization centric administration, (2) Administration policy definition at the organizational level, (3) Contextual delegation capabilities and (4) Inheritance of security rules through hierarchies. An ontological specification of OrBAC using OWL-DL[14] which rests on the four aforementioned administration principles, has been defined to meet interoperability objectives.[18]

In order to avoid overloading the figure 1.1, we have just represented classes and their hierarchies. In the OrBAC model, concrete security policy rules are derived automatically

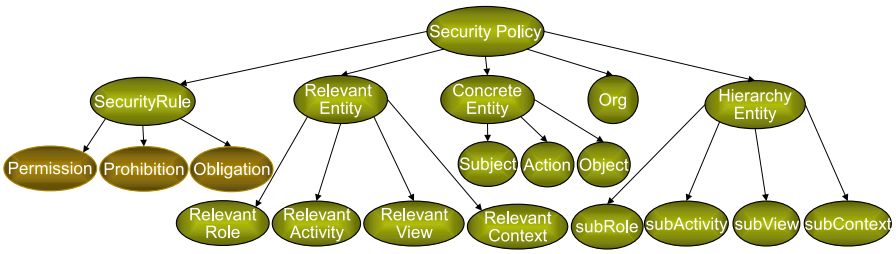

Fig. 1.1. OrBAC Ontology.

from abstract ones. Using OWL-DL, this derivation is expressed as the following:

$$
\begin{aligned}
&is\_permitted(?subject, ?action, ?object) \leftarrow \\
&\ securityRule(?org, permission(?role, ?activity, ?view, ?context)) \wedge \\
&\ hasProperty(?org, empower(?subject, ?role)) \wedge \\
&\ hasProperty(?org, consider(?action, ?activity)) \wedge \\
&\ hasProperty(?org, use(?object, ?view)) \wedge \\
&\ hold(?org, context(?subject, ?action, ?object, ?context)).
\end{aligned}
$$

Derivation in the OrBAC model is computable in polynomial time.[19]

1.3.2. *Interoperability Framework: O2O principles*

O2O (Organization to Organization)[1] is both a model and a framework to manage interoperability between components having their own policies defined by different organizations. To explain the basic principles of O2O, let us consider a given organization Alice.org that wants to interoperate with another organization Bob.org. In this case, each organization has to define a Virtual Private Organization (VPO) respectively called Alice2Bob (A2B for short) and Bob2Alice (B2A for short). The VPO A2B is associated with a security policy that manages how subjects from the grantee organization Alice.org, $Ograntee$, may have an access to the grantor organization Bob.org, $Ograntor$. We say that the VPO A2B manages the interoperability security policy from Alice.org to Bob.org. The VPO B2A is similarly defined to control accesses of subjects from Bob.org to Alice.org. Hence, a VPO is a dynamic organization created to achieve a given interoperability purpose and canceled once this purpose in no more needed.

In a VPO, the Ograntor organization controls interoperability policy and so, the assignment of concrete entity (subjects, actions and objects) to abstract entities (roles, activities and views). Unlike usual virtual organization, in O2O objects are always controlled by their organization. When assigning subjects, actions and objects to their respective roles, activities and views defined in a VPO, some restrictions apply:

- objects come from the Ograntor organization.

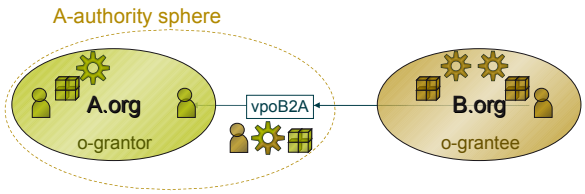

Fig. 1.2. O2O (Organization to Organization) Framework.

This is because, in a VPO, the Ograntor organization can only grant access to its ”own” objects.

- subjects are from Ograntee organization.
  This is because a VPO is designed to control how subjects from the Ograntee organization may have an access to the Ograntor organization.
- actions are from Ograntor organization or could be initiated by Ograntee organization, they are still defined by Ograntor. This is because a VPO must control the action realized on its objects. When this action comes from the Ograntee organization, Ograntor should trust it to execute it.

O2O is formally defined as an extension of the OrBAC model. Organizations which use O2O to interoperate could have different trust levels. For instance, we can have in one hand a military organization which requires a very strict interoperability policy and in a second hand, a store which needs a more attractive policy. Each of these organizations can define interoperability policy with its own security level and access conditions.

1.4. Interoperability Establishment Steps: the O2O process

The interoperability process we propose is composed of three steps. (1) The contract definition step, (2) the interoperability security policy definition step and (3) the interoperability security policy management step. The figure 1.3 shows the sequencing of interoperability security policy establishment.

First of all, before the interoperation sessions, the *Ograntor* organization defines contracts for each organization type it has to interoperate with (*cf.* section 1.5). For that purpose, the *Ograntor* defines the scope of each contract. It specifies entities and information that can be provided during interoperability. In the second step (*cf.* section 1.6), the *Ograntor* organization defines some patterns of interoperability security rules on the basis of its local security policy and according to the type of interoperability. Thirdly, using ontologies, we establish compatibility relations (*cf.* section 1.7). Next, using these compatibility relations and contracts, we derive the interoperability security rules (*cf.* section 1.8). At this stage, the VPO and its interoperability policy is created.

Finally, the *Ograntor* decides how this VPO will be managed: Centralized, decen-

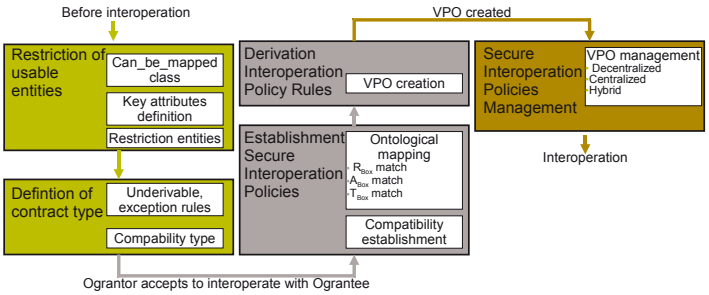

Fig. 1.3. Sequence of Interoperability Security Policy Establishment.

tralized or hybrid administration (*cf.* section 1.9). The *Ograntor* should also ensure the administration of its interoperability policy. To reach this goal, the *Ograntor* must express administration rules (*cf.* section 1.10).

1.5. Interoperability Contract

The contract

Interoperability security policies defined and managed in VPOs must be compliant with local security policies. They have not to be defined either on the fly or from scratch. Thus, the VPO security policies are derived from the local policies. The *Ograntor*, based jointly on its objectives (security), its environment (interoperability) and also its cognitive capacities (its own knowledge and knowledge it has about the grantee organization) proposes interoperability contracts. According to the grantee organization type, a **contract** specifies the way the local policy may be tuned and fixes the scope of this adaptation.

The scope

In the interoperability contract, entities that can be shared are specified. This specification is done using the classes (or sub-classes) to whom these entities belongs. The benefit of using classes is threefold: (1) we avoid to overload all the entities, (2) it does not matter if the entity is a subject, an object or an action as the class overrides this difference and, (3) finally, a class is not organization dependent.

