Integration of Heterogeneous Policies for Trust Management

Samane Abdi
Insight Centre for Data Analytics
Department of Computer Science
University College Cork, Ireland
samane.abdi@insight-centre.org

Abstract—In this paper, we demonstrate an ontology-based approach that allows integration of heterogeneous security policies for subterfuge safe trust management. This approach, SSAL\textsuperscript{O}, represents the Subterfuge Safe Authorization Language (SSAL) using a description logic subset of the Web Ontology Language, and Semantic Web Rule Language. This implementation provides a policy engine that enforces subterfuge safe authorization of requests for accessing the protected resources of distributed principals. SSAL\textsuperscript{O} provides a common domain model for integration of heterogeneous security policies. This approach is useful for secure cooperation and interoperability among principals in open environments where each principal may have a different security policy and different implementation. We discuss the characteristics of SSAL\textsuperscript{O} in capturing SSAL and providing a framework for secure and dynamic integration of heterogeneous security policies specified by distributed principals in different domains. We employ various tools such as Prot\'eg\'e and Pellet to implement our model.

Keywords—Integration, Policy, Permission, Trust management, Open environments.

I. INTRODUCTION

In traditional access control mechanisms an access decision on whether to allow or to deny access to a resource is based on authentication and authorization processes. Such a mechanism is suitable for closed systems, in which the identities of requesters are known to the administrator in advance. However, when the environment becomes decentralized and open, where the identity of the requester is not known in advance, another approach for access control is required. Trust management was proposed as a solution for addressing access control in open systems [1]. Access decisions are accomplished by determining whether principals (authenticated entities) are authorized to use a resource. Access decisions are made based on whether a set of certificates (provided by the requester) comply with the policy rules. Thus, a technique is required to express certificates and policy statements, i.e. a policy language. The Subterfuge Safe Authorization Language (SSAL) was introduced as a policy language with the purpose of subterfuge safe delegation of permissions among principals [2]. Subterfuge refers to a deceptive behaviour with the goal of evading the actual intention of a security mechanism. A malicious principal uses a vulnerability in the permission naming scheme of current trust management frameworks. When two or more principals define the same permission specification for accessing their resources, they enable a vulnerability for subterfuge. The identical permission specifications might be ambiguous and cause confusion for the principal who receives similar permissions from different resource owners. This ambiguity and associated confusion can result in delegation subterfuge, whereby the principal who receives the permission for one domain can misuse it to access the resources of another domain by providing an apparently legitimate set of certificates. Specifying two similar permission names occurs because none of the principals in distributed and open environments have a complete picture of the naming scheme that other principals are using to specify permissions. SSAL provides a systematic permission naming scheme, rather than using the current ad-hoc permission naming schemes, to ensure global unique interpretation for permissions, and in this way prevents subterfuge [3], [4]. In addition to the subterfuge safe policy language for trust management among distributed principals from different domains, it is also important to capture a common vocabulary that allows information sharing and reuse to facilitate integration of security policies defined by individual principles for access to their own resources. Using ontologies facilitates sharing and reuse of knowledge and interoperability in the domain of interest. For a policy language, the ontology technique provides the ability to express the policy statements and certificates in a conceptual way. In addition, the ontology provides a common vocabulary for integration of heterogeneous policies defined by different principles in distributed environments. In this paper, an ontology-based approach models the SSAL policy language using the Web Ontology Language (OWL) [5] and the Prot\'eg\'e [6] knowledge-modelling tool. Description Logic (DL) [7] is a decidable fragment of First Order Logic (FOL) [8] and constitutes the formal basis for OWL-DL, a very expressive and yet decidable subspecies of OWL. The Semantic Web Rule Language (SWRL) [9] is a rule language that can extend the semantic information described in an OWL (consequently in an OWL-DL) ontology. For instance, some policy rules may not be expressed in OWL-DL, therefore we use SWRL. In this work, we model SSAL in OWL-DL and SWRL, and we use a DL-reasoner to reason over the knowledge asserted in the ontology. This paper has two contributions: one is modelling a policy engine using an ontology technique (SSAL\textsuperscript{O}) to address subterfuge safe trust management, and the other is a solution for secure, dynamic, and automatic integration of the heterogeneous policies (which may be implemented with different techniques such as XML ) for secure cooperation, and resource sharing among principals in open environments.

