POLIPO: Policies & OntoLogies for Interoperability, Portability, and autOnomy

Daniel Trivellato
Ei/Ψ Group, TU/e
d.trivellato@tue.nl

Fred Spiessens
Ei/Ψ Group, TU/e
a.o.d.spiessens@tue.nl

Nicola Zannone
Ei/Ψ Group, TU/e
n.zannone@tue.nl

Sandro Etalle
Ei/Ψ Group, TU/e
s.etalle@tue.nl

Abstract

In this paper we identify the requirements for the definition of a security framework for distributed access control in dynamic coalitions of heterogeneous systems. Based on the elicited requirements, we introduce the POLIPO framework that combines distributed access control with ontologies to give a globally understandable semantics to policies, enabling interoperability between autonomous and heterogeneous systems. In particular, we present a policy language and an architecture for policy evaluation. We also show that the framework is suitable for the specification and remote evaluation of portable policies.

1 Introduction

In the context of the POSEIDON project[1] we are developing a security framework for distributed access control in a System of Systems (SoS) [15], a system whose components are highly complex systems in themselves. POSEIDON is a joint project of a consortium of industrial (Thales Netherlands[2] and Noldus[3]) and academic partners. Its goal is to enable flexibility, interoperability and adaptability in an SoS, while ensuring reliability and security. The SoS model is suitable to represent and analyze the Maritime Safety and Security (MSS) domain which is the focus of POSEIDON.

As many other domains involving SoS (e.g., air traffic control and Internet), the MSS domain is characterized by the interaction and collaboration between autonomous and heterogeneous systems sharing information, processes, and resources. These collaborations are not fixed a priori, but can dynamically change over time as new parties join, leave, or change their responsibilities and objectives within the SoS. An example of SoS in the MSS domain is represented by a coalition of multinational vessels, such as the task force created by the European Union to protect commercial shipping off the Somali coast against piracy (Operation Atalanta[4]).

The SoS model offers several advantages: it eliminates the physical boundaries of organizations and increases operational flexibility. However, it has a strong impact on interoperability as well as on the security requirements of collaborating parties. Sharing sensitive information with other (possibly unknown) parties may be required for the success of the coalition. For example, a commercial ship having an engine failure off Somali coast can communicate its position to the EU task force for aid. Surveillance vessels, however, may employ different communication models and thus are not able to interpret the distress message. Moreover, the message can be intercepted by pirates that can exploit this emergency situation to attack the ship.

The development of authorization mechanisms for distributed systems where resources are accessed across system boundaries has become a major research challenge. Several trust management frameworks have been proposed to address this problem [1, 2, 3, 13, 14, 18]. In these frameworks, access control decisions are based on credentials. Credentials are certificates that attest that a subject has a certain attribute, and are digitally signed to ensure their authenticity and integrity.

However, using only credential for access control is not effective, because the data owner loses the control on the data after their disclosure. To allow the data owner to impose constraints on the usage and redistribution of data to the data recipient, distributed access control also requires portable policies [5]. Parties can attach policies (hereafter *sticky policies*) to their own data, that are evaluated and enforced remotely at the side of the data recipient.

When heterogeneous systems form dynamic coalitions that transgress the traditional boundaries between organizational, cultural, and legal units, they face interoperability concerns. The quest for interoperability has led to approaches for terminology and semantic alignment. Ontologies have been largely used in the Semantic Web to address these issues. Ontologies provide a means for establishing common vocabularies and capturing domain knowl-

---

edge. These properties have spurred researchers to use ontologies for the specification of security policies [22]. However, the expressive power of these policies is restricted to the one of ontology languages. To overcome this limitation, ontologies have been extended with rules [9], but this extension often causes ontology reasoning to become undecidable [8, 19].

This paper proposes the POLIPO (Policies & OntoLogies for Interoperability, Portability, and autOnomy) framework to address the problem of distributed access control in an SoS. In particular, we make the following contributions:

- We identify the requirements for the definition of a security framework for distributed access control in dynamic coalitions of heterogeneous systems.
- We propose a policy language that combines access control and trust management for the specification of security policies in an SoS. Access control policies are evaluated locally by a party, but the use of credentials makes it possible to escape from the boundaries affecting centralized access control models.
- We enable interoperability between heterogeneous systems, by applying ontology-based semantic alignment to distributed access control. Ontologies are used as a common vocabulary: the attributes certified in a credential and the actions characterizing a permission are defined in an ontology shared by the members of the coalition.
- We combine policy rules with ontologies to improve the query answering support to the knowledge base. This makes it possible to exploit the reasoning services offered by ontologies. However, we do not extend ontologies with rules to avoid undecidability problems. In the POLIPO framework, ontologies are used as remote oracles to infer domain knowledge.
- We discuss an architecture and an implementation of the POLIPO framework.
- We enable control over data after their disclosure using sticky policies. In particular, we study the portability of POLIPO policies and identify the restriction in the language that make policies understandable to remote parties for evaluation and enforcement.