Each class is associated with some attributes. In the logical formalism of the OrBAC model, an attribute is specified by a binary predicate. For example, the fact $peername(p1, p1name)$ means that $p1$ has an attribute *peername* whose value is $p1name$. The predicate *classAssign* is used to express that some object belongs to some class. For example, $classAssign(p1, peerDesc)$ means that $p1$ belongs to the class *peerDesc*. Then, the grantor organization specifies the possible interoperation entities in some given contract using the predicate *can_be_mapped*. The fact $can\_be\_mapped(?grantee, ?Ograntor, ?className)$ means that all entities belonging to the class $?className$ can be used, and so mapped during the interoperation between $?Ograntor$ and $?grantee$. Notice that, in that case,

?*Ograntor* is some organization *A* and ?*grantee* can be an organization *B* or also an organization type, for instance every organization that belongs to some peer to peer network.

The effective mapping

Among attributes of a class, there are decisive attributes. These decisive attributes indicate which attributes must be used in the ontological mapping process between the grantee and the grantor organizations; other attributes are not taken into account. So, for each class appearing in the interoperability contract, its decisive attributes are specified. This is achieved using the predicate $class(?className, ?attribute, Key\_att)$. For instance, if the interoperability contract requires that, in order to authenticate a user in some peer to peer network, only one of its identities (IP address, Id_client, etc.) have to be mapped, the attributes related to the peer identity are decisive attributes. Moreover, in this example a threshold is specified (only one attribute is needed) and required to conclude that the mapping is accepted.

We can go further. The key attribute requirement can be strengthened. The interoperability contract could specify that, for a mapping to be effective, a particular key attribute (key key attribute) have to be mapped between entities within ontological key attributes.[a]

For instance, in a peer to peer network, only the hash is used to establish a compatibility between the shared files' contents. So, the hash attribute is such a key key attribute. The figure 1.4 gives examples of such attributes. In figure 1.4, the ontological decisive attributes are underlined. To establish compatibility between two elements, either all the ontological key attributes should match, or more than *AmatchThreshold* ratio decisive attributes should match.

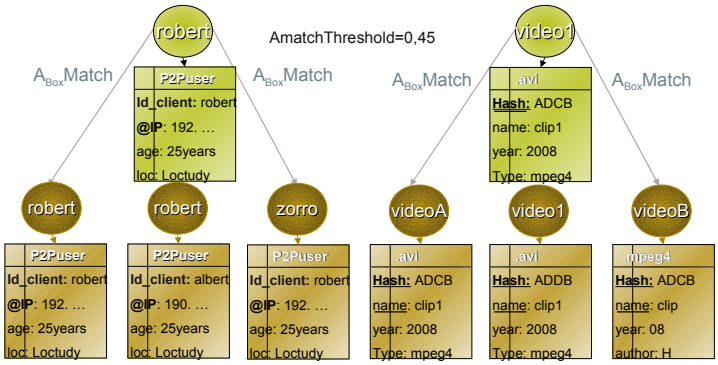

Fig. 1.4. Decisive Attributes and Ontological Key Attributes.

[a]To have a mapping between two entities, it is necessary that all their ontological key attributes match (the conjunction of their attributes matches); whereas it is only when some sets of decisive attributes (some disjunction) match that a mapping between two entities is validated (*cf.* §1.7.1).

1.6. Interoperability Contract Specification

In the O2O approach, each organization defines and manages its VPO policies. The administration of VPO must be both compliant with the local security policy and allows the grantor organization to carry out a flexible interoperability. In the following, we set forth VPO policy derivation requirements.

1.6.1. *Underivability and Exception*

Obviously, some privileges are strictly local to the grantor organization. So, to bound the propagation of the security rules related to these kinds of privileges, the predicate *underivable* is used. Thus, the following fact:

$$underivable(?grantee, securityRule(?Ograntor, permission(?r, ?activity, ?view, ?cxt)))$$

specifies that the *securityRule* defined in the local policy of $?Ograntor$ cannot be derivable in the VPO policy of the $?grantee$ organization or organization type.

Moreover, a collaborative organization can have specific security rules related to its security policy externalization. These rules are not used in the local policy. The predicate *exception* is used to express such exception rules. The fact:

$$exception(?grantee, securityRule(?Ograntor, prohibition(?r, ?activity, ?view, ?cxt)))$$

means that a prohibition is added to the VPO policy when a $?grantee$ organization requires an interoperability session with $?Ograntor$. An exception security rule is either a prohibition or an obligation.

Usually, a conflict can appear between a permission and a prohibition. Thus, exceptions can also create new conflicts. To ease the policy designer's work, we anticipate the resolution of conflicts generated by such exception rules. For this purpose and as suggested in,[19] conflicts are solved by assigning priorities to security rules. Thus, exception rules are always associated with higher priority than other security rules (including the non derivable rules). This conflict resolution strategy is chosen to manage the shadowing anomaly[b] and[19] shows that this strategy is computable in polynomial time.

1.6.2. *Compatibility Relation Patterns*

A compatibility relation can be defined if and only if an ontological mapping between *Ograntee* entities and *Ograntor* entities has been established elsewhere a new package of security rules is defined and added to the VPO policy. Security rules of the VPO are derived according to this mapping, the contract, underivability and exception requirements and the restrictions to be applied to the local roles specified by the *Ograntor* organization for a given *Ograntee* organization (see figure 1.5). We model these restrictions as compatibility relations.

[b]A rule is shadowed by a higher priority rule if it can never be activated.

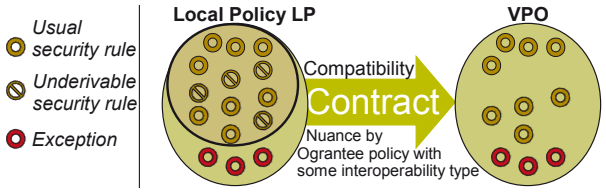

Fig. 1.5. VPO Derived from Contract.

Let *rOgrantor* be a role defined in the *Ograntor* organization and *rOgrantee* the corresponding grantee role defined in the *Ograntee* organization. And let O_{vpo} be the VPO associated with *Ograntee*. There are four main compatibility patterns:

- **Total compatibility relation** (*T_compatibility*)**.**
  If *Ograntor* and *Ograntee* are *T_compatible* then, in O_{vpo}, the grantee role *rOgrantee* inherits all the security rules assigned to *rOgrantor* along with exception rules. Notice that underivable security rules are not inherited.
- **Partial compatibility relation** (*P_compatibility*)**.**
  If *Ograntor* and *Ograntee* are *P_compatible* then, in O_{vpo}, the grantee role *rOgrantee* inherits the security rules associated with *rOgrantor* along with some constraints and exceptions. These constraints are restrictions on activities, views, contexts. Underivable security rules are not inherited.
- **Symmetric compatibility relation** (*S_compatibility*)**.**
  If *Ograntor* and *Ograntee* are *S_compatible* then, in O_{vpo}, the grantee role *rOgrantee* inherits the security rules of a common derivable subset of the two local policies of *Ograntee* and *Ograntor*, with some constraints, along with some exceptions.
- **No compatibility relation** (*No_compatibility*)**.**
  If *Ograntor* and *Ograntee* are *No_compatible* then, in O_{vpo}, there is no security rule related to *rOgrantee*. In this case, the interoperation cannot take place through the role *rOgrantee*.

