A REVIEW OF TRUST MANAGEMENT, SECURITY AND PRIVACY POLICY LANGUAGES

Juri Luca De Coi, Daniel Olmedilla
L3S Research Center, University of Hannover, Appelstr. 9a, D-30167 Hannover, Germany
decoi@L3S.de, olmedilla@L3S.de

Keywords: Policy languages, comparison, trust management, privacy, security, review.

Abstract: Policies are a well-known approach to protecting security and privacy of users as well as for flexible trust management in distributed environments. In the last years a number of policy languages were proposed to address different application scenarios. In order to help both developers and users in choosing the language best suiting her needs, policy language comparisons were proposed in the literature. Nevertheless available comparisons address only a small number of languages, are either out-of-date or too narrow in order to provide a broader picture of the research field. In this paper we consider twelve relevant policy languages and compare them on the strength of ten criteria which should be taken into account in designing every policy language. Some criteria are already known in the literature, others are introduced in our work for the first time.
By comparing the choices designers made in addressing such criteria, useful conclusions can be drawn about strong points and weaknesses of each policy language.

1 Introduction

Security management is a foremost issue in large scale networks like the World Wide Web. In such a scenario, traditional assumptions for enforcing access control regulations do not hold anymore. In particular identity-based access control mechanisms have proved to be ineffective, since in decentralized and multicentric environments, the requester and the service provider are often unknown to each other.

Policies are a well-known approach to protecting security and privacy of users in the context of the Semantic Web: policies specify who is allowed to perform which action on which object depending on properties of the requester and of the object as well as parameters of the action and environmental factors (e.g., time).

The potential policies have proved to own is not fully exploited yet, since nowadays their usage is mainly restricted to specific application areas. In the last years many policy languages were proposed, targeting different application scenarios and provided with different features and expressiveness: scope and properties of available languages have to be known to the user in order to help her in choosing the one most suitable to her needs.

In an attempt to help with these and other problems, comparisons among policy languages have been provided in the literature, anyway existing comparisons either do not consider a relevant number of available solutions or are mainly focused on the application scenarios the authors worked with, moreover policy-based security management is a rapidly evolving field and most of this comparison work is now out-of-date.

Currently a broad and up-to-date overview covering most of the relevant available policy languages is lacking; in this paper we intend to fill this gap by providing an extensive comparison covering twelve policy languages. Such a comparison will be carried out on the strength of eleven criteria, partly already known in the literature and partly introduced in our work for the first time.

This paper is organized as follows: in section 2 related work is accounted for. Sections 3 and 4 respectively introduce the languages which will be compared in the rest of the paper and the criteria according to which the comparison will be carried out. The actual comparison takes place in section 5, whereas sections 6 and 7 respectively present overall results and concludes the paper.

2 Related work

The paper of Seamons et al. (Seamons et al., 2002) is the basis of our work: some of the insights they suggested have proved to be still valuable right now and as such they are addressed in our work as well. Nevertheless in over three years the research field has considerably changed and nowadays many aspects of (Seamons et al., 2002) are out of date.

The pioneer paper of Seamons et al. paved the way to future research on policy language comparisons like Tonti et al. (Tonti et al., 2003), Anderson (Anderson, 2006) and Duma et al. (Duma et al., 2007): although (Tonti et al., 2003) actually presents a comparison of two ontology-based languages (namely KAoS and Rei) with the object-oriented language Ponder, the work is rather an argument for ontology-based systems, since it clearly shows the advantages of ontologies.

Because of the impressive amount of details it provides, (Anderson, 2006) restricts the comparison to only two (privacy) policy languages, namely EPAL and XACML, therefore a comprehensive overview of the research field is not provided.

Finally (Duma et al., 2007) provides a comparison specifically targeted to giving insights and suggestions to policy writers (*designers*): therefore the criteria, according to which the comparison is carried out, are mainly practical ones and scenario-oriented, whereas more abstract issues are considered out of scope and hence not addressed.

3 Presentation of the considered policy languages

To date a bunch of policy languages have been developed and are currently available: we have chosen those which at present seem to be the most popular ones, namely Cassandra (Becker and Sewell, 2004), EPAL (Ashley et al., 2003), KAoS (Uszok et al., 2003), PeerTrust (Gavriloaie et al., 2004), Ponder (Damianou et al., 2001), Protune (Bonatti et al., 2006), PSPL (Bonatti and Samarati, 2000), Rei (Kagal et al., 2003), *RT* (Li and Mitchell, 2003), TPL (Herzberg et al., 2000), WSPL (Anderson, 2004) and XACML (Lorch et al., 2003).