The remainder of the paper is organized as follows. Section 2 motivates our work in the context of the Maritime Safety and Security domain. We discuss the requirements for the policy language in Section 3. We present the POLIPO framework in Section 4, and discuss the restrictions for sticky policies in Section 5. Finally, we discuss related work in Section 6, and conclude in Section 7.

2 Motivation Example

In the Maritime Safety and Security (MSS) domain, collaborating systems differ in size and composition, from the loosely coupled subsystems onboard a ship (e.g., operational communication, navigation, etc.) to bigger organizational units like a fleet of which the ship is part. MSS coalitions also include collaboration with other types of systems such as the PDA of a police officer inspecting a cargo ship, a coast surveillance center, a maritime weather station, an insurance broker, a fisher boat, a surveillance satellite, etc.

Even if a strong integration of such diverse systems would be feasible, the result would be unacceptably rigid and lack the flexibility to adapt quickly to new situations without requiring the SoS to be completely re-engineered. In the POSEIDON context, coalitions are expected to react rapidly and adequately to unforeseen changes such as systems joining and leaving the coalition or being modified.

The members of an MSS coalition need to exchange data of different type, structure, origin, reliability and level of confidentiality. For instance, streams of AIS-data (Automatic Identification System) sent by vessels, containing location and destination data, can be combined with privacy sensitive data from commercial institutions and publicly available information from the Internet, to be analyzed by the harbor authorities who might forward the information to a coast guard ship. The MSS domain requires a trustworthy collaborative environment where data can be exchanged and processed by different parties securely.

Each SoS component is an autonomous system that has the right to protect its resources and regulate access to its sensitive data according to its own security policy. When data are sent to other SoS components, security policies can be attached to the data to regulate their redistribution and usage. For instance, the harbor authority could specify that the data it forwards to the coast guard ship is meant for disclosure to officers only.

The problems of terminology and semantic alignment that arise in these coalitions extend to security policies. For instance, a security policy may use "words" whose semantics differs among the collaborating parties. Ontologies modeling the MSS domain can be used to disambiguate not only the data but also the policies that are exchanged between systems. This would allow the coast guard ship to understand what "for officers only" actually means.

Figure 1 shows a simple scenario in which the components joining a coalition use a shared ontology (hereafter POSEIDON ontology) to communicate with other parties in the coalition. These parties can (re)use the concepts (plain rectangles) and relationships (plain diamonds) in the POSEIDON ontology, as well as extend the ontology with their own concepts and relationships (dashed parts). In the context of the POSEIDON project, ontologies are used to model not only the abstract concepts and relationships characterizing the MSS domain but also the concrete entities and data objects.
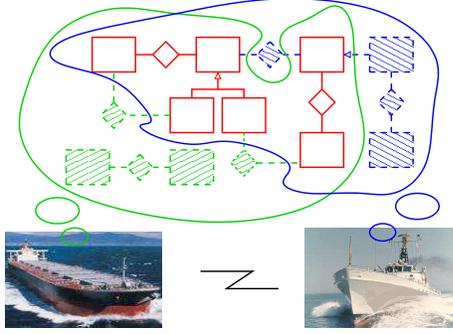
Figure 1. Communication Using Ontologies

(1)	SeniorOfficer ⊑ Officer JuniorOfficer ⊑ Officer
(2)	TemporaryOfficer ⊑ SeniorOfficer TemporaryOfficer ⊑ JuniorOfficer
(3)	JuniorOfficer ⊑ ¬ SeniorOfficer
(4)	StarNavy ⊑ AlliedNavy

Figure 2. Sample Ontology

3 Requirements

Several policy specification frameworks have been proposed to address authorization problems in distributed systems [1, 2, 3, 5, 13, 14, 18]. These proposals, however, only provide a partial solution (see a discussion in Section 6). We argue that a practical policy framework should not only support autonomy of the policy owner but also provide policy interoperability and portability. In this section, we clarify and discuss the requirements that have driven the design of the POLIPO framework.

