Computer Networks 56 (2012) 477–489

Contents lists available at SciVerse ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Survey Paper

A survey on policy languages in network and security management

Weili Han *, Chang Lei

Software School, Fudan University, Shanghai 201203, China

ARTICLE INFO

Article history:
Received 13 May 2011
Received in revised form 21 August 2011
Accepted 20 September 2011
Available online 12 October 2011

Keywords:
Policy language
Policy-driven management
Network management
Security management

ABSTRACT

Policy-driven management is gaining popularity today, mainly due to the ever-growing scale and complexity of large networked systems. In these systems, policies are usually used to simplify the tasks of the system management, thus making way for further system enlargement. Accordingly, a variety of policy languages are proposed to express intentions of administrators in the policy-driven systems, especially for the network and security management. This paper, therefore, investigates current works, discusses the key issues, and then outlines the future work of policy languages.

1. Introduction

Today's information systems are rapidly growing in scale, and at the same time incorporating various emerging technologies. Facing the resulting complexity, traditional system management (including security management) strategies, which mainly rely on IT professionals' manual work, seem effort-consuming and error-prone for large-scale networks or distributed systems. Especially in today's context, with prevailing technologies such as Cloud Computing [1], large-scale data centers may in the near future consist of millions of machines, taking Google as an example [2].

To resolve these issues, policy-driven management [3] is proposed to be leveraged, to simplify the administration of the large scale systems, including the Cloud Infrastructures [4]. In a policy-driven management system, administrators just need to specify their targets and constraints in the form of policies, to guide the behaviors of system elements. From this perspective, a policy can be seen as a common intermediate format, domain- or device-independent, to map requirements of the system to specific and implementable operations.

Advantages of the policy-driven approach over traditional management methods mainly lie in three aspects. First of all, policies in the policy-driven approach are predefined by policy makers, and stored in a repository point. Once demanded, appropriate policies can be retrieved autonomically and immediately, freeing the administrator from manual countermeasure on a per-event basis. Secondly, the formal foundation of most policy languages introduces automated analysis and verification of policies, with the purpose of ensuring consistency and providing explanation to some extent. Thirdly, because of the abstraction from lower technical details, policies in the policy-driven approach can be inspected and changed dynamically at run time, without changing the underlying implementation.

This paper, therefore, overviews policy-driven management, and investigates the status quo of the researches and development of policy languages.

We discuss the languages from two categories: network management policy languages and security management policy languages. The former focuses on how to drive the network devices and resources to meet system requirements, e.g. SLA (service level agreement) assignments, whereas the latter focuses on the protection of system resources, including users' privacy.

The rest of this paper is organized as follows: Section 2 overviews the policy-driven management mechanism; Section 3 introduces the typical construction of a policy language; Section 4 illustrates two classifications of

* Corresponding author.
E-mail address: wlhan@fudan.edu.cn (W. Han).

1389-1286/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2011.09.014

existing policy languages, and analyzes the policy languages respectively; Section 5 discusses some important issues in the policy languages; Section 6 outlines the future work of policy languages; Section 7 gives out a summary to this paper.

2. Overview of policy-driven management

2.1. Typical architecture

According to the PCIM (Policy Core Information Model) [5] developed jointly by IETF and DMTF, the policy-driven management consists of three key components: PR (Policy Repository), PDP (Policy Decision Point), and PEP (Policy Enforcement Point). Besides, there may exist an additional PAP (Policy Authorization Point), facilitating formulation, analysis and verification of polices on the part of humans. In such an architecture, as is shown in Fig. 1, policy makers could pre-define management policies through a PAP, then deposit them in a PR. After that, a PDP will be monitoring the system. Once the specific events occur, they will trigger the PDP to retrieve the PR for applicable policies. According to these policies, when the specific conditions are met, the corresponding actions should be enforced by the related PEP. COPS (Common Open Policy Service) [6] or SNMP (Simple Network Management Protocol) [7] can be used to communicate between a PDP and a PEP. And LDAP (Lightweight Directory Access Protocol) [8] can be used by a PDP to retrieve a PR.

Fig. 1 shows a typical policy-driven management architecture. In a real system, actually, these components may be more than one, and the location of them could be centralized or decentralized. For example, in a large organization, there may be several PAPs, to authorize polices to a single PR; or a PDP can be connected to several PEPs to enforce policies.

Policies can be grouped according to different criteria. Nowadays it is generally accepted that there are two main kinds of policies: low-level policies, which are more domain- or device-related, and high-level policies, which are more user-friendly. In addition, Kephart et al. [9] have further divided high-level policies into Goal policies and Utility Function policies, along with low-level Action policies, making it three categories in total. Thus in this perspective high-level goals such as enterprise's business objectives, SLAs, can be specified as high-level policies, either Goal or Utility Function ones. For example, as in the data center scenario exemplified in [9], a high-level Goal policy can take the form of *Response time of Gold Class should be less than 100 ms*, and a high-level Utility Function policy can be *Maximize the sum of Gold and Silver classes' utilities*. In a practical implementation, these high-level requirements are then refined into low-level policies, which are more direct, technical, and can be assigned to and operated by agents (devices, human users, or software programs). Often low-level policies are Action policies, taking the form of *IF (Condition) THEN (Action)*, such as *IF (Gold_Class. Response_-Time > 100 ms)* THEN (increase CPU by 5%).

A policy-driven system can be applied in a wide spectrum of domains, such as QoS management within a network, e.g. Cisco QoS Policy Manager [10]; internet services access control, e.g. the access control model for web services [11] based on XACML [12]; privacy management for web users, e.g. P3P [13] and APPEL [14]; access and transmission management of location information over the Internet, e.g. IETF's Geopriv [15]; call control in telephony [16], e.g. the ACCENT project [17]; enterprise modeling, e.g. RM-ODP [18]; and most recently, management of the Cloud architecture [4].