In the case of partial compatibility relation, **restrictions** on view, activity and context are specified. To do so, we define a restriction predicate for each of these OrBAC entities. For instance, $restrictionActivity(Ograntor, A, RA)$ means that during partial compatibility interoperation the activity *A* is replaced by the restriction activity *RA*. For instance, creating activity during interoperability, can be restricted to updating activity. Notice that restriction entity should be included in previous entities.

In our approach, the type of compatibility between organizations having to undertake interoperability sessions must be known. We define the predicate *type _compatibility* for this purpose. The fact,

$type\_compatibility(Ograntor, Ograntee, Type)$,

where $Type \in \{T\_compatible, P\_compatible, S\_compatible, No\_compatible\}$, means that $Ograntor$ and $Ograntee$ have a $Type$ compatibility relation.

$No\_compatibility$ is the default compatible type. When this default type is used this means the end of the collaboration. In this case, all privileges related to this collaboration, if any, are revoked.

1.6.3. *Contract example*

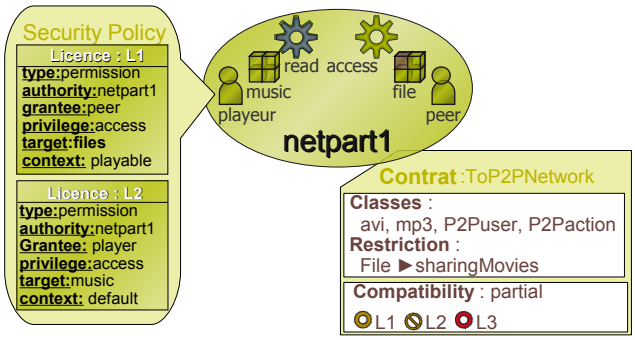

Fig. 1.6. Example of Contract.

First part of contract

In this part, the $Ograntor$ organization indicates which entities are usable during interoperability. By default, all entities are not usable. For instance, in figure 1.6, the action *read* is not usable (in grey), because any usable class (can_be_mapped) contains this action.

$can\_be\_mapped(toP2PNetwork, netpart1, avi)$.
$can\_be\_mapped(toP2PNetwork, netpart1, mp3)$.
$can\_be\_mapped(toP2PNetwork, netpart1, p2pUser)$.
$can\_be\_mapped(toP2PNetwork, netpart1, p2pAction)$.

In figure 1.6, we can see that there is a restriction on view $files$.

$restrictionView(netpart1, files, sharingMovies)$.

Second part of contract

In this second part, the $Ograntor$ organization limits the security rules which can be derived to create the interoperation policy. By default, all security rules can be derived such as $Licence1$.

$Licence1$: $securityRule(netpart1, permission(peer, access, files, default))$.

Some licence , such as *Licence2* are not derivable.

Licence2:
$underivable(toP2PNetwork, securityRule(netpart1, permission(player, access, music, default)))$.

The *Ograntor* organization also expresses exception.

Licence3: $exception(toP2PNetwork,\ securityRule(netpart1,\ prohibition(peer,\ access,\ music,\ default)))$.

The *Ograntor* indicates which compatibility will be used to create interoperability policy.

$type\_compatibility(netpart1,\ toP2PNetwork,\ P\_compatible)$.

1.7. Secure Interoperability Policy Establishment

1.7.1. *Ontological Mapping*

During interoperability, access control cannot be set up without semantic compatibility establishment between security policies of parties that have to interoperate. Entities involved in a collaboration need common knowledge which should be understandable by all parties. We stress on the fact that to get good interoperability, organizations need to share understandable information. To facilitate this sharing, our approach is based on the definition of ontologies and mappings between these ontologies. These mappings allow organizations to provide a shared and common understanding of an application domain that can be exchanged between organizations and application systems undertaking collaboration.

An ontology is composed of instances, ontological classes and properties. Instances represent entities related to ontology domain, and consequently to information system. Ontological classes are sets of instances, used to structure the ontology by the way of relations like hierarchy, intersection, union,... Properties are relationships between instances. Our ontological mapping is based on T_{BOX} mapping related to the mapping of ontological classes, properties between classes and hierarchies, A_{BOX} mapping related to the mappings of instances and R_{BOX} mapping related to the mappings of rules and properties between instances. These mappings apply only to entities that are authorized to be mapped during interoperability by the corresponding contracts. To illustrate the ontological mapping process, we further describe the A_{BOX} mapping, the R_{BOX} mapping and the T_{BOX} mapping.

A_{BOX} mapping: instances mapping

A_{BOX} mapping rests on correspondences between *attributes* and their associated *values*. Thus, we first have to map attributes before concepts. This kind of mapping has already been studied in several works, such as those done in COMA++[20] and GLUE,[21] but without considering security issues. Notice that A_{BOX} mapping is easy to establish or to revoke when the decisive attributes are set up. Thus, to establish A_{BOX} mapping, we compare the number of correspondences of decisive attributes with a threshold, beginning first with ontological key attributes. Let us consider two entities $c_1 \in Ograntor$, $c_2 \in Ograntee$ and

$Sim(c_1,c_2)$ the measure of similarity between c_1 and c_2. First case: c_1 and c_2 have ontological key attributes. In that case, if all the ontological key attributes of c_1 and c_2 match, $Sim(c_1,c_2) = 1$, else $Sim(c_1,c_2) = 0$. Second case: c_1 and c_2 have no ontological key attributes but decisive attributes. We use a normalized value. So, let $|A_{c_1}|$ be the number of decisive attributes associated to c_1 and $|A_{c_1} \cap A_{c_2}|$ the number of decisive attributes c_1 and c_2 have in common. Then, the similarity Sim between c_1 and c_2 is evaluated using the formula $Sim(c_1,c_2) = \frac{|A_{c_1} \cap A_{c_2}|}{min(|A_{c_1}|,|A_{c_2}|)} \in [0..1]$. Then, A_{BOX} mappings are established if the similarity value belongs to the interval $[AmatchThreshold, 1]$. $AmatchThreshold$ is a threshold set by the $Ograntor$ organization and specified in the contract.

R_{BOX} mapping: properties mapping

To obtain the R_{BOX} mapping, we analyze class taxonomies of the two organizations that need to interoperate and their *inference rules*. From internal and composition relations of the security policy ontology, we could derive new information about security policies in one or both of these organizations. For instance, if we consider a common knowledge about the existence of a hierarchical relation between two entities, the views *movies* and *shared_movies*, this hierarchical relation is automatically derived by the following rule:

$sub\_view(?org, ?v1, ?v2) \leftarrow$
$view(?org, ?v1) \wedge view(?org, ?v2) \wedge$
$restrictionView(?v1, ?v2)$.

Let us assume that both organizations use the same context taxonomy. So, the inference rules associated with the security policy ontology will be the same. In this case, we define a fact $viewRmatch(org, pred1(?v1), pred2(?v2))$ where $pred1$ and $pred2$ are view predicates, $?arg1s$ and $?args2$ their arguments vertex, and $viewRmatch$ means that the two view predicates are compatible. If one of the two view predicates belongs to the other organization undertaking an interoperation, the same fact can be used to establish a compatibility and then derive the security rule to be activated.