Requirement 1 (Autonomy) *Every party shall be able to design and express its policy autonomously.*

This means that every party can specify its policies using the concepts of its choice, regardless of which parties have already joined the coalition before. While this requirement may seem obvious and trivial to implement, we state it explicitly here because it has implications that can easily conflict with other requirements.

Requirement 2 (Interoperability) *Parties shall be able to interact with each other unambiguously.*

As security policies have to be understood by multiple parties, the problem of semantic interoperability emerges. To tackle this problem, we turn to the established practice of ontologies. An ontology is a formal representation of a domain in terms of concepts and relationships, each with a precise semantics. A party can thus refer to an ontology to denote their semantics unambiguously. Some ontologies can be expressed using description logics which support non-monotonic constraints (e.g., disjointness of concepts). This implies that extending an ontology can introduce inconsistencies.

Example 1 *The POSEIDON ontology states that senior officers and junior officers are officers (part 1 in Fig. 2). A new party joining the coalition extends the ontology by*

defining temporary officers, which can be both senior officers and junior officers (part 2 in Fig. 2). Furthermore, it assigns the role of temporary officer to some of its crew members, to face emergency situations. Now, assume that another party joins the coalition and constraints junior officers to be disjoint from senior officers (part 3 in Fig. 2). This makes the POSEIDON ontology inconsistent because of the definition of temporary officer.

When using a single shared ontology that can be extended by all parties in the coalition, consistency can conflict with autonomy. We cannot simply allow parties to modify the shared ontology by adding their own local concepts and constraints, because this would introduce limitations that are based on the order in which the parties join the coalition, which violates the autonomy requirement.

Example 2 *Take the initial situation in part 1 of Fig. 2 and suppose two parties want to join the coalition and add their constraints to the global ontology. One party wants to add part 2 while the other party wants to add part 3. Whatever party joins the coalition first will be able to make its adaptations without making the ontology inconsistent. But thereafter, extending the ontology for the other party would be illegal.*

The example above shows that demanding a shared ontology to be consistent would conflict with autonomy. For this reason, we simply require local consistency which is also acceptable in the context of the POSEIDON project. When a party joins the coalition, it imports the POSEIDON ontology into its local ontology, and possibly reuses and extends some global concepts. Every party should keep its local ontology consistent. However, the extensions made to the imported ontology stay locally inside the ontology of a certain party and, therefore, cannot affect the consistency of the ontologies of other parties.

Requirement 3 (Portability) *Remote evaluation of policies shall preserve the interpretation of the policy owner.*

Policies can be transferred to other parties, with the intention of restricting the usage and redistribution of data. Typically, sticky policies are attached to the data they protect, following them from system to system, and are evaluated and enforced remotely. When an access request is

processed remotely, the data owner wants to be sure that the sticky policy is evaluated according to his own interpretation. Thereby, the data owner has to be careful when designing a policy, to preclude illegitimate flows of information. In this setting, the use of credentials allows the data owner to maintain more control over remote access to data.

Example 3 *The Red Star Navy allows senior officers of allied navies to access some sensitive information, where allied navies include Star navies according to the POSEIDON ontology (part 4 of Fig. 2). Senior officers of the Blue Star Navy are thus allowed to access the information. However, the Blue Star Navy also includes the Black Cross Navy among its allies in its local ontology. If the policy is evaluated locally within the Blue Star Navy, senior officers of the Black Cross Navy are granted permission to access the information, against the intention of the Red Star Navy which does not have the Black Cross Navy among its allies. To prevent this disclosure of information, the Red Star Navy can specify that only senior officers of navies having an 'AlliedNavy' credential signed by the Red Star Navy itself are entitled to access the information.*

Moreover, data recipients need to avail of all necessary information to evaluate the policy correctly. Therefore, the data owner must make sure that sticky policies are self-contained as far as their evaluation is concerned.

4 The POLIPO framework

In this section we present POLIPO (Policies & OntoLogies for Interoperability, Portability, and autOnomy), a security framework that combines distributed access control with ontologies to enable interoperability, portability, and autonomy in dynamic coalitions of heterogeneous systems. In particular, we introduce a logic-, ontology-based language for the specification of distributed access control policies and discuss an architecture for their evaluation.