The number and variety of policy languages proposed so far is justified by the different requirements they had to accomplish Ponder was meant to help local security policy specification, therefore typical addressed application scenarios include registration of users or logging and audit events. WSPL's name itself (namely Web Services Policy Language) suggests its goal: supporting description and control of various aspects and features of a web service. Web services are addressed by KAoS too, as well as general-purpose grid computing, although it was originally oriented to software agent applications. Rei's design was primarily concerned with support to pervasive computing applications (i.e. those in which people and devices are mobile and use wireless networking technologies to discover and access services and devices). EPAL (Enterprise Privacy Authorization Language) was proposed by IBM in order to support enterprise privacy policies. Some years before IBM had already introduced the pioneer role-based policy language TPL (Trust Policy Language), which paved the way to other role-based policy languages like Cassandra and *RT* (Role-based Trust-management framework), both of which aimed to address access control and authorization problems which arise in large-scale decentralized systems The main goal of PSPL (Portfolio and Service Protection Language) was providing a uniform formal framework for regulating service access and information disclosure in an open, distributed network system like the web. PeerTrust is a simple yet powerful language for trust negotiation on the Semantic Web based on a distributed query evaluation. Trust negotiation is addressed by Protune too, which supports a broad notion of "policy" and does not require shared knowledge besides evidences and a common vocabulary. Finally XACML (eXtensible Access Control Markup Language) was meant to be a standard general purpose access control policy language, ideally suitable to the needs of most authorization systems.

Given the multiplicity of available languages and the sometimes very specific contexts they fit into, one may argue that a meaningful comparison among them is impossible or, at least, meaningless. We claim that such a comparison is not only possible but even worth: to this aim we identified eleven criteria which should be taken into account in designing every policy language. By comparing the choices designers made in addressing such criteria, useful conclusions can be drawn about strong points and weaknesses of each policy language. More important yet, by outlining advantages and drawbacks of each language, our analysis will hopefully help a user in choosing the one which mostly suits her needs.

4 Presentation of the considered criteria

We acknowledge the remark made by (Duma et al., 2007), according to which a comparison among policy languages on the basis of the criteria presented in (Seamons et al., 2002) is only partially satisfactory for a designer, since general features do not help in understanding which kind of policies can be practically expressed with the constructs available in a language. Therefore in our comparison we selected a good deal of criteria having a concrete relevance: on the other hand, since we did not want to come short on theoretical issues, we selected four additional criteria, basically taken from (Seamons et al., 2002) and somehow reworked and updated them. We called these more theoretical criteria *core policy properties* whereas more practical issues have been grouped under the common label *contextual properties*.

4.1 Core policy properties

Well-defined semantics According to (Seamons et al., 2002) we consider a policy language's semantics to be well-defined if the meaning of a policy written in that language is independent of the particular implementation of the language.

Monotonicity In the sense of logic a system is monotonic if the set of conclusions which can be drawn from the current knowledge base does not decrease by adding new information to the knowledge base. In the sense of (Seamons et al., 2002) a policy language is considered to be monotonic if an accomplished request would also be accomplished if accompanied by additional disclosure of information by the peers: in other words, disclosure of additional evidences and policies should only result in the granting of additional privileges. Policy languages may be not monotonic in the sense of logic but still be monotonic in the sense of (Seamons et al., 2002), like Protune.

Condition expressiveness A policy language must allow to specify under which conditions the request of the user should be accomplished. Policy languages differ in the expressiveness of such conditions: some languages allow to set constraints on properties of the requester, but not on parameters of the requested action, moreover constraints on environmental factors (e.g., time) are not always supported.

Underlying formalism A good deal of policy languages base on some well-known formalism: Protune and PSPL base on Logic programming, whereas Cassandra, Peer-Trust and *RT* on a subset of it (namely Constrained DATALOG) and KAoS on Description logics. Knowledge about the formalism a language bases upon can be useful in order to understand some basic features of the language itself.