T_{BOX} mapping: ontological classes mapping

This is a *schema* mapping. We could base T_{BOX} matching on existing works.[20–22] As T_{BOX} matching requires both schema and semantics correspondences, these works simplify and automatize search of entities matching. An OrBAC security policy is defined independently of its implementation. So, in order to preserve this suitable property in our ontology mapping, we have to define a more generic mapping than the classical mappings between two organizations. The reuse of previously global established mapping results simplifies interoperability.

For instance, during file exchanges in a P2P network, several view mappings are similar. In this case, there are two ways to reuse these pre-established similar mappings: 1) we could define mappings common to all peers of the networks. These common mappings could be managed by a server, or 2) we could determine a mapping between two peers of the network and try to generalize this mapping to other peers. This allows us to define a hierarchy of mappings in the ontology structures.

CUPID[22] is a good T_{BOX} mapping approach. To establish when *view* $Tmatch(?org, ?v1, ?v2)$ is true, A_{BOX} similarities have to be established first. There are three structuring cases to get a T_{BOX} matching: (1) the compared views or entities have no sub-views, (2) all sub-views of the two mapped views are globally mapped by A_{BOX} mappings, 3) the hierarchy schema of the two views are structurally similar and a set of sub-views are mapped by A_{BOX} mappings, even if immediate sub-views of the two views are not mapped.

1.7.2. *Establishment of Compatibility Relations*

The establishment of compatibility relations is a prerequisite condition to the derivation process of interoperability security rules. It is based on the three aforementioned ontological mappings, T_{BOX}, A_{BOX} and R_{BOX} and the four compatibility relation patterns: no compatibility, total compatibility, partial compatibility and symmetric compatibility. So, for each entity used to express a security rule in the access control model (namely role, activity, view and context), we define a compatibility predicate. The *role_compatibility* predicate is defined as the following:

$role\_compatibility(?orga2orgb, ?ra, ?rb) \leftarrow roleTmatch(?orga2orgb, ?ra, ?rb).$

which means that, in the VPO *orga2orgb*, two roles *ra* and *rb* are compatible, if there exists an ontological mapping between *ra* and *rb*. The *view_compatibility* predicate is defined as the following:

$$\begin{aligned} &view\_compatibility(?orga2orgb, ?resVa, ?vb) \leftarrow \\ &\ viewTmatch(?orga2orgb, ?va, ?vb) \wedge grantor(?orga2orgb, ?orga) \wedge \\ &\ grantee(?orga2orgb, ?orgb) \wedge restrictionView(?orga, ?va, ?resVa) \wedge \\ &\ type\_compatibility(?orga, ?orgb, P\_compatibility). \end{aligned}$$

which means that two views *resVa* and *vb* in the VPO *orga2orgb* are compatible if there exists an ontological mapping between a view *va* belonging to the grantor organization *orga* and a view *vb* belonging to the grantee organization *orgb* and, *resVa* is a restricted view of *va* and, there is a *P_compatibility* relation between *orga* and *orgb*.

The two predicates *activity_compatibility* and *context_compatibility* are similarly defined.

1.8. Derivation of the Interoperability Security Policy: automatic VPO creation

1.8.1. *Derivation rules*

Once some agreements exist on the compatibility of entities of those organizations having to undertake some interoperability sessions, security rules are derived from the local policies.

Total compatibility
In the case of total compatibility, a security rule is derived as the following:

$$\begin{aligned}&securityRule(?orga2orgb, permission(?rb, ?a, ?v, ?c)) \leftarrow\\ &\ type\_compatibility(?orga, ?orgb, T\_compatibility) \wedge\\ &\ grantor(?orga2orgb, ?orga) \wedge grantee(?orga2orgb, ?orgb) \wedge\\ &\ securityRule(?orga, permission(?ra, ?a, ?v, ?c)) \wedge\\ &\ role\_compatibility(?orga2orgb, ?ra, ?rb) \wedge\\ &\ not(underivable(?orgb, securityRule(?orga, permission(?rb, ?a, ?v, ?c)))).\end{aligned}$$

Partial compatibility
In the case of partial compatibility type, the security rule derivation depends on restrictions. Notice that if an entity has no restriction, the restriction of this entity is itself:

$$\begin{aligned}&securityRule(?orga2orgb, permission(?rb, ?resA, ?resV, ?resC)) \leftarrow\\ &\ type\_compatibility(?orga, ?orgb, P\_compatibility) \wedge\\ &\ grantor(?orga2orgb, ?orga) \wedge grantee(?orga2orgb, ?orgb) \wedge\\ &\ securityRule(?orga, permission(?ra, ?a, ?v, ?cxt)) \wedge\\ &\ role\_compatibility(?orga2orgb, ?ra, ?rb) \wedge restrictionView(?orga, ?v, ?resV) \wedge\\ &\ restrictionActivity(?orga, ?a, ?resA) \wedge restrictionContext(?orga, ?c, ?resC) \wedge\\ &\ not(underivable(?orgb, securityRule(?orga, permission(?rb, ?a, ?v, ?c)))).\end{aligned}$$

Symmetric compatibility
When the contract specifies a symmetric compatibility, the concerned activities, views and contexts in the grantee and grantor organizations have to be compatible:

$$\begin{aligned}&securityRule(?orga2orgb, permission(?r, ?ab, ?vb, ?cb)) \leftarrow\\ &\ type\_compatibility(?orga, ?orgb, S\_compatibility) \wedge\\ &\ grantor(?orga2orgb, ?orga) \wedge grantee(?orga2orgb, ?orgb) \wedge\\ &\ securityRule(?orgb, permission(?r, ?aa, ?va, ?ca)) \wedge\\ &\ activity\_compatibility(?orga2orgb, ?aa, ?ab) \wedge\\ &\ view\_compatibility(?orga2orgb, ?va, ?vb) \wedge\\ &\ context\_compatibility(?orga2orgb, ?ca, ?cb) \wedge\\ &\ not(underivable(?orgb, securityRule(?orga, permission(?r, ?ab, ?vb, ?cb)))).\end{aligned}$$

1.8.2. *Example of derivation of an interoperability rule*

Let us take an example of derivation of an interoperability rule.

$$\begin{aligned}&securityRule(network, permission(node, access, sharingMovies, lawfullyMovies)) \leftarrow\\ &\ type\_compatibility(netpart1, netpart2, P\_compatibility) \wedge\\ &\ grantor(network, netpart1) \wedge grantee(network, netpart2) \wedge\\ &\ securityRule(netpart1, permission(peer, access, files, default)) \wedge\\ &\ role\_compatibility(network, peer, node) \wedge\\ &\ restrictionView(netpart1, files, sharingMovies) \wedge\end{aligned}$$

$restrictionActivity(netpart1, access, access) \wedge$
$restrictionContext(netpart1, default, lawfullyMovies) \wedge$
$not(underivable(netpart2, securityRule(netpart1, permission(peer, access, files,$
$lawfullyMovies))))$.