4.1 Syntax and Semantics

POLIPO policies are specified using four constructs:

- *ontology atoms*: are used to query the knowledge base represented by ontologies. Each concept in an ontology is identified by a $conceptURI$. $conceptURI(a)$ holds if a is an instance of $conceptURI$. Each relationship in an ontology is identified by a $relationshipURI$. $relationshipURI(a_1, a_2)$ holds if instance a_1 is related to instance a_2 via $relationshipURI$. Examples of ontology atoms are given in part 1 of Fig. 3: $psd{:}SeniorOfficer$ is a $conceptURI$ and $psd{:}worksFor$ is a $relationshipURI$. These atoms

(1)	$psd{:}SeniorOfficer(\text{'}John\text{'})$ $psd{:}worksFor(\text{'}John\text{'}, \text{'}RS\text{'})$
(2)	$\mathsf{cred}(\text{'}BS\text{'}, \text{'}psd{:}SeniorOfficer\text{'}, \text{'}John\text{'},$ $[[(\text{'}psd{:}ValidUntil\text{'}, \text{“}31/12/2009\text{”})],$ $[(\text{'}psd{:}WorksFor\text{'}, \text{'}RS\text{'})]])$
(3)	$\mathsf{perm}(\text{'}psd{:}read\text{'}, \text{'}John\text{'}, \text{'}File\text{'})$
(4)	$X = Y + 3$ $X \geq Y$
(5)	$aboutSurveillance(\text{'}File\text{'})$

Figure 3. Examples of POLIPO Atoms

check whether '$John$' is a senior officer and if he works for the Red Star Navy (RS is the unique identifier of the Red Star Navy) according to the definitions given in the POSEIDON ontology. Consistently with the XML (and OWL) convention, we use prefixes followed by symbol ":" to substitute base URIs (e.g., *psd:* stands for *www.example.net/poseidon/PSD-Ontology/*).

- *credential atoms*: represent digitally signed statements made by an issuer about an attribute of a subject. A credential atom has the form:

$$\mathsf{cred}(issuer, att, subject, [[c_1, \ldots, c_n], [a_1, \ldots, a_n]])$$

where $issuer$ is the unique name of the entity signing the credential; att is a $conceptURI$ specifying the attribute for which the credential is issued; $subject$ is the unique name of the entity to whom the credential refers; $[c_1, \ldots, c_n]$ and $[a_1, \ldots, a_n]$ are lists of optional properties. Each c_i (respectively a_i) is a pair $(property, value)$, where $property$ is a $relationshipURI$ related to the the concept credential[5] (resp. to att), and $value$ bounds the range of the relationship. Part 2 of Fig. 3 shows a credential issued by the Blue Star Navy (BS) and certifying the fact that '$John$' is a senior officer of the Red Star Navy. The credential has validity period as an optional property.

- *authorization atoms*: denote the permission of a subject to perform an action on an object. They have form:

$$\mathsf{perm}(action, subject, object)$$

where $action$ is a $relationshipURI$ specifying the action that $subject$ is allowed to perform on $object$; $subject$ is the unique name of the entity to whom the permission is granted; $object$ represents the target of the permission. An example of authorization atom is shown in Fig. 3 part (3), where '$John$' is given the permission to read '$File$'.

- *constraints*: are specified using Constraints Logic Programming (CLP) [16] constraints (e.g., $=, >, <$, etc.) or user-defined predicates (respectively parts 4 and 5 in Figure 3).

---

[5]cred and perm are formally defined in the POSEIDON ontology.

Credential Release Rule
cred('BS', '$psd{:}SeniorOfficer$', X, []) $\leftarrow$ $psd{:}SeniorOfficer(X)$
Authorization Rule
perm('$psd{:}read$', X, Y) $\leftarrow$ $aboutSurveillance(Y)$,
cred('BS', '$psd{:}SeniorOfficer$', X, [])
Predicate Definition Rule
$aboutSurveillance(X) \leftarrow bs{:}aboutMission(X, \text{'}Surveillance\text{'})$,
$bs{:}sensitivityLevel(X, Y), Y < 3$

Figure 4. Examples of POLIPO Rules

We formalize POLIPO policies in the logic programming paradigm. POLIPO rules are Horn clauses of the form $h \leftarrow b_1, \ldots, b_n$, where h, called head, is an atom, and $b_1, \ldots, b_n$, called body, are literals (i.e., positive or negative atoms) with $n \geq 0$. Negation is treated as negation-as-failure (denoted by not): if there is no evidence that an atom is true, it is considered to be false. We also assume that each variable occurring in the head of a rule, in a negative literal, as issuer of a credential atom, or in a CLP constraint also occurs in at least one positive literal in the body of the same rule (safety condition).

We distinguish three types of rules: *credential release rules*, *authorization rules*, and *predicate definition rules*.