This paper is organized as follows: first we give a background on related work and preliminaries. Then we demonstrate the ontology model, SSAL\textsuperscript{O}. We later show how SSAL\textsuperscript{O} is
used for integration of heterogeneous policies that may be implemented in different languages with different techniques. An example on using SSAL\textsuperscript{O} as a policy engine is given. The work is then discussed in terms of its characteristics for automatic integration of locally defined policies to capture subterfuge safe trust management for cooperation of principals in open environments. Finally, some conclusions are drawn.

II. RELATED WORK

Grid technology was the very first effort to resolve heterogeneity problem in integration of data types such as security policies with different implementations [10]. The Grid infrastructure was a good start for principals to share their data, integrate them and consequently cooperate with each others. The main focus of that effort was resource sharing among virtual organization for seamless interactivity. However, some of heterogeneity problems still remained unresolved. Another approach for solving heterogeneity problem for integration of security policies was using matching scheme. Matching schemes are usually based on XML technology, that the match operator takes two graph-like structures as input and produces matching between their elements. If two structures correspond semantically to each other, matching is successful. The work presented in [11] which performs a match between the permission attributes, is a good example of this approach. Within the last few years, there has been an increase of interest in using ontologies for representing access control policies. Ontologies are believed to be the next trend for solving heterogeneity problem in integration of security policies. Ontologies provide formal specification and common vocabulary for a domain of interest. However, ontologies alone cannot resolve heterogeneity problems for all domains because there is not any single ontology that can address specifications for all domains of interests. Much existing work assumes semantic mapping between ontologies is performed manually [12], [13]. An example of an ontology approach for integration of policies is AIR (Accountability In RDF) proposed by Kagel et al. [14]. They used Accountability In RDF (AIR) as an ontology-based language to express the access control policy for sharing data. AIR enables users to share their data in open environments such as web services. However, the integration of multiple policies is based on matching the condition of a set of rules. The rules also have to be designed in the same language to be able to perform matching process on them. Integration and matching are technically difficult specially if multiple policies are defined in different languages with different implementation techniques. The identification of semantic relationships between these policies is also a difficult problem. Thus, an automated dynamic integration solution is required. In SSAL\textsuperscript{O}, we address not only the automatic integration of heterogeneous policies implemented with different techniques, but also we address the subterfuge safe collaboration of distributed principals after integration of their policies. This feature is unique to SSAL\textsuperscript{O}. To our best knowledge, no formal framework for integration of heterogeneous policies that can address subterfuge safe cooperation among distributed principals exists.

III. AN OVERVIEW ON SUBTERFUGE SAFE AUTHORIZATION LANGUAGE

The Subterfuge Safe Authorization Language (SSAL) is designed with the purpose of preventing the subterfuge problem in delegation of permissions in current trust management frameworks. SSAL prevents subterfuge by providing a global unique interpretation for permissions in a systematic way. Binding permission names defined locally in a name space (local permissions) to a global context provides a global unique interpretation for permissions [2]–[4]. In SSAL the notions of local name (inherited from Simple Distributed Security Infrastructure (SDSI) approach [15]) and local permission represent a non-ambiguous global and unique interpretation for principals and permissions, respectively. Principal names and permission names remain local to their originator’s name space and also globally they have a unique interpretation without requiring pre-agreed global naming services.