In this example of security rule derivation, organization *netpart*2 has established a partial compatibility type contract with organization *netpart*2. So, the permission of a peer in the grantee organization *netpart*1 to get an access to a file belonging to the grantor organization *netpart*2 must be tuned in the VPO *network* during the interoperability sessions. The default context is restricted to a context related to $lawfullyMovies$. The role *peer* must be compatible with the role *node*. Furthermore, organization *netpart*1 restricts its file to an interoperable view *SharingMovies*. So, in the interoperability security policy we derive the permission for a node in the VPO organization *network* to access to interoperation files in the $lawfullyMovies$ context, that is to say movies which respect the legal age according to the *grantee* country.

1.9. VPO management: Secure interoperation policy management

In a Virtual Organization (VO), several organizations share some of their subjects, actions and objects to achieve a common purpose. Usually, an initiator organization, which wants to create a VO, will have to issue a query to other organizations it wants to interoperate with. The VO will be created if all the organizations that receive this query agree to be a member of this VO. Each of these organizations will require that the access to its resources must be compliant with some security policy. We claim that these interoperability security policies defined by the different organizations actually correspond to VPOs.

Thus, in our O2O approach, the security policy of the VO is the union of all these VPOs. The problem is then to define how to manage the security policy of the VO. There are three main approaches: decentralized VPO management, centralized VPO management and hybrid VPO management.

Decentralized VPO management:
Like we have seen before, each organization defines its VPOs to control interoperability with other organizations in the authority sphere. Then, each organization will manage those VPOs that are inside its sphere of authority. The decentralized VPO management corresponds to the figure 1.7. When a subject of a given organization A wants to have an access to another organization B, this subject will issue a query. Organization B will apply the VPO A2B to check whether this query is authorized. This will generally require exchanging credentials between A and B for negotiating the access. If this negotiation phase succeeds, then the access will be granted.

Centralized VPO management:
In the centralized VPO management case, a VPO is both in the authority sphere of a given organization which is in charge of defining its interoperability policies and in the *manage-*

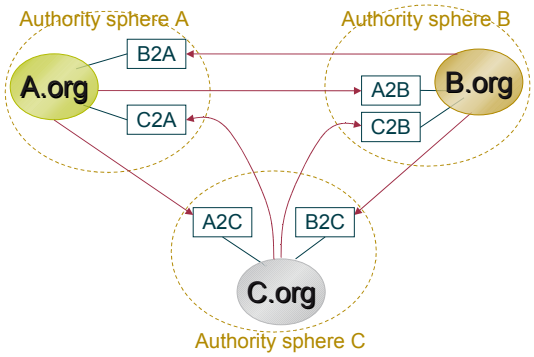

Fig. 1.7. Decentralized VPO Management.

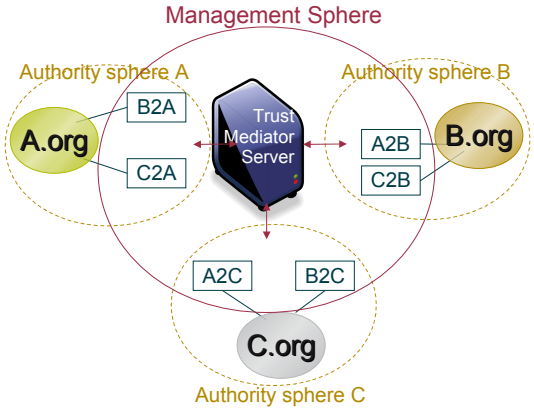

Fig. 1.8. Centralized VPO Management.

ment sphere of a server (see figure 1.8) which is in charge of managing all the interoperability policies of those organizations that trust this server. So, managing the VPOs is delegated to a unique trust server, which may be viewed as an extension of a CAS server (Globus toolkit)[23] or an advanced PEP, say PMP for Policy Management Point. Once a VO is created, each organization involved in this VO will have to send its VPOs to this server. When a subject s_A from a given organization A wants to have an access to another organization B, this subject must send its query to the server. The server will first authenticate this subject to get the proof that this subject is member of one of those organizations involved in the VO. Then the server will apply the VPO A2B and negotiate the access on behalf of organization B. If this negotiation succeeds, the server will sign the query so that the subject can then present this query to organization B for evaluation.

Hybrid VPO management:

The sensitivity of interoperation may vary. In the case of organizations (P2P or Server exchanges for instance) that deal with high sensitive information, assigning the task of managing the interoperability policies to a server may not meet the high confidentiality requirements of such organizations. When some Virtual Organization is created, organizations involved in this VO may not trust the server used in the Centralized VPO Management approach and/or may not accept to send their interoperability policies to this server because this may lead a leakage of some sensitive information and/or some of these organizations may not agree to interoperate with some organizations involved in this VO. In both cases, Hybrid VPO management may be used (see figure 1.9). In this figure, three organizations A.org, B.org and C.org agree to interoperate through Centralized VPO management, whereas the fourth organization D.org only accepts to interoperate with organization A.org using Decentralized VPO management.

In every approach, the interoperability policies are specified using the OrBAC model. The authority or the management sphere checks for each query(s,a,o) if a concrete permission for subject s to do action a on the object o can be derived from the specified VPO policies. The main advantage of the centralized management approach over the decentralized one is that, since the trust server has a global view of all the VPOs, it can manage possible conflicts between these VPOs.

1.10. AdOrBAC: interoperability policy administration

1.10.1. *AdOrBAC administration views*

Administration defines who is permitted to manage the security policy, that is to say who is permitted to create new security rules, or update or revoke existing security rules. Adminis-

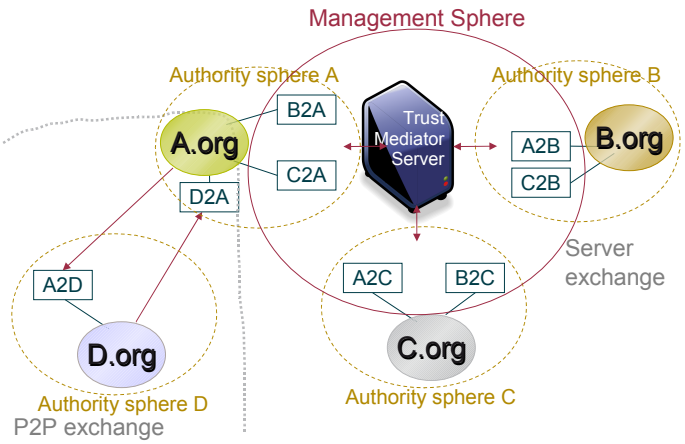

Fig. 1.9. Hybrid VPO Management.

tration is also used to define who is permitted to manage entities and their assignment, that is to say who is permitted to create, update or revoke existing entities and their relations with organization.

O2O is used to manage interoperation security policies. To secure the management of these interoperation security policies, we should have information on their administration. An information system has a great number of security rules and to satisfy integrity and security requirements, someone should manage the creation, updating and deletion of these rules. That is why information system should be administrated. In general, due to information system sizes, there are several policy designers. these policy designers have several administration privileges.

There are some administration requirements to be satisfied. In particular,

- we have to specify a structure that defines and/or administrates the security policy. In O2O, this structure is the organization $Ograntor$.
- we should have multi-grained privileges to limit the number of security rules and manage exceptions. The view *L*icense defined in the administration model of OrBAC can be used to achieve this goal.
- Administration should provide means to specify delegation. In usual administration models, delegation is specified using a separated model.