Definition 1 (Credential Release Rule) *A credential release rule is a Horn clause where the head is a credential atom and the body can contain positive credential and ontology atoms, and constraints.*

Example 4 *The first rule in Fig. 4 states that the Blue Star Navy releases a $psd{:}SeniorOfficer$ credential to entities defined as senior officers in the POSEIDON ontology.*

It is worth noting the difference in the use of $psd{:}SeniorOfficer$ in the example. When it occurs as the predicate name in the ontology atom, it refers to the knowledge base of the Blue Star Navy. When it occurs in the credential atom, it is the attribute of the subject certified by the Blue Star Navy.

Definition 2 (Authorization Rule) *An authorization rule is a Horn clause where the head is an authorization atom and the body can contain positive credential, authorization, and ontology atoms, constraints, and negative ontology and user-defined atoms.*

Example 5 *The second rule in Fig. 4 states that subject X is authorized to read object Y if Y is about surveillance as defined by the policy owner and X provides a credential issued by the Blue Star Navy stating that he is a senior officer.*

We only allow negation-as-failure of ontology and user-defined atoms in the body of authorization rules for two reasons. First, we restrict non-monotonicity to predicates in the knowledge base. This guarantees that POLIPO policies

are stratified logic programs, ensuring efficiency and unambiguity. Second, there is a semantic difference between credential release rules and authorization rules. Authorization rules are evaluated to determine whether a permission should be granted or not at a certain instant; changes in the truth value of an atom in the body only affect future access decision, without impacting past decisions. On the contrary, once a credential is issued, it is valid for a period of time.

Definition 3 (Predicate Definition Rule) *A predicate definition rule is a Horn clause where the head is a user-defined atom and the body can contain positive ontology atoms and constraints.*

Example 6 *The third rule in Fig. 4 defines the user-defined predicate aboutSurveillance. An object is about surveillance if it concerns mission 'Surveillance' and has sensitivity level less than 3 according to the Blue Star Navy.*

We have limited the occurrence of ontology atoms to the body of POLIPO rules. This is to keep the policy and ontology reasoning separated. Indeed, by allowing a free interaction of ontologies with rules (i.e., using rules to modify the semantics of ontologies) the ontology reasoning may becomes undecidable [8, 19].

POLIPO policies consist of sets of POLIPO rules. In particular, we distinguish two types of policies: *credential release policies* and *authorization policies*.

Definition 4 (Credential Release Policy) *A credential release policy is a set of credential release rules.*

Definition 5 (Authorization Policy) *An authorization policy is a set of authorization rules.*

4.2 Architecture and Implementation

Figure 5 shows a high level overview of the architecture of the POLIPO framework. Requests for credential release and data access are sent to the policy enforcement point (PEP). The PEP forwards the request to the policy decision point (PDP) together with the information necessary to evaluate the request. By now, we abstract from the problems of credential discovery and trust negotiation and assume that the requester provides all the credentials necessary for the evaluation of the policy.

Depending on the type of the request, the PDP either consults the access control policy or the credential release policy to make a decision. A predicate layer supports the evaluation of policies by interfacing with the local ontology (that imports the global ontology) or inferring user-defined predicates using predicate derivation rules. Finally, the PEP enforces the decision of the PDP, providing the access to the data or releasing the requested credential.
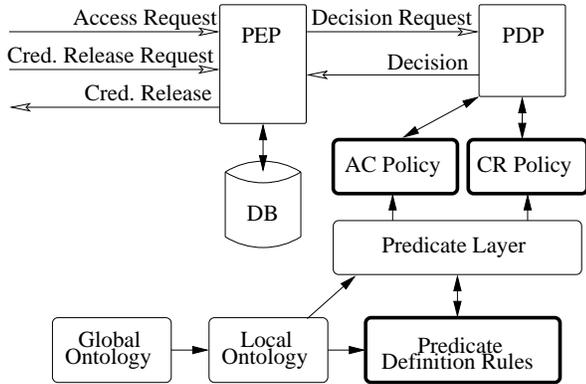
Figure 5. Component Architecture

$$\mathsf{perm}(`psd{:}read", X, `File") \leftarrow psd{:}SeniorOfficer(X),$$
$$psd{:}worksFor(X, Y),$$
$$cred(`RS", `psd{:}AlliedNavy", Y, []),$$
$$Y = yourself()$$