Delegation refers to the act of a principal to propagate the permission that it obtains from other principals further. In delegation, the principal who propagate permissions to others is called delegator and the principal who receives permissions is called delegatee. In conventional trust management systems a principal that is being delegated a permission is authorized for that permission, which by the delegation statement $P \rightarrow X \rightarrow Q$, $Q$ is considered to be authorized for $X$. However, in SSAL the principal who issues a delegation statement must hold the permission in the first place to be able to propagate the permission further. This prevents a malicious principal to delegate permissions that legitimately it is not expected to obtain and consequently delegate further to other principals. In this way, in delegation of a permission which the delegator cannot assert about holding that permission, the recipient also cannot deduce the holding of that permission and therefore is not authorized to perform actions based on that permission.

IV. ONTOLOGY-BASED APPROACH

SSAL\textsuperscript{O} includes five concepts as: Principal, Key, LocalName, LocalPermission and Delegation and sixteen object properties: asAuthAs, delegatesPermission, hasDelegator, hasDelegatee, hasNameSpace, hasOriginator, holds, isHeld, speaksFor, isSpokenBy, ; (the property isHeldBy is the inverse of property holds; and property speaksFor is the inverse of property isSpokenBy). In addition SSAL\textsuperscript{O} includes a data type (string) property: hasName. In the following we define the concepts using property definitions. The constructor (\forall) is a value restriction which all values of a property for instances of a concept must belong to the specified concept or data type. The constructor (\exists) is an existential quantifier that restricts those individuals which have a relationship to instances of a concept. The constructor (\equiv) is interpreted as a union of sets of individuals and the constructor (\cap) is interpreted as the intersection of sets of individuals. The inclusion (\subseteq) states a necessary but not sufficient condition for being an instance of a concept and the equality (\equiv) states necessary and sufficient conditions that an individual must hold in order for it to be included in a concept. The cardinality restriction (\equiv, \geq) specify the intended number of relationships that an individual must have for a given property.

**Concept Principal:** A principal is identified by either its public key or local name. The concept Principal subsumes two concepts as LocalName and Key where an instance $p$ is a member of concept Principal if and only if it is an instance of either LocalName or Key. This concept is expressed with the following restrictions:
**Principal** ≡ \((\text{LocalName} \cap \text{Key})\)

From this definition we do not know whether an individual which is a local name is also a public key or not. We can specify that any individual that is a local name is not a public key and vice-versa. That is the set of public keys and the set of local names are disjoint in the following way:

\((\text{LocalName} \cap \text{Key}) \subseteq \perp\)

**Concept LocalName**: A local name is an arbitrary name \(N\) that principal \(P\) (identified by its public key) chooses for principal \(Q\) in its name space. Principal \(P\) refers to the principal \(Q\) in its name space as \(N\) and uses the speaks for relation to link the local name to principal \(Q\). It means that any statement that is signed by \(Q\) can be viewed that it is originated from \(N\) in the name space of \(P\). The concept **LocalName**, subsumed by concept **Principal** and disjoint with concept **Key**, is the conceptualization of the local name entity in SSAL\(^O\). This concept is expressed with the following restrictions:

\[
\text{LocalName} \subseteq \text{Principal} \cap \\
\exists \text{isSpokenBy.} \text{Principal} \cap \\
\exists_{=1} \text{hasNameSpace.} \text{Principal} \cap \\
\exists_{=1} \text{hasName.string}
\]

An individual belongs to the concept **LocalName** if it has the object property relation **isSpokenBy** (corresponds to the inverse of speaks for relation in SSAL) to at least one instance of the concept **Principal**. Similarly, the individual must have the object property relation **hasNameSpace** to exactly one instance of concept **Principal** (either an instance of concept **Key** or **LocalName**), and have data type property relation of **hasName** to string type.