The administration model for OrBAC, AdOrBAC[24] is defined as an extension of the OrBAC model. AdOrBAC objectives are : (1) no separation between regular and administrative roles, (2) the self-administration, AdOrBAC uses same predicates as OrBAC to administrate OrBAC, (3) a multi-grain administration and (4) the enforcement of confinement principle. The confinement principle which limits the scope of privileges to organization, restricts the authority of a subject to the organization (or sub-organizations) to which this subject has been assigned administration privileges. The administration is achieved using administration views.

There are four types of administration views related to our security policies:

- $OEntities$: we have to manage organizational and concrete entities used to express security rules (subjects, actions, roles,...).
- $OEntity\_Assignement$: typically abstract entities have to be assigned to concrete entities (subject/role, action/activity).
- $Olicence$: contextual and fine grained privileges (permission, prohibition, obligation) are assigned using an administrative view called licence.
- $OEntity\_Hierarchy$: we have to control hierarchy relations, because hierarchies propagate privileges (role hierarchy,...).

When someone obtains a permission to insert an administrative object in one of the above administration views, he obtains the administrative permissions related to this view.

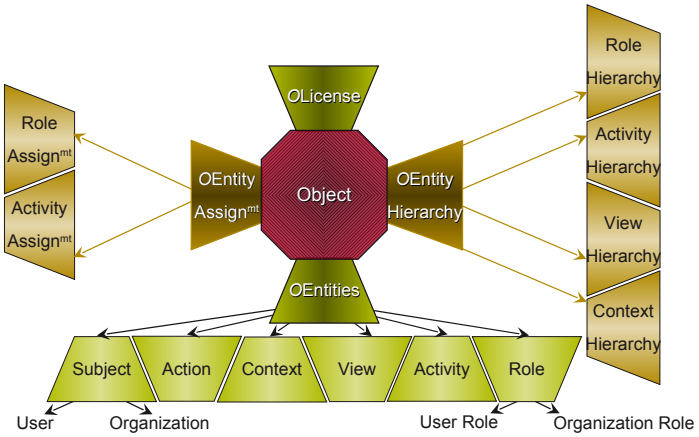

Fig. 1.10. AdOrBAC Administration Views.

1.10.2. *Licence*

In our approach, we use administrative access control object called *licence* (*cf.* the paper[24]). To specify administrative security policies, some subjects will be permitted to create, update or delete particular objects. These objects (licences) have a specific structure and meaning, namely the existence of a valid licence will be interpreted as the assignment of some permission.

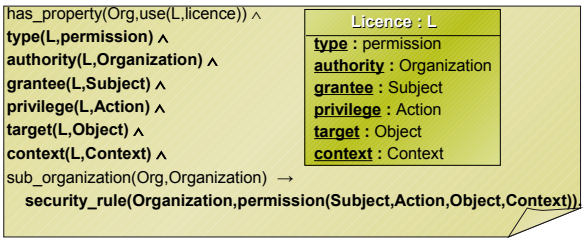

Fig. 1.11. Licence and Derivation Rule from Licence Attributes.

The notion of licence facilitates exportation of our security rule during exchanges. The license class, related to administration objects of *Olicence* view, is used to specify and manage the security policy. This class is associated with the three following attributes: (1) `authority`: organization in which the licence applies, (2) `grantee`: subject or role to which the licence is granted, (3) `privilege`: action or activity permitted by the licence, (4) `target`: object or view to which the licence grants an access and (5) `context`: specific conditions that must be satisfied to use the licence. Figure 1.11 shows the derivation rule related to a licence.

1.11. Privacy

During interoperability organizations should provide information on their security policies and some of these information are private and should not be known by non authorized external entities. So, we must provide a solution to protect the privacy of the organization when it enters an interoperability session. As our interoperability security policies are expressed using ontological languages and XML files,[25] we can deal with the privacy aspect through an obfuscation process of these security policies based on XML-BB principles[26] .

1.11.1. *XML-BB*

XML-BB model introduces the concept of blocks and provides means to protect relationships in the XML documents. XML-BB rest on two principles to build the authorized view associated with an XML document.

Let us consider that the security policy is an XML document represented by a tree and some nodes in this tree are connected by edges, two principles are used to build the authorized view associated with an XML document:

- `Cumulative principle`: The effect of two GRANT (*i.e.* permission) rules is cumulative, that is the resulting permissions actually correspond to the union of the nodes respectively selected by theses rules.
- `Connectivity principle`: If two selected nodes are connected by an edge in the XML document, then this edge will appear in the view presented to the user.

It is the combination of these two principles that sometimes fails to protect some information we need to hide. XML-BB is based on the new notion of block that avoids this problem. Intuitively, relationships between nodes selected in the same block are preserved whereas relationships between nodes selected in two different blocks are broken. Thus, an access control policy is specified using several blocks. In some sense, this is similar to the Chinese Wall security model suggested by Brewer-Nash.[27] In a Chinese Wall policy, a user may be permitted to have an access to different entities, but due to some conflict of interest between theses entities, this user is not permitted to aggregate this information. When we use two different blocks to specify an access control policy, we can in some sense consider that there is a wall between these two blocks. XML-BB allows us to express both a GRANT or REVOKE read access privilege and write privileges (insert, update and delete) on an XML file.

In the case of obfuscation, we use the same procedure to limit the access to security policy. There are three options to perform this obfuscation:

- `Propagate`: This option will select every sub-node of a selected node.
- `Shuffle`: This option will randomly shuffle the selected nodes. If a shuffled node is the root of a sub-tree, then this node will be the root of the same sub-tree after the shuffling operation.
- `Masquerade`: This option will replace the value of a selected node by the new

value specified in the option. This is used to hide the value of a given node without hiding the existence of this node.

1.11.2. *Obfuscation*

With XML-BB, we consider that an access control policy is defined as a set of blocks. As we have seen in §1.7.1, an ontology is composed of instances, ontological classes and properties. So, we have two types of blocks in an ontology: instance block and ontological class block. Due to the existence of these two kinds of blocks, according to XML-BB block concept, we have two ways to obfuscate security policies: the generic obfuscation and the specific obfuscation.

Generic obfuscation

This method is related to ontological class blocks. When we specify the type of access on a class block, we shall prohibit or permit access under some conditions to a whole set of instances. For example, our XML-BB approach may obfuscate any set of entities considered as internal objects and noncommunicable. Let us consider that during an interoperability session, we do not want to communicate information on concrete entities that cannot be used for this session. From the *Ograntor* point of view, all information related to object and action classes should be hidden. In this case, we will specify that we revoke privileges on object class block to an external subject using the fact: $REVOKE\ read, write\ /P\ ON = object\ TO\ externalSubject$. The /P means that the revocation is propagated to all the sub-nodes.

The advantage of generic obfuscation is that as a class is represented by a block with a unique path, we only have to find in the XML file the node related to this class. A simple XML file search algorithm based on depth-first search can be used to find the chosen node. Furthermore, we can distinguish ontological class block from the other blocks, as they begin by the tag `rdfs:Class` and finish by `/rdfs:Class`. Once, the block related to ontological class is founded, we have two cases: (1) The class does not have any subclass, in this case, we only have to hide all instances related to this class, that is done by propagation. (2) The class has some subclasses, in this case, we have to decide, if the obfuscation has to be propagated to the subclass or not. In last case, we have to distinguish (based on the tags) sub-blocks related to classes or to instances in the XML file. In that sense, we can define an exception rule to limit the propagation: $GRANT\ read\ /P\ ON\ /object/.\ WHERE\ rdfs : subClassOf := object\ TO\ externalSubject$. This requirement means that *externalSubject* is permitted to read sub-nodes of node *object* if they are subclasses of the class *object*. As noticed by the XML-BB's authors,[26] XML-BB algorithm applies the last matching principle; when several rules apply to the selected node, the last rule is applied.