4.2 Contextual properties

Action execution During the evaluation of a policy some actions may have to be performed: one may want to retrieve the current system time, to send a query to a database or to record some information in a log file. It is worth noticing that this criterion evaluates whether a language allows the *policy writer* to specify actions within a policy: during the evaluation of a policy the engine may carry out non-trivial actions on its own (e.g., both *RT* and TPL engines provide automatic resolution of credential chains) but such actions are not considered in our investigation

Delegation Delegation is often used in access control systems to cater for temporary transfer of access rights to agents acting on behalf of other ones (e.g., passing write rights to a printer spooler in order to print a file). The right of delegating is a right as well and as such can be delegated, too. Some languages provide a means for cascaded delegations up to a certain length, whereas others allow unbounded delegation chains. In order to support delegation many languages provide a specific built-in construct, whereas others (e.g., Cassandra and Protune) exploit more fine-grained features of the language in order to simulate high-level constructs.

Type of evaluation In languages supporting negotiations, policy evaluation is distributed: at each step of the negotiation a peer sends the other one information which (possibly) lets the negotiation advance. Each peer can (unsuccessfully) terminate the negotiation, moreover the peer who received the original request may decide to accomplish it, thereby successfully terminating the negotiation. Evaluation is supposed to be performed locally by languages which do not support negotiations, although some of them may allow to split a policy on several nodes and provide some means for collecting all fragments before (or during) the evaluation. Finally some languages like Ponder neither support distributed evaluation nor distributed policies

Evidences During the evaluation of authentication policies, it may be needed for the requester to provide some signed statements (*credentials*) issued by a trusted entity and asserting some properties about the requester itself. PeerTrust, Protune and PSPL provide another kind of evidence, namely *declarations* which are non-signed statements about properties of the holder (e.g., credit-card numbers)

Negotiation support (Anderson, 2004) adopts a broad notion of "negotiation", namely a negotiation is supposed to happen between two peers whenever (i) both peers are allowed to define a policy and (ii) both policies are taken into account when processing a request. In this paper we adopt a narrower definition of negotiation by adding a third prerequisite stating that (iii) the evaluation of the request must be distributed, i.e., both peers must locally evaluate the request and either decide to terminate the negotiation or send a partial result to the other peer who will go on with the evaluation. Whether the evaluation is local or distributed may be considered an implementation issue, as long as policies are freely disclosable. Distributed evaluation is required under a conceptual point of view as soon as the need for keeping policies private arises: indeed if policies were not private, simply merging the peers' policies would reveal possible compatibilities between them

Policy engine decision The result of the evaluation of a policy must be notified to the requester. The result sent back by the policy engine may carry information to different extents: in the easiest case a boolean answer may be sent (allowed vs. denied). Some languages support error messages, whereas Protune is the only language providing enough informative content to let the user understand how the result was computed (and thereby why the query succeeded/failed)

Extensibility Since experience shows that each system needs to be updated and extended with new features, a good programming practice requires to keep things as general as possible in order to support future extensions. Almost every language provides some support to extensibility. RT may be regarded as an exception since, as pointed out by (Becker and Sewell, 2004), the need for more advanced features was handled by releasing a new flavor of the language (available RT flavors can be obtained by combining RT_0 and RT_1 on the one hand with RT^T and/or RT^D on the other one). In the following we will provide a description of the mechanisms languages adopt in order to support extensibility

5 Comparison

In this section the considered policy languages will be compared according to the criteria outlined in section 4. The overall results of the comparison are summarized in Table 5 [1].

Well-defined semantics A policy language's semantics is well-defined if the meaning of a policy written in that language is independent of the particular implementation of the language. We assume policy languages based on Logic programming or Description logics to have well-defined semantics: the formalisms underlying the considered policy languages will be accounted for in the following. So far we restrict ourselves to list the languages provided with a well-defined semantics, namely, Cassandra, EPAL, KAoS, PeerTrust, Protune, PSPL, Rei and RT

Monotonicity In the sense of (Seamons et al., 2002) a policy language is considered to be monotonic if disclosure of additional evidences and policies only results in granting additional privileges, therefore the concept of "monotonicity" does not apply to languages which do not provide support for credentials, namely EPAL, Ponder, WSPL and XACML. All other languages are monotonic, with the exception of TPL, which explicitly chose to support *negative certificates*, stating that a user can be assigned a role R if there exists no credential of some type claiming something about it. The authors of TPL acknowledge that it is almost impossible proving that there does not exist such a credential somewhere, therefore they interpret their statement in a restrictive way, i.e., they assume that such a credential does not exist if it is not present in the local repository. Despite this restrictive definition the language is not monotonic since, as soon as such a credential is released and stored in the repository, consequences which could be previously drawn cannot be drawn anymore