Figure 6. A Sticky Policy Using *yourself* $()$

We have adopted SWI-Prolog to implement a prototype of the PDP. The main reason for using SWI-Prolog is that it provides an interface with ontologies, through the Semantic Web Library.[6] This library consists of packages for reading, querying and storing RDF documents, and hence ontologies (every OWL ontology can be represented in RDF). For example, the ontology atom $psd{:}SeniorOfficer(`John")$ can be expressed using the built-in predicate *rdf(John,rdf:type,SeniorOfficer)*, where *rdf:* is the prefix of the URI where relationship *type* is defined. The Semantic Web Library allows the specification of other types of queries based, for instance, on subclass relationships. This can be used for the definition of metapolicies for policy management. For example, if an issuer can release a credential with an attribute *SeniorOfficer*, then it can also release credentials for superclasses of *SeniorOfficer* without naming them explicitly. The downside of this implementation choice is that SWI-Prolog does not automatically deal with loops. Discussing our solution for handling recursive credentials is outside the scope of this paper and will be the subject of a whole paper devoted to the algorithms of POLIPO.

5 Sticky Policies

The data owner can exploit the POLIPO framework to make sure that data are accessed only by the coalition parties he trusts. However, a party may still want to maintain some control over its data after their disclosure. We adopt sticky policies to allow the data owner to impose constraints on the remote usage and redistribution of data. The data owner is likely to trust that the recipient enforces those constraints, since parties who are trusted to access the data are also likely to be trusted to comply with the attached policy.

Sticky policies are expressed using the syntax given in Section 4, but they have some specific needs and restric-

tions because of the autonomy, interoperability and portability requirements. This section discusses these needs and restrictions.

Referencing the Policy Evaluator   It is important that the creator of the sticky policy (usually the data owner) is able to reference the entity that will evaluate the sticky policy. However, the policy creator may not know the identity of the evaluator a priori. For this purpose, we introduce the nullary function $yourself()$ which returns the identity of the entity evaluating the sticky policy. The following example shows the use of $yourself()$ in a rule.

Example 7 *As in Example 3, the Red Star Navy allows its data to be distributed among its allies, but now with the restriction that each allied navy should only disclose the data to its own senior officers. The Blue Star Navy can disclose the data to Blue Star senior officers, and forward the data to the Green Star Navy (also an ally of the Red Star Navy). However, the Blue Star Navy is not supposed to disclose the data directly to senior officers of the Green Star Navy, for reasons of trust and/or accountability. Figure 6 shows this sticky policy: the read permission can be given to a senior officer X of an ally Y of the Red Star Navy only by ally Y, where Y is the actual navy evaluating the policy.*

Predicate Restrictions   Sticky policies can contain only predicates that refer to the global ontology. Predicates from a local ontology are not interpretable by other parties, as neither the data owner nor the receiver are supposed to (be able to) access each other's local ontology. The fact that global concepts are imported into the local ontologies still allows for the necessary autonomy, as parties may have different interpretations of global concepts, but the use of the global ontology as a shared vocabulary provides a common base for interoperability.

Rule Restrictions   We do not allow the occurrence of credential release rules in sticky policies. Sticky policies are meant to protect data, and should not interfere with the evaluator's credential management for two reasons. First, every party can only issue credentials that are signed by itself. Suppose that the policy owner includes a credential release rule in a sticky policy. This would require the policy evaluator to sign credentials in behalf of the data owner; to do so,

---

[6]http://www.swi-prolog.org/packages/semweb.html

he has to know the private key of the policy owner. This situation must be avoided as the data owner "loses" his identity and thus his autonomy, besides raising accountability issues. Second, the policy owner cannot impose the policy evaluator to issue credentials (signed by the evaluator itself) because of the autonomy requirement.

Policy Restrictions   A sticky policy must be sent with all the predicate definition rules needed to evaluate the policy. Remote PDPs can evaluate a sticky policy only if they have all the information necessary for its evaluation. POLIPO policies can have rules in which user-defined predicates occur in the body, and remote PDPs should be able to evaluate such predicates that are not globally understandable.