**Concept LocalPermission**: Each principal chooses a permission specification arbitrarily for its own resource and binds that specification to its name space (identified by a public key or local name). The binding permission to its name space is captured with the **hasNameSpace** property in SSAL\(^O\). In addition, each permission must be originated by a principal and this is captured in SSAL\(^O\) with the **isHeldBy** property. There is always an ordering relationship (either implicitly or explicitly) among permissions in local policies defined by principals. The local permissions ordering relationship is represented using the asAuthAs relation in the ontology whereby the statement asAuthAs\((Y,X)\) indicates that permission \(Y\) is no less authoritative than permission \(X\). The following necessary and sufficient condition restricts an individual to be included as an instance of this concept:

\[
\text{LocalPermission} \equiv \\
\exists_{=1} \text{hasNameSpace.} \text{Principal} \cap \\
\exists \text{isHeldBy.} \text{Principal} \cap \\
\forall \text{asAuthAs.} \text{LocalPermission}\]

This states that every instance of the **LocalPermission** concept has exactly one **hasNameSpace** relation to the instances of concept **Principal**. An instance of a **LocalPermission** concept has the binary relation to at least one instance of the **Principal** concept **isHeldBy** property. An instance of **LocalPermission** has the asAuthAs relation to instances of only the same concept (LocalPermission).

**Concept Delegation**: A delegation statement indicates that the authority for a permission is delegated from one principal to the other principals. An instance of **Delegation** concept is defined with the following restrictions:

\[
\text{Delegation} \equiv \exists_{=1} \text{hasDelegator.} \text{Principal} \cap \\
\exists \text{hasDelegatee.} \text{Principal} \cap \\
\exists \text{delegatesPermission.} \text{LocalPermission}
\]

This restriction states that an instance of a **Delegation** concept has a binary relation of **hasDelegator** to exactly one instance of **Principal** concept. This means that the delegation statement is signed by only one principal called delegator. However, a delegation statement may state the delegation of multiple permissions to multiple principals as delegates. Each instance of the **Delegation** concept has a **hasDelegatee** relation to at least one instance of concept **Principal**. Moreover, an instance of concept **Delegation** has the relation of **delegatesPermission** to at least one instance of concept **LocalPermission**.

### A. SSAL Policy Rules

We use SWRL to represent policy rules introduced in [3], [4]. The policy statements are represented by individuals and their relations (via properties) with other individuals in the ontology. In addition to the asserted knowledge, there is also hidden knowledge in the ontology that can be inferred from the asserted knowledge. To infer that knowledge, a reasoning tool is required to accomplish the process of inference. Along with SWRL there is a reasoning engine (Jess) that provides reasoning over the asserted knowledge and produces new knowledge regarding the asserted knowledge.

**a) Name Rule**: The principal \(Q\) identifies a name \(N\) in its name space, and \(Q\) speaks for \(R\); then it is inferred that \(R\) is the implicit name space of \(N\). A reduction rule is derived to reduce local names to principals; which considering \(?p, ?q, \) and \(?r\) as principals, if principal \(?q\) speaks for principal \(?r\), and \(?q\) chooses name \(?n\) for principal \(?p\), it can be inferred \(?p\) is identified as \(?n\) in the name space of \(?r\).

\[
N1 : \text{Principal}(?p) \land \text{Principal}(?q) \land \text{Principal}(?r) \land \\
\text{speaksFor}(?p, ?qn) \land \text{LocalName}(?qn) \land \text{hasNameSpace}(?qn, ?q) \land \\
\text{hasName}(?qn, ?n) \land \text{speaksFor}(?q, ?r) \land \\
\text{swrlx : makeOWLIndividual}(?rn, ?qn) \\
\rightarrow \text{LocalName}(?rn) \land \text{hasNameSpace}(?rn, ?r) \land \\
\text{speaksFor}(?p, ?rn) \land \text{hasName}(?rn, ?n)
\]
b) Permission Delegation Rules: Delegation refers to the act of a principal to propagate further the permission that it obtains from other principals. If a principal ?r speaks for principal ?p, any permission ?x that is delegated to ?q is implicitly delegated to ?r (P1). Moreover, if a principal ?p delegates permission ?x, it is implicitly delegates any permission ?y that is dominated by ?x (P2). This is modelled in the following SWRL rules:

\[ P1: \text{Principal}(?p) \land \text{Principal}(?q) \land \text{LocalPermission}(?x) \land \\
\quad \text{delegatesPermission}(?d, ?perm) \land \text{hasDelegatee}(?d, ?q) \land \\
\quad \text{hasDelegator}(?d, ?p) \land \text{speaksFor}(?r, ?q) \\
\quad \longrightarrow \text{hasDelegatee}(?d, ?r) \]

\[ P2: \text{asAuthAs}(?x, ?y) \land \text{hasDelegator}(?d, ?p) \land \\
\quad \text{hasDelegatee}(?d, ?q) \land \text{delegatesPermission}(?d, ?x) \\
\quad \longrightarrow \text{delegatesPermission}(?d, ?y) \]

Executing the Jess engine would have the effect of setting the delegatesPermission property as a relation from the individual ?d to ?y that satisfies the rule.

c) Permission Holding Rule: Delegation of a permission does not necessarily imply that the recipient of the permission holds it. Holding a permission depends on either the delegator already originated the permission in its name space or already held the permission to propagate it further. This prevents malicious principals delegating permissions that they do not hold, and as a consequence are not expected to delegate. A principal by originating a permission asserts that it holds that permission. Holding a permission means that a principal is authorized to perform actions based on that permission. Following implements this with SWRL rule:

\[ P3: \text{Principal}(?p) \land \text{Principal}(?q) \land \text{LocalPermission}(?x) \\
\quad \land \text{holds}(?p, ?x) \land \text{delegatesPermission}(?d, ?x) \\
\quad \land \text{hasDelegatee}(?d, ?q) \land \text{hasDelegator}(?d, ?p) \\
\quad \longrightarrow \text{holds}(?q, ?x) \]

d) Permission Global Ordering Rule: There are reduction rules to reduce local permissions defined in the name space of a principal, and that principal is identified by a local name. Considering ?p, ?q, and ?r as principals, and ?pm as local permission, the rule P4 indicates reduction of local permissions. It indicates that, if the permission ?pm is issued by principal ?p and principal ?q speaks for ?p then the permission ?pm is implicitly in the name space of ?q. Note that, the asAuthAs property is transitive relation.

\[ P4: \text{Principal}(?p) \land \text{Principal}(?q) \land \text{LocalPermission}(?x) \\
\quad \land \text{LocalPermission}(?pm) \land \text{hasNameSpace}(?pm, ?p) \land \\
\quad \text{speaksFor}(?q, ?pm) \land \text{asAuthAs}(?x, ?pm) \\
\quad \longrightarrow \text{hasNameSpace}(?pm, ?q) \]

B. Integration of Policies within SSAL

Different principals define specific kinds of security policies to meet their specific needs. For example, confidentiality hierarchy is a kind of security policy in military environments, or access permission hierarchy is another kind of security policy in file systems. These security policies are defined by different principals in different name spaces and they may have their local techniques for implementation. To effectively manage security policies we must be able to produce compatible policy representations. The existence of a large number of representation methods leads to the conclusion that security policies, even when semantically compliant, can be represented in ways that differ substantially in terms of formalism, structure, and hierarchy, thus raising obstacles to their reconciliation. Therefore, in order to effective management of trust and authorization relationship among distributed principals one has to be able to integrate all policy representations to make proper access decision. Different principals define different security policies in their name spaces for accessing to their resources, so called a local policy. Local policies may have different implementation in each name space. For example, one principal may implement its security policy using XML technique and the other principal use ontology technique to implement its local security policy. In order to integrate with local policies of other principals for federation, SSAL provides a common vocabulary that is understandable by different parties. We assume that each local policy contains a set of permissions that constrain access to the corresponding resources. The set of all permissions SP in a principal’s name space may be considered to form a pre-order relation as \( (SP, \sqsubseteq) : (sp_i \sqsubseteq sp_j) \sqsubseteq (sp_i, sp_j \in SP) \). In other words, sp_j it also holds the permission sp_i (inferred by reasoning over the rules P2 and P3). In addition to providing a common vocabulary, SSAL enables the subterfuge safe integration of policies described locally by each resource owner. Subterfuge is a result of ambiguity in permission specification when two different principals define the same permission specification for their resources; this can be avoided by using the local permission mechanism. A permission x for a given resource of principal P will be represented as the following individual and its relations in SSAL:

\[ \text{LocalPermission}(x) \leftarrow \lnot \text{isHeldBy}(x, P) \land \text{hasNameSpace}(x, P) \]

In each local policy, the set of permissions and their ordering relationship is specified locally, therefore the principal who defines the policy must explicitly define a global interpretation for the set of permissions and their ordering relationship. By signing the set of permissions and their ordering a principal provides a global unique interpretation for permissions and
consequently prevents subterfuge during open cooperation with other principals. This is modelled in the SSAL\textsuperscript{O} through OWL individuals and properties; where, the set of permissions are considered as instances of concept \textit{LocalPermission}, the name space of these permissions are instances of concept \textit{Principal} that signs the whole set of permissions, and the pre-order relations among the permissions is considered as the as\textit{AuthAs} property. For example, the permission \textit{read} of a set of permissions including \{\textit{read, write}\}, and their ordering relationship as \textit{read} \sqsubseteq \textit{write} signed by \( k_A \) \((\{\textit{read, write}, \sqsubseteq \}\_s_{k_A})\) will be captured in SSAL\textsuperscript{O} as the following individual and properties:

\[
\text{LocalPermission}(k_A \textit{read}) \leftarrow \text{isHeldBy}(k_A, \textit{read}, k_A) \cap \\
\text{hasNameSpace}(k_A \textit{read}, k_A) \cap \text{asAuthAs}(k_A \textit{write}, k_A \textit{read})
\]

Figure 1 depicts the locally defined policy by principal \( k_A \) as a fragment of SSAL\textsuperscript{O}. This way each principal can define its security policy locally, in any policy language. The signed set of permissions and their pre-order relations are captured as instances of concept \textit{LocalPermission} and as\textit{AuthAs} property (through an interpreter implemented in Java). These individuals will be part of the knowledge in SSAL\textsuperscript{O} and can be reasoned over for making the proper access decision.

\textbf{C. Trust Management}

In addition to reasoning over the knowledge in SSAL\textsuperscript{O}, a query engine is required for querying over the knowledge to answer an access request. Semantic Query-enhanced Web Rule Language (SQWRL) \cite{16} is a language to support querying of OWL ontologies. An access request can be answered by querying the knowledge in SSAL\textsuperscript{O}. For example to verify whether or not a principal \( p \) is authorized to sell flight number 123 for airline \( A \), it is necessary to verify if the relation \( \text{holds}(p, (k_A \textit{sell flight No.123})) \) exists or can be inferred in the ontology. This requires that before any possible request evaluation all the policy rules (implemented in SWRL) have to be executed to infer all \( \text{holds} \) relation for the principal \( p \) and permission \( (k_A \textit{sell flight No.123}) \). Followings are the queries to evaluate if a request complies with the represented policy in the knowledge base.

\textbf{Query for Authorization:} The following query determines an authorization where checks whether the requester holds the permission to accomplish what it requests.

\[
Q:\text{Principal(?p)} \land \text{LocalPermission(?perm)} \land \\
\text{hasNameSpace(?perm, k_A)} \land \text{asAuthAs(?perm)} \land \text{query engine is required for querying over the knowledge to answer an access request.}
\]

\[
\quad\text{select(?p, ?perm)}
\]