Condition expressiveness A role-based policy language maps requesters to roles, the assigned role is afterwards exploited in order (not) to authorize the requester to execute some actions. The mapping to a role may in principle be performed according to the identity or other properties of the requester (to be stated by some evidence) and eventually environmental factors (e.g., current time). Cassandra (equipped with a suitable constraint domain) supports both scenarios. Environmental factors are not taken into account by TPL, where the mapping to a role is just performed according to the properties of the requester; such properties can be combined by using boolean operators, moreover a set of built-in operators (e.g., greater than, equal to) is provided in order to set constraints on their values. Environmental factors are not taken into account by RT_0 either, where role membership is identity-based, meaning that a role must explicitly list its members; nevertheless since (i) roles are allowed to express set of entities having a certain property and (ii) conjunctions and disjunctions can be applied to existing roles in order to create new ones, then role membership is finally based on properties of the requester. RT_1 goes a step beyond and, by adding the notion of *parametrized role*, allows to set constraints not only on properties of the requester but even on the ones of the object, the requested action should be performed upon. A non role-based policy language does not split the authentication process in two different steps but directly provides an answer to the problem whether the requester should be allowed to execute some action. In this case the authorization decision can be made in principle not only depending on properties of the requester or the environment, but also according to the ones of the object the action would be performed upon as well as parameters of the action itself. EPAL introduces the further notion of

[1] Table 5 does not contain criterion "condition expressiveness" since such criterion is not well-suited to be represented in a table.

"purpose" for which a request was sent and allows to set conditions on it. Some non role-based languages make a distinction between conditions which must be fulfilled in order for the request to be taken into consideration (which we call *prerequisites*, according to the terminology introduced by (Bonatti and Samarati, 2000)) and conditions which must be fulfilled in order for the request to be satisfied (*requisites*); not always both kinds of conditions have the same expressiveness. Let start checking whether and to which extent the non role-based policy languages we consider support prerequisites: WSPL and XACML allow only to use a simple set of criteria to determine a policy's applicability to a request, whereas Ponder provides a complete solution which allows to set prerequisites involving properties of requester, object, environment and parameters of the action. Prerequisites can be set in EPAL and PSPL as well. With the exception of Ponder, which allows restrictions on the environment just for delegation policies, each other language supports requisites (Rei is even redundant in this respect): KAoS allows to set constraints on properties of the requester and the environment, Rei also on action parameters and Protune, PSPL, WSPL and XACML also on properties of the object. EPAL supports conditions on the purpose for which a request was sent but not on environmental factors. Attributes must be typed in EPAL, WSPL, XACML and typing can be considered a constraint on the values the attribute can assume, anyway the definition of the semantics of such attributes is outside WSPL's scope. Finally, in PeerTrust conditions can be expressed by setting guards on policies: each policy consists of a guard and a body, the body is not evaluated until the guard is satisfied

Underlying formalism The most part of languages provided with a well-defined semantics rely on some kind of Logic programming or Description logics. Logic programming is the semantic foundation of Protune and PSPL, whereas a subset of it, namely Constraint DATALOG, is the basis for Cassandra, PeerTrust and *RT*. KAoS relies on Description logics, whereas Rei combines features of Description logics (ontologies are used in order to define domain classes and properties associated with the classes), Logic programming (Rei policies are actually particular Logic programs) and Deontic logic (in order to express concepts like rights, prohibitions, obligations and dispensations). EPAL exploits Predicate logic without quantifiers. Finally, no formalisms underly Ponder (which only bases on the Object-oriented paradigm), TPL, WSPL and XACML