Specific obfuscation

This method is related to instance blocks. It consists in hiding all blocks related to some information of a given entity. For instance, we can obfuscate all information on an identity if we hide all attributes related to this identity. We can also hide specific entities using

administrative objects. For instance, if we want to get some information on a role *R* in a security policy, we look for the relation *role* or for all administrative objects with role attributes such as *grantee*. So, if we want to revoke privileges related to role *Root* to external subjects, the following fact meets this security requirement: $REVOKE\ read, write\ /P\ ON\ /parent :: .\ WHERE\ grantee = Root\ TO\ externalSubject$. Structurally, in the XML file, instance blocks are sub-trees of the ontology. So, to take care about all information related to an instance, we should browse the different sub-trees and find all the sub-blocks related to some specified information. Then, depending on the obfuscation expression, we hide all information related to instance attributes by masquerading (specifying "*ON* /.:" on the current block, that is to say attribute) or we hide all instances related to some attribute description by propagation (specifying "*ON* /*parent* :: .:" on an upper block, that is to say instance).

1.12. Illustration

1.12.1. *P2P and interoperability*

As seen in the related works (*cf.* §1.2), most of the usual approaches rest on a centralized administration of policies to create collaborations, like in CAS server,[28] or are not sufficient to express fine grained access control. The increasing use of spontaneous networks like peer to peer[29,30] tends to require a decentralized administration of the collaboration resources. In that case, the security of this kind of interoperability environments remains a hard problem. As a matter of fact, peer-to-peer systems can be very dynamic and the peers' volatility is a barrier to a negotiation process setting during access control checking. Furthermore, P2P networks have many specific problems. For instance, in P2P systems, a peer is anonymous, so it is difficult to establish trust relationships. File sharing mitigates free-riding problem, but it is a spread vector of corrupted and malicious files. As suggested in this chapter, the O2O framework is based on the concept of contract that each interoperability policy of sub-organization has to comply with. This approach significantly contributes to secure the collaboration. In the following, we illustrate the interoperability policy derivation process using a P2P scenario.

1.12.2. *Obfuscation during interoperability*

In figure 1.12, we can see the composition of view *files*. The corresponding interoperability contract specifies a restriction on this view: $restrictionView(netpart1, files, sharingMovies)$. $netpart1$ wants to obfuscate unusable elements on *files* view. To achieve this goal, $netpart1$ defines the bloc $FILEStoP2Pnetwork$ and its access rules.

$BEGINBLOCKFILEStoP2Pnetwork$
$REVOKE\ read, write\ /P\ ON\ /file\ TO\ P2Pnetwork.$
$REVOKE\ read, write\ /P\ ON\ /music\ TO\ P2Pnetwork.$
$REVOKE\ read, write\ /P\ ON\ /persoFiles\ TO\ P2Pnetwork.$

GRANT read /P ON /file/sharingFiles/sharingMovies/. TO P2Pnetwork.
ENDBLOCKFILEStoP2Pnetwork

1.12.3. *P2P and O2O contract*

Let *Robert* be a new peer that joins the P2P network *peerNetwork*. *Robert* wants to get an access to the *Resident Evil* movie. So, First, *Robert* emits a search request to find this movie. The Distributed Hash Table indicates that two peers own the *Resident Evil* movie: P_1 and P_3. Then, *Robert* sends a resource request to P_3 to get an access to Resident Evil. A resource access request is composed of an interrogative license and can also contain the authorization proofs (credentials). The corresponding access request looks like the following:

$accessRequest(?number, @Robert, resourceAccess,$
$license[authority(L, @Dest), grantee(L, Robert), privilege(L, download),$
$target(L, ResidentEvil), context(L, default)]), @polRobert).$

where ?*number* is the identifier of the request, *grantee*, *privilege*, *target* and *context* are license information. $@polRobert$ is the URI of Robert's security policy. $@Robert$ is Robert's address. $@Dest$ is the grantor's address, in this case P_3 address.

P_3 contract specifies that there is $No\_compatibility$ type with a peer which has never exchanged resources in *peerNetwork*. Thus, *Robert* cannot download Resident Evil from P_3. This denial of access leads *Robert* to send a request to another peer P_1 which owns the same resource, the Resident Evil movie. P_1 contract specifies that there is a $T\_compatibility$ type with a peer pertaining to *peerNetwork*. So, to establish a VPO security rules between P_1 and *Robert*, we only have to establish role compatibility. To do so, *Robert* provides a RDF file of his security policy, which is easily exchangeable because it rests on XML. *Robert* can hide information which are not related to roles. Then, P_1 derives from its local security policy and this RDF file the VPO $Robert2P_1$. The security policy of VPO $Robert2P_1$ is created in a new RDF file and identified by the URI $@Robert2P_1$. Thus, P_1 can send the VPO file to a server; and *Robert* can consult the VPO at the address $@Robert2P_1$. In our example P_1 decides to restrict the access to Resident Evil according to legal condition related to the grantee country. So, the $lawfullyMovies$ context is then defined as the following:

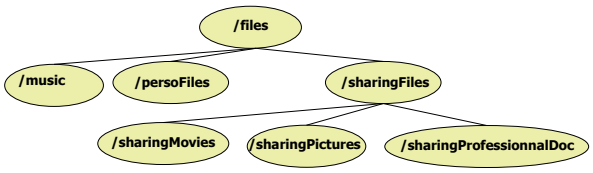

Fig. 1.12. Composition of *netpart*1 Files.

$$hold(Robert2P_1, lawfullyMovies(?S, download, ?Movie)) \leftarrow$$
$$ip\_adress(?S, ?ipAd) \wedge country(?ipAdr, ?Country) \wedge declared\_age(?S, ?AgeS) \wedge$$
$$lawMovie(?Age, ?Country, ?Movie) \wedge inferior(?Age, ?AgeS).$$

In France, children less than 12 years old are prohibited from watching Resident Evil movie. Thus, the management sphere, who is in charge of managing the VPO $Robert2P_1$, requires Robert to prove his age (for instance, via a credential). If this phase succeeds, then *Robert* can download Resident Evil from P_1.

1.13. Conclusion

To facilitate collaboration between organizations, conflict between security policy scopes should be detected and solved. Furthermore, interoperability requires reactivity, flexibility and continuity of service. In this chapter, we suggest a new approach to facilitate secure interoperation between organizations and preserving privacy property. This approach is defined as an extension of the O2O approach. In O2O, each organization administrates its resources so that it is always possible to know which security policy is applied to the resource. The objective of the suggested extension is to automatically derive interoperability security rules. This derivation depends on the following taxonomy of interoperability relations: total compatibility relation, partial compatibility relation, symmetric compatibility relation and no compatibility relation. These compatibilities can be refined with exception rules.