6   Related Work

In recent years, more and more effort has been invested in the specification and enforcement of access control policies in distributed systems. For instance, Li et al. [14] introduce the RT framework, a family of role-based trust management languages. *RT* addresses the issue of vocabulary alignment using Application Domain Specification Documents (ADSDs). An ADSD defines a vocabulary that specifies data types and role names, and it is globally uniquely identified through a scheme inspired by XML namespaces. Czenko et al. propose TuLiP [3], a credential-based, role-based trust management system that uses XML to represent credentials. Similarly to RT, TuLiP uses XML namespaces to avoid name conflicts and facilitate the definition of a common vocabulary. However, these approaches are purely syntactical and do not provide the semantics, expressiveness and reasoning facilities that ontologies do. In *PeerTrust* [18] policy rules are definite Horn clauses, whose predicates can be external procedure calls. This allows one to import RDF metadata, enabling the use of ontology predicates in policy specifications. Nonetheless, the authors do not investigate how the ontology knowledge is integrated into the inference reasoning. Many other trust management frameworks have been proposed (e.g, [1, 2, 13]), but they do not explicitly address the problem of global unique names. A drawback of trust management systems is that the policy portability requirement is neglected. This issue is tackled by sticky policy languages [5, 11, 23], but those approaches do not exploit ontologies for facilitating the interpretation of policies when evaluated remotely.

In the attempt to enhance the Semantic Web with security policies, some languages combine ontologies with policy rules. KAOS [22] entirely specifies policies in terms of Description Logic (DL). This, however, restricts the expressive power to DL. Rei [10] allows the specification of policies both in Prolog and RDF (ontologies). Prolog rules use predicates from ontologies, but they can also include logic-like variables to exploit variable unification. However, by adding variables it is no longer possible to benefit from all DL reasoning services (e.g., static conflict detection), because rule knowledge must be treated separately from ontology knowledge. Finin et al. [6] express RBAC policies in OWL to exploit DL reasoning for authorization decision making. Nevertheless, OWL policies are not expressive enough to represent all type of security constraints (e.g., separation of duty) and need to be complemented with rules. Finally, Toninelli et al. [21] suggest a hybrid approach between ontologies and rules to capture frequent changes in dynamic settings. They suggest to adopt DL to classify context information and policies and to use a rule-based approach for the enforcement of policies defined over dynamically determined values. However, the authors do not provide any practical solution to address the problem.

The major problem of integrating ontologies and rules stems from the fact that adding linear structures (e.g., role-value maps) to ontologies causes DL reasoning to become undecidable [8]. An example is given by SWRL [9] that extends OWL with Horn clauses. This limitation has spurred researchers to investigate syntactic restrictions that keep DL with rules decidable. Grosof et al. [7] define a language that is the intersection of DL and Horn logic programs, called Description Logic Programs (DLP). Being the intersection of the two, DLP is less expressive of both DL and Horn logic programs. The *SweetProlog* [12] system translates web rules into Prolog to build a rule layer on top of ontologies, offering enhanced representation and reasoning capabilities. The main limitation of *SweetProlog* is its expressive power as it uses DLP to enable the integration between ontology and rules. Another system layering rules on top of ontologies is *SWORIER* [20], a query answering engine which translates RuleML rules and OWL ontologies into Prolog programs. Motik et al. [17] preserve the decidability of the combination of ontologies and rules by restricting rules to DL-safe rules. In other words, every instance of ontology concepts and properties must be present as a ground fact in the logic program, to perform close-world reasoning. Similarly, Rosati [19] guarantees decidability by enforcing Datalog safeness (i.e, each variable must appear in a positive body atom) and weak safeness (i.e, every variable in the head of a rules must appear in a non-DL body atom). These restrictions, however, would result in a limitation of the expressive power that is not acceptable in context of the POSEIDON project.

Our approach relates well to the proposal by Eiter et al. [4] who focus on exploiting the knowledge contained in ontologies in logic programs, rather than integrating the two frameworks. The difference lies in the fact that we do not allow the flow of information from the logic program to the ontology. Eiter et al. use the knowledge inferred by rules to temporarily feed the ontology for further reasoning. This,

however, can introduce inconsistencies in the ontology.

7 Conclusions and Future Work

This paper has discussed the requirements for the definition of a security framework for distributed access control in the context of the POSEIDON project. Based on the elicited requirements, we have proposed POLIPO, a framework that combines access control, trust management and ontologies to guarantee interoperability and autonomy of entities. We have also studied the restrictions that make the language suitable for the specification and remote evaluation of portable policies.

The work is still in progress to increase the flexibility and expressive power of the language, preserving the interoperability, autonomy, and portability requirements. In this paper, we have assumed that every party in the coalition uses concepts and relationships from the shared ontology when communicating with other parties. However, there may be cases where a party specifies local concepts that are similar but not equivalent to the ones defined in the shared ontology. We are investigating a method to relax the assumption by allowing parties to specify policies using local concepts and relationships in such a way that policies can be still understood and evaluated correctly by other parties. Moreover, sticky policies may conflict with the local policies of the remote evaluator. Therefore, methods for policy combination are needed to solve such conflicts. Finally, in this work, sticky policies mainly focus on the redistribution of data to other parties in the coalition. We are extending the language to fully support data usage control.