Action execution Ponder allows to access system properties (e.g., time) from within a policy, moreover it supports obligation policies, asserting which actions should be executed if some event happens. XACML allows to specify actions within a policy; these actions are collected during the policy evaluation and executed before sending a response back to the requester. A similar mechanism is provided by EPAL and of course by WSPL, which is indeed a specific profile of XACML. The only actions which the policy writer may specify in PeerTrust and PSPL are related to the sending of evidences, whereas Protune supports whatever kind of actions, not necessarily side-effect free, as long as a basic assumption holds, namely that action results do not interfere with each other. Cassandra (equipped with a suitable constraint domain) allows to call side-effect free functions (e.g., to access the current time). Finally, KAoS, Rei, *RT* and TPL do not support execution of actions

Delegation Ponder defines a specific kind of policies in order to deal with delegation: the field `valid` allows *positive* delegation policies to specify constraints (e.g., time restrictions) to limit the validity of the delegated access rights. Rei allows not only to define policy delegating rights but even policy delegating the right to delegate (some other right). Delegation is supported by RT^D ("D" stands indeed for "delegation"). Ponder delegation chains have length 1, whereas in *RT* delegation chains always have unbounded length. Cassandra and Protune provide a more flexible mechanism which allows to explicitly set the desired length of a delegation chain (as well as other properties of the delegation). Delegation (of authority) can be expressed in PeerTrust by exploiting operator "@". Finally, EPAL, KAoS, PSPL, TPL, WSPL and XACML do not support delegation

Type of evaluation The most part of the considered languages require that all policies to be evaluated are collected in some place before starting the evaluation, which is hence performed locally: this is the way EPAL, KAoS, Ponder, *RT* and TPL work. Other languages, namely Cassandra, Rei, WSPL and XACML, perform policy evaluation locally, nevertheless they provide some facility in order to collect policies (or policy fragments) which are spread over the net. Policies can be collected into a single place if they are freely disclosable, therefore the languages mentioned so far do not address the possibility that policies themselves may have to be kept private. Protection of sensitive policies can be obtained only by providing support to distributed policy evaluation, like the one carried out by PeerTrust, Protune or PSPL

Evidences The result of a policy's evaluation may depend on the identities or other properties of the peer who requested for its evaluation: a means needs hence to be provided in order for the peers to communicate such properties to each other. Such information is usually sent in the form of digital certificates signed by trusted entities (*certification authorities*) and called *credentials*. Credentials are a key element in Cassandra, *RT* and TPL, whereas they are unnecessary in Ponder, whose policies are concerned with limiting the activity of users who have already been successfully authenticated. The authors of PSPL were the first ones advocating for the need of exchanging non-signed statements (e.g., credit card numbers), which they called *declarations*; declarations are supported by PeerTrust and Protune as well. Finally, EPAL, KAoS, Rei, WSPL and XACML do not support evidences

Negotiation support As stated above, we use a narrower definition of negotiation than the one provided in (Anderson, 2004), into which WSPL does not fit, therefore only pretty few languages support negotiation in the sense we specified above, namely Cassandra, PeerTrust, Protune and PSPL

Policy engine decision The evaluation of a policy should end up with a result to be sent back to the requester. In the easiest case such result is a boolean stating whether the request was (not) accepted (and thereby accomplished): KAoS, PeerTrust, Ponder, PSPL, *RT* and TPL conform to this pattern. Besides `permit` and `deny` WSPL and XACML provide two other result values to cater for particular situations: `not_applicable` is returned whenever no applicable policies or rules could be found, whereas `indeterminate` accounts for some error which occurred during the processing; in the latter case optional information is available to explain the error. A boolean value, stating whether the request was (not) fulfilled, does not make sense in the case of an obligation policy, which simply describes the actions which must be executed as soon as an event happens, therefore besides the so-called *rulings* `allow` and `deny` EPAL defines a third value (`don't care`) to be returned by obligation policies; one of the elements an EPAL policy consists of is a global condition which is checked at the very beginning of the policy evaluation: not fulfilling such a condition is considered an error and a corresponding error message (`policy_error`) is returned; a further message (`scope_error`) is returned in case no applicable policies were found. Cassandra's request format contains (among others) a set of constraints c belonging to some constraint domain; the response consists of a subset c' of c which satisfies the policy; in case $c' = c$ (resp. c' is the empty set) `true` (resp. `false`) is returned. Protune allows for more advanced explanation capabilities: not only is it possible to ask why (part of) a request was (not) fulfilled (`Why` and `Why-not` queries respectively), but the requester is even allowed to ask since the beginning which steps she has to perform in order for her request to be accomplished (`How-to` and `What-if` queries). A rudimentary form of `What-if` queries is supported also by Rei obligation policies: the requester can decide whether to complete the obligation by comparing the effects of meeting the obligation (`MetEffects`) and the effects of not meeting the obligation (`NotMetEffects`)