The main innovation of our approach is to provide a solution to anticipate over security requirements before the interoperability establishment. This anticipation is possible due to the abstraction of the whole security policy and its independency of the implementation. To interoperate securely and anticipate on this interoperation, we use the notion of contract. A contract which is defined by the grantor organization, specifies for each grantee organization, which grantor's resources or part of these resources are accessible. Moreover, as the interoperability security policies are derived from the grantor security policy, according to the nature of the contract, adaptation constraints of this security policy may be also specified. In this way, the grantor organization controls the accesses to its resources during interoperability sessions without weakening the security policies of the grantee organizations. An automatic derivation process of the interoperability policies based on these contracts has been defined. Since our approach to manage interoperability is defined as an extension of the OrBAC model, derivation of interoperability policies has actually similar complexity as derivation in the OrBAC model, namely polynomial. We are currently implementing this derivation process as an extension of MotOrBAC,[31] a toolkit that supports the specification of security policies using the OrBAC model.

References

1. F. Cuppens, N. Cuppens-Boulahia, and C. Coma. O2O: Virtual Private Organizations to Manage Security Policy Interoperability. In *Second International Conference on Information Systems*

Security (ICISS'06) (December, 2006).
2. R. Sandhu, D. Ferraiolo, and R. Kuhn, Role-based access control, *In American national standard for information technology: ANSI INCI TS 359-2004* (February 3. 2004).
3. O. et al., *The OrBAC Model Web Site*. 2006.
4. A. Abou El Kalam, R. E. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. In *4th IEEE International Workshop on Policies for Distributed Systems and Networks (Policy'03)* (June, 2003).
5. S. Cantor, J. Hodges, J. Kemp, and P. Thompson, *Liberty ID-FF Architecture Overview*. (Thomas Wason edition, https://www.projectliberty.org/resources/ specifications.php#box1, 2005).
6. R. Oppliger, Microsoft .net passport: A security analysis, *Computer*. **36**(7), 29–35, (2003). ISSN 0018-9162.
7. A. Pashalidis and C. J. Mitchell. A Taxonomy of Single Sign-On Systems. In *Lecture Notes in Computer Science*, vol. 2727, pp. 249 – 264 (Junary, 2003).
8. J. Li, N. Li, and W. H. Winsborough. Automated trust negotiation using cryptographic credentials. In *12th ACM Conference on Computer and Communications Security, CCS 2005*, pp. 46–57, Alexandria, VA, USA (November 7-11, 2005).
9. T. toolkit. Trustbuilder download: http://dais.cs.uiuc.edu/dais/security/trustb.php, (2003).
10. E. Bertino, E. Ferrari, and A. C. Squicciarini, Trust-X: A Peer-to-Peer Framework for Trust Establishment, *IEEE Trans. Knowl. Data Eng.* **16**(7), 827–842, (2004).
11. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, Role-Based Access Control Models, *Computer*. **29**(2), 38–47, (1996). ISSN 0018-9162.
12. E. Yuan and J. Tong. Attributed based access control (ABAC) for Web services. In *IEEE International Conference on Web Services (ICWS'05)* (July 11-15, 2005).
13. E. Bertino, E. Ferrari, and A. Squicciarini. X-TNL: An XML-based Language for Trust Negotiations. In *Fourth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'03)* (June 04-06, 2003).
14. D. L. Mcguinness and F. van Harmelen. Owl web ontology language overview. In *http://www.w3.org/TR/2004/REC-owl-features-20040210/* (February, 2004).
15. D. Connolly, F. van Harmelen, I. Horrocks, D. L. McGuinness, L. A. Stein, and L. Technologies. Daml+oil reference description. In *http://www.w3.org/TRdaml+oil-reference* (December, 2001).
16. R. Masuoka, M. Chopra, Z. Song, Y. K. Labrou, L. Kagal, and T. Finin. Policy-based Access Control for Task Computing Using Rei . In *Policy Management for the Web Workshop*, WWW 2005, pp. 37–43 (May, 2005).
17. A. Uszok, J. M. Bradshaw, M. Johnson, R. Jeffers, A. Tate, J. Dalton, and S. Aitken, Kaos policy management for semantic web services, *IEEE Intelligent Systems*. **19**(4), (2004).
18. C. Coma, N. Cuppens-Boulahia, F. Cuppens, and A. R. Cavalli. Context Ontology for Secure Interoperability. In *3rd International Conference on Availability, Reliability and Security (ARES'08)* (March 4-7, 2008).
19. F. Cuppens, N. Cuppens-Boulahia, and M. B. Ghorbel, High level conflict management strategies in advanced access control models, *Electronic Notes in Theoretical Computer Science (ENTCS)*. **186**, 3–26 (Jully, 2007).
20. H.-H. Do and E. Rahm. COMA - A System for Flexible Combination of Schema Matching Approaches. In *28th Conference on Very Large Databases (VLDB'02)* (August, 2002).
21. A. Doan, J. Madhavan, R. Dhamankar, P. Domingos, and A. Halevy, Learning to match ontologies on the Semantic Web, *The VLDB Journal*. **12**(4), 303–319, (2003). ISSN 1066-8888.
22. J. Madhavan, P. A. Bernstein, and E. Rahm. Generic Schema Matching with Cupid. In *27th International Conference on Very Large Data Bases (VLDB'01)*, pp. 49–58 (September 11-14, 2001). ISBN 1-55860-804-4.
23. globus.org. globus toolkit, http://www.globus.org/toolkit/, (2004).
24. F. Cuppens, N. Cuppens-Boulahia, and C. Coma. Multi-granular licences to decentralize security

administration. In *SSS/WRAS 2007: First international Workshop on Reliability, Availability and Security* (November 14-16, 2007).
25. T. Bray, J. Paoli, and C. Sperberg-McQueen, Extensible Markup Language (XML), *The World Wide Web Journal*. **2**(4), 29–66, (1997).
26. F. Cuppens, N. Cuppens-Boulahia, and T. Sans. Protection of Relationships in XML Documents with the XML-BB Model. In *Information Systems Security, First International Conference (ICISS'05)*, pp. 148–163 (December 19-21, 2005).
27. M. J. N. David F. C. Brewer. The Chinese Wall security policy. In *IEEE Symposium on Security and Privacy*, (1989).
28. L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A community Authorization Service for Group Collaboration. In *3rd international workshop on Policies for Distributed Systems and Networks (POLICY'02)*, Monterey, California, U.S.A (June 5-7, 2002).
29. R. Sandhu and X. Zhang. Peer-to-peer access control architecture using trusted computing technology. In *Proceedings of the tenth ACM symposium on Access control models and technologies (SACMAT'05)*, pp. 147–158, (2005).
30. H. Tran, M. Hitchens, V. Varadharajan, and P. Watters. A Trust based Access Control Framework for P2P File-Sharing Systems. In *38th Annual Hawaii International Conference on System Sciences (HICSS'05)*, Hawaii (January 03-06, 2005).
31. F. Autrel, F. Cuppens, N. Cuppens-Boulahia, and C. Coma. MotOrBAC 2: a security policy tool. In *Third Joint Conference on Security in Networks Architectures and Security of Information Systems (SARSSI'09)*, Loctudy, France (october 13-17, 2008).