References

[1] M. Y. Becker and P. Sewell. Cassandra: Distributed access control policies with tunable expressiveness. In *Proc. of POLICY'04*, pages 159–168. IEEE Computer Society.

[2] P. Bonatti and P. Samarati. Regulating service access and information release on the web. In *Proc. of CCS'00*, pages 134–143. ACM, 2000.

[3] M. Czenko and S. Etalle. Core TuLiP Logic Programming for Trust Management. In *Proc. of ICLP'07*, LNCS 4670, pages 380–394. Springer, 2007.

[4] T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Well-founded semantics for description logic programs in the semantic web. In *In Proc. of RuleML'04*, LNCS 3323, pages 81–97. Springer, 2004.

[5] S. Etalle and W. H. Winsborough. A posteriori compliance control. In *Proc. of SACMAT'07*, pages 11–20. ACM, 2007.

[6] T. W. Finin, A. Joshi, L. Kagal, J. Niu, R. S. Sandhu, W. H. Winsborough, and B. M. Thuraisingham. R*WL*BAC: representing role based access control in *WL*. In *Proc. of SACMAT'08*, pages 73–82. ACM, 2008.

[7] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: combining logic programs with description logic. In *Proc. of WWW'03*, pages 48–57. ACM, 2003.

[8] I. Horrocks, P. F. Patel-Schneider, S. Bechhofer, and D. Tsarkov. OWL rules: A proposal and prototype implementation. *JWS*, 3(1):23–40, 2005.

[9] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission, World Wide Web Consortium, 2004.

[10] L. Kagal, M. Paolucci, N. Srinivasan, G. Denker, T. Finin, and K. Sycara. Authorization and Privacy for Semantic Web Services. *IEEE Intelligent Systems*, 19(4):50–56, 2004.

[11] G. Karjoth, M. Schunter, and M. Waidner. Platform for Enterprise Privacy Practices: Privacy-Enabled Management of Customer Data. In *Proc. of PET'02*, LNCS 2482, pages 69–84. Springer, 2002.

[12] L. Laera, V. A. M. Tamma, T. J. M. Bench-Capon, and G. Semeraro. SweetProlog: A System to Integrate Ontologies and Rules. In *Proc. of RuleML'04*, LNCS 3323, pages 188–193. Springer, 2004.

[13] N. Li, B. N. Grosof, and J. Feigenbaum. Delegation logic: A logic-based approach to distributed authorization. *ACM Trans. Inf. Syst. Secur.*, 6(1):128–171, 2003.

[14] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a Role-Based Trust-Management Framework. In *Proc. of Symp. on Security and Privacy*, pages 114–130. IEEE Computer Society, 2002.

[15] M. W. Maier. Architecting principles for systems-of-systems. *Systems Engineering*, 1(4):267–284, 1999.

[16] K. Marriott and P. J. Stuckey. *Programming with constraints: an introduction*. MIT Press, 1998.

[17] B. Motik, U. Sattler, and R. Studer. Query Answering for OWL-DL with Rules. *JWS*, 3(1):41–60, 2005.

[18] W. Nejdl, D. Olmedilla, and M. Winslett. PeerTrust: Automated Trust Negotiation for Peers on the Semantic Web. In *Proc. of SDM'04*, LNCS 3178, pages 118–132. Springer, 2004.

[19] R. Rosati. DL+log: Tight Integration of Description Logics and Disjunctive Datalog. In *Proc. of KR'06*, pages 68–78. AAAI Press, 2006.

[20] K. Samuel, L. Obrst, S. Stoutenberg, K. Fox, P. Franklin, A. Johnson, K. Laskey, D. Nichols, S. Lopez, and J. Peterson. Translating OWL and Semantic Web rules into Prolog: Moving toward description logic programs. *Theory Pract. Log. Program.*, 8(3):301–322, 2008.

[21] A. Toninelli, J. Bradshaw, L. Kagal, and R. Montanari. Rule-based and Ontology-based Policies: Toward a Hybrid Approach to Control Agents in Pervasive Environments. In *Proc. of SWPW'05*, 2005.

[22] 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):32–41, 2004.

[23] M. Winslett, C. C. Zhang, and P. A. Bonatti. PeerAccess: a logic for distributed authorization. In *Proc. of CCS'05*, pages 168–179. ACM, 2005.