Extensibility Extensibility is a fuzzy concept: almost all languages provide some extension points to let the user adapt the language to her current needs. Extensibility is described as one of the criteria taken into account in designing Ponder: in order to provide smoothly support to new types of policies that may arise in the future, inheritance was considered a suitable solution and Ponder itself was therefore implemented as an object-oriented language. XACML's support to extensibility is two-fold: (i) on the one hand new datatypes, as well as functions for dealing with them, may be defined in addition to the ones already provided by XACML (ii) as we mentioned above, XACML policies can consist of any number of distributed rules; XACML already provides a number of combining algorithms which define how to take results from multiple policies and derive a single result, nevertheless a standard extension mechanism is available to define new algorithms. Using non-standard user-defined datatypes would lead to wasting one of the strong points of WSPL, namely the standard algorithm for merging two policies, resulting in a single policy that satisfies the requirements of both (assuming that such a policy exists), since there can be no standard algorithm for merging policies exploiting user-defined attributes. Ontologies are the means to cater for extensibility in KAoS and Rei: both KAoS and Rei define basic built-in ontologies, which are supposed to be further extended for a given application. Extensibility was the main issue taken into account in the design of Cassandra: its authors realized that standard policy idioms (e.g., role hierarchy or role delegation) occur in real-world policies in many subtle variants: instead of embedding such variants in an *ad hoc* way, they decided to define a policy language able to express this variety of features smoothly; in order to achieve this goal, the key element is the notion of *constraint domain*, an independent module which is plugged into the policy evaluation engine in order to adjust the expressiveness of the language. A standard interface to external packages is the means provided by Protune in order to support extensibility: functionalities of a component implementing such interface can be called from within a Protune policy. Finally, PeerTrust, PSPL, *RT* and TPL do not provide extension mechanisms

	Cassandra	EPAL	KAoS	PeerTrust	Ponder	Protune	PSPL	Rei	RT	TPL	WSPL	XACML
Well-defined semantics	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	No	No	No
Monotonicity	Yes	–	Yes	Yes	–	Yes	Yes	Yes	Yes	No	–	–
Underlying formalism	Constraint DATALOG	Predicate logic without quantifiers	Description logics	Constraint DATALOG	Object-oriented paradigm	Logic programming	Logic programming	Deontic logic, Logic programming, Description logics	Constraint DATALOG	–	–	–
Action execution	Yes (side-effect free)	Yes	No	Yes (only sending evidences)	Yes (access to system properties)	Yes	Yes (only sending evidences)	No	No	No	Yes	Yes
Delegation	Yes	No	No	Yes	Yes	Yes	No	Yes	Yes (RT^D)	No	No	No
Type of evaluation	Distributed policies, Local evaluation	Local	Local	Distributed	Local	Distributed	Distributed	Distributed policies, Local evaluation	Local	Local	Distributed policies, Local evaluation	Distributed policies, Local evaluation
Evidences	Credentials	No	No	Credentials, Declarations	–	Credentials, Declarations	Credentials, Declarations	–	Credentials	Credentials	No	No
Negotiation	Yes	No	No	Yes	No	Yes	Yes	No	No	Yes	No (policy matching supported)	No
Result format	A/D and a set of constraints	A/D, `scope error`, `policy error`	A/D	A/D	A/D	Explanations	A/D	A/D [a]	A/D	A/D	A/D, `not applicable`, `indeterminate`	A/D, `not applicable`, `indeterminate`
Extensibility	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	No	No	No	Yes

Table 1: Policy language comparison (“–” = not applicable)

[a] For obligation policies a rough version of “What-if” query is available.

6 Discussion

In this section we review the comparison performed in section 5 and provide some general comments.