This query returns all tuples of a principal and the permissions that it holds. A DL reasoner, Pellet, is integrated in the Prot\é\`eg\é and performs the reasoning task over knowledge in SSAL\textsuperscript{O}. Thus, we use SSAL\textsuperscript{O} as a policy engine to make a decision for an access request. A policy engine takes a request, a set of certificates, and policy as inputs, then outputs a decision for that request \cite{17}. In our model, a requester makes a request to access to some protected resources through an Application Programming Interface (API) in the resource owner’s trusted environment. The API queries the trust management engine (SSAL\textsuperscript{O}) via a query interpreter. The query interpreter which is implemented in the Java programming language, queries the ontology about the request. Note that, the requester may present a set of certificates (encoded within Security Asserted Markup Language (SAML) \cite{18}) to the resource owner as proof of its authorization. The certificates are in XML data format and are added to the ontology as OWL individuals and their relations via OWL properties. Figure 2 depicts the application of SSAL\textsuperscript{O} as policy engine.

\textbf{D. Case study: Trust Management for a Selling Service by Brokers}

Consider that the owner of airline \( A \) trusts broker \( B \) to act as its broker. Brokers are authorized to sell flights. Airline \( A \) (the owner of public key \( k_A \)) originates (and therefore holds) a local permission \( (k_A \textit{sell flight No.123}) \). Then the airline \( A \) adds broker \( B \) to the group of brokers identified by the local name \( (k_A \textit{flight broker}) \) (the broker \( B \) is the owner of public key \( k_B \)). The local name, local permission, and delegation statements that airline \( A \) issues are captured in the following individuals and their relationships in SSAL\textsuperscript{O}.
LocalName(kA flight broker) ←
  isSpokenBy(kA flight broker, kB)\(\sqcap\)
  hasNameSpace(kA flight broker, kA)\(\sqcap\)
  hasName(kA flight broker, flight broker)
LocalPermission(sell flight–all) ←
  hasNameSpace(sell flight–all, kA)\(\sqcap\)
  isHeldBy(sell flight–all, kB)\(\sqcap\)
Delegation(delSell) ←
  hasDelegator(delSell, kA)\(\sqcap\)
  delegatesPermission(delSell, sell flight–all)\(\sqcap\)
  hasDelegatee(delSell, kA flight broker)

The set of policy rules that is encoded in SWRL are used to reason over the asserted knowledge and infer new knowledge for making proper access decision. Therefore, in receiving the broker B’s request for selling flights, the airline A wishes to check whether kB’s request for selling flight at the airline A is authorized or not. The policy engine executes the SWRL rule P1 to reason over the asserted knowledge within SSAL\(^O\). Thus, the SQWRL query: Q checks if the following statements can be inferred:

\[
\text{isHeldBy(sell flight–all, kB)}
\]

As a result of a successful query, broker B’s access for selling flights of airline A is granted.

V. Run Time Test

In this section, we present basis evaluation of SSAL\(^O\) as a policy engine. The objective of these experiments is to evaluate the ability of the SSAL\(^O\) to integrate more security policies defined by different principals in a certain time intervals. We used our implementation of SSAL\(^O\) (description logic based ontology) and the description logic reasoner Pellet [19] to carry out experiments. We used Jena Semantic Web Toolkit [20], which supports rule-based inference over OWL-DL knowledge base. The experiments have been conducted on a Linux workstation with the following hardware configurations: 8Gb RAM with AMD A10-4655M quad core processor. Figure 3 shows the results of the experiments. The run time performance of SSAL\(^O\) depends on two factors: size of asserted individuals in the ontology, number of heterogeneous policies for integration. This experiment shows that although the run-time increases when the number of asserted individuals (certificates and policies) increase, but the performance is adequate for most intended scenarios such as web services, web-based applications apart from time-critical ones. It also shows that reasoning based on SSAL\(^O\) is a computational task.

Fig. 3. Run time performance of reasoning over SSAL\(^O\)

VI. Discussions

In the context of trust management, an ontology produces a shared understanding of different policies in different domains, represented as a set of concepts, relations, functions, axioms and individuals. There are several reasons for developing our trust management model based on ontology:

Knowledge Sharing: The use of SSAL\(^O\) enables different principals to have a common set of concepts about their security policies while interacting with one another.