By carrying out the task of comparing a considerable amount of policy languages, we came to believe that they may be classified in two big groups collecting, so to say, *standard-oriented* and *research-oriented* languages respectively. EPAL, WSPL and XACML can be considered standard-oriented languages since they provide a well-defined but restricted set of features: standard-oriented languages are hence a good choice for users who do not need advanced features but for whom compatibility with standards is a foremost issue. Ponder, *RT* and TPL are somehow placed in between: on the one hand Ponder provides a complete authorization solution, which however takes place after a previously overcome authentication step, therefore Ponder cannot be applied to contexts (like pervasive environments) were users cannot be accurately identified; on the other hand *RT* and TPL do not provide a complete authorization solution, since they can only map requesters to roles and need to rely on some external component to perform the actual authentication. Finally research-oriented languages strive toward generality and extensibility and provide a number of more advanced features in comparison with standard-oriented languages (e.g., conflict harmonization in KAoS and Rei, negotiations in Cassandra, PeerTrust and PSPL or explanations in Protune); they should be hence the preferred choice for users who do not mind about standardization issues but require the advanced functionalities that research-oriented languages provide.

7 Conclusions

Policies are a well-known approach to protecting security and privacy of users as well as for flexible trust management in distributed environments. In the last years a number of policy languages were proposed to address different application scenarios. In order to help both developers and users in choosing the language best suiting her needs, policy language comparisons were proposed in the literature. Nevertheless available comparisons address only a small number of languages, are either out-of-date or too narrow in order to provide a broader picture of the research field. In this paper we considered twelve relevant policy languages and compared them on the strength of eleven criteria which should be taken into account in designing every policy language.

By comparing the choices designers made in addressing such criteria, useful conclusions can be drawn about strong points and weaknesses of each policy language.

REFERENCES

Anderson, A. H. (2004). An introduction to the web services policy language (wspl). In *POLICY 2004*. IEEE Computer Society.

Anderson, A. H. (2006). A comparison of two privacy policy languages: Epal and xacml. In *SWS 2004*. ACM Press.

Ashley, P., Hada, S., Karjoth, G., Powers, C., and Schunter, M. (2003). Enterprise privacy authorization language (epal 1.2). Technical report.

Becker, M. Y. and Sewell, P. (2004). Cassandra: Distributed access control policies with tunable expressiveness. In *POLICY 2004*. IEEE Computer Society.

Bonatti, P., Olmedilla, D., and Peer, J. (2006). Advanced policy explanations. In *ECAI 2006*. IOS Press.

Bonatti, P. and Samarati, P. (2000). Regulating service access and information release on the web. In *CCS 2000*. ACM Press.

Damianou, N., Dulay, N., Lupu, E., and Sloman, M. (2001). The ponder policy specification language. In *POLICY 2001*. Springer.

Duma, C., Herzog, A., and Shahmehri, N. (2007). Privacy in the semantic web: What policy languages have to offer. In *POLICY 2007*. IEEE Computer Society.

Gavriloaie, R., Nejdl, W., Olmedilla, D., Seamons, K. E., and Winslett, M. (2004). No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *ESWS 2004*. Springer.

Herzberg, A., Mass, Y., Michaeli, J., Ravid, Y., and Naor, D. (2000). Access control meets public key infrastructure, or: Assigning roles to strangers. In *2000 IEEE Symposium on Security and Privacy*. IEEE Computer Society.

Kagal, L., Finin, T. W., and Joshi, A. (2003). A policy language for a pervasive computing environment. In *POLICY 2003*. IEEE Computer Society.

Li, N. and Mitchell, J. C. (2003). Rt: A role-based trust-management framework. In *DISCEX III*. IEEE Computer Society.

Lorch, M., Proctor, S., Lepro, R., Kafura, D., and Shah, S. (2003). First experiences using xacml for access control in distributed systems. In *XMLSEC 2003*. ACM Press.

Seamons, K. E., Winslett, M., Yu, T., Smith, B., Child, E., Jacobson, J., Mills, H., and Yu, L. (2002). Requirements for policy languages for trust negotiation. In *POLICY 2002*. IEEE Computer Society.

Tonti, G., Bradshaw, J. M., Jeffers, R., Montanari, R., Suri, N., and Uszok, A. (2003). Semantic web languages for policy representation and reasoning: A comparison of kaos, rei, and ponder. In *ISWC 2003*. Springer.

Uszok, A., Bradshaw, J. M., Jeffers, R., Suri, N., Hayes, P. J., Breedy, M. R., Bunch, L., Johnson, M., Kulkarni, S., and Lott, J. (2003). Kaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In *POLICY 2003*. IEEE Computer Society.