Knowledge Reuse: SSAL\(^O\) as a policy engine can be reused by different principals without building a new policy engine from scratch. Reusing ontologies reduces engineering costs since it avoids rebuilding existing ontologies. Moreover, since SSAL\(^O\) is understood as a means for sharing knowledge concepts, reusing that increases the secure interoperability between different principals both on the syntactic and on the semantic level. Principals using the same ontology are assumed to hold the same view upon the modelled universe of discourse, and thus define and use domain concepts in the same way.

Scalability: The description of trust management system in a machine-understandable fashion (ontology) is expected to have a great impact in areas of policy integration, as it is expected to enable dynamic and scalable cooperation among different principals of open environments such as web services, organizations, coalitions.

Complexity: Choosing OWL-DL provides the possibility of using the OWL-DL reasoner as a policy engine and a query tool. The DL reasoner Pellet and the SWRL engine have high complexity (NExpTime-complete) but DL reasoners can handle all features of the OWL-DL language. The DL expressibility of SSAL\(^O\) model is SHOIN(D), where S stands for ALC [21] plus role transitivity, H stands for role hierarchy, O stands for nominals, I stands for inverse role, N stands for cardinality restrictions, and D stands for datatypes. However, we did not use some features such as nominals or role hierarchy in our model. We evaluated the complexity of our model with its specific features used in SSAL\(^O\) via a calculator for complexity of reasoning in description logics [22], and determined it to be NP-complete.

Subterfuge Safe Policy Integration: Ontologies are frameworks for organizing structured data. Defining permissions for hierarchical resources is a very common requirement for security policies. The permissions for hierarchical resources can be modelled in the ontology since ontologies are a set of hierarchical and relational representational primitives. In the
SSAL policy language we defined the No less Authoritative than relation as the global ordering relationship among permissions. This ordering relationship is assumed to be specified explicitly. However, resources in a system have a hierarchical structure and therefore permissions to access those resources form an implicit partial ordering relation. For example, the airline A may specify permission \( k_A \) (sell flight, all) for selling all flights in its domain and the permission \( k_A \) (sell flight, No.123) for selling flight number 123. Consider that there is an implicit ordering relationship among the permissions that A defines in its name space. For instance, there is an ordering relationship as (sell flight, No.123) \( \subseteq \) (sell flight, all) in which permission (sell flight, all) implies permission (sell flight, No.123). Any requester that holds the permission \( k_A \) (sell flight, all) implicitly should be able to sell flight number 123. The resource owner must specify the ordering relation among the permissions it defines in its local policies. This relation is modelled more simply in SSAL\(^O\) using the OWL property asAuthAs to address subterfuge safe cooperation among distributed principals when their local policies are integrated with one another.

VII. CONCLUSIONS

In this paper, we demonstrated an ontology-based trust management system, which uses SSAL\(^O\) as its policy engine. Using an ontology for representing and reasoning over the policies provides a common vocabulary and well understood approach for open environments, where multiple organizations with heterogeneous security policies wish to cooperate. The implementation also supports integration of heterogeneous policies (policies specified in different languages which may have different implementation in their issuer’s name space) to facilitate trust management in open environments. SSAL\(^O\) can be potentially used in open systems such as distributed systems, web services, coalitions, web-based applications, and cloud federations [23] for subterfuge safe, and dynamic cooperation. The complexity of reasoning over the knowledge in SSAL\(^O\) was evaluated as NP-complete which means that the runtime required to reason over the asserted knowledge increases quickly as the size of the asserted knowledge grows in SSAL\(^O\). Experiments have shown adequate performance for typical non-time critical situations.

ACKNOWLEDGMENT

This work was instigated by Dr. Simon N. Foley and supported by Science Foundation Ireland under grant 08/SRC/11403. The author would like to thank Dr. John Herbert for his support.

REFERENCES