The policy-driven management approach is researched in both industry and academia. In the industry area, several commercial tools have been developed, mainly based on the PCIM framework. In the academic area, researchers have formulated many policy languages to model system elements and specify various policies.

2.2. Existing implementations

Here we list some commercial tools for realizing the policy-driven management.

Basically, the common ground of them is the architecture, which is based on the above mentioned PCIM framework. The PCIM framework consists of three components: a user interface (acts as a PAP), a policy server (acts as a PDP and a PR), and an agent (acts as a PEP). Administrators can define and edit policies through the user interface, in the form of *IF (Condition) THEN (Action)* rules compliant with the PCIM. And then the policy server will monitor the system and trigger actions when conditions are met. Finally these actions are enforced by the agents, e.g. specific applications or physical devices.

There are many differences among the tools, and our concern in this paper is the various languages they use to model policies. In this aspect, they either adopt their own

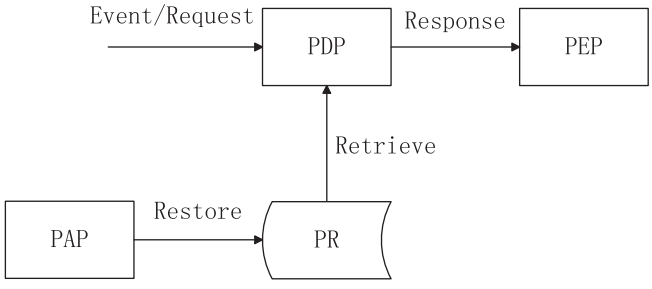

Fig. 1. Policy-driven management architecture.

specific languages, such as HP Openview PolicyXpert and CiscoAssure do, or partly/fully support one or more standard languages. For example, IBM Tivoli supports the OASIS standard XACML, and Axiomatics can fully support XAMCL. To the best of our knowledge, no other critical assessment research on other aspects of these implementations exists. In other aspects, we believe these commercial implementations are specially developed to manage their own networks and information systems, thus we leave it as future work to compare them from other perspectives such as performance.

- *IBM Tivoli*: Tivoli [19] is a family of policy-driven products for the IT infrastructures management, including resource allocation, security and privacy protection, storage management. The policy-driven authorization and authentication engines used by the series of Tivoli products are Tivoli Access Manager and Tivoli Security Policy Manager.
  As one product of the Tivoli family, Tivoli Security Policy Manager [20] can define and enforce security policies in a policy-driven system, whose architecture just complies with the typical PCIM architecture as is shown in Fig. 1. In the aspect of policy representing, it supports open and standard formats, i.e., OASIS WS-Security [21] for message protection policies, and OASIS XACML (eXtensible Access Control Markup Language) [22] for authorization policies.
  Another Tivoli product, Tivoli Access Manager [23], basically supports the PCIM architecture as well. It checks an access to a resource according to the ACL (Access Control Lists) and POP (Protected Object Policies) attached to objects, which are stored in an authorization database. In a simple access evaluation process, the Access Manager checks the requester's identifier against the requested object's ACL, to determine whether to grant a permission, by using an evaluation scheme on ACLs. In addition, ACL can be translated into XACML as well [24].
  According to the report from literature [23], the Web Portal Manager supported Role-Based Access Control as is described in Section 4. Even, the Privacy Manager, another extension of Tivoli Access Manager, added support for "dynamic roles".
- *HP Openview PolicyXpert*: HP Openview is also a family of products for system and network administration, with PolicyXpert [25] one of them.
  PolicyXpert provides the policy-driven network management solution where service-level agreements in a heterogeneous network environment can be configured and controlled in standard [26]. PolicyXpert performs this task by creating business level policies, and then translating and enforcing them as low-level configurations.
  PolicyXpert is also built upon the typical policy management architecture in Fig. 1. A PAP is implemented as a console, which is an interface for users to create, edit and deploy policies. A policy server performs the task of a PDP and a PR, for policy storage, distribution and decision-making. A PEP is a vendor-specific policy agent, which enforces policies by making configuration changes to resources. COPS (Common Open Policy Service) [6] is supported for the sake of communication between these components.
  But, unlike IBM Tivoli, no literature has ever reported that a standardized policy language is used in HP Openview PolicyXpert.
- *Cisco CiscoAssure*: CiscoAssure [10] is a family of different policy-based tools, i.e., QoS Policy Manager, Cisco Secure Policy Manager and Cisco Network Registrar.
  Cisco QoS Policy Manager is based on the typical model as well, and supports COPS protocol. Through this tool, network administrators specify QoS requirements or SLAs in the form of *IF (Condition) THEN (Action)* rules, according to the PCIM standard, and then store them in the PR of a policy server. The physical actions enforced by PEPs are finally applied to routers by using CLI (Command Line Interface), a low-level language supported by Cisco hardware. Multi-vendor interoperability is also provided, through the support for COPS standard to carry the policy information within the network.
  Besides QoS related management, this tool also supports access control over devices using ACLs (access control lists).
  Like HP Openview PolicyXpert, no literature has ever reported that a standardized policy language has been used in Cisco CiscoAssure yet.
- *Axiomatics*: Axiomatics [27] is the first series of products that fully support XACML 2.0 and 3.0. Thus users could readily use the full range of capabilities that XACML provides.
  There are three Axiomatics products in this family: Axiomatics Policy Server, Axiomatics Policy Auditor, and Axiomatics Policy Enforcement Points (PEP). Slightly different from the typical architecture of this kind of tools, Axiomatics Policy Server combines the functionalities of PAP and PDP. Axiomatics Policy Auditor is a separated tool for users to analyze and verify policies. Users can query information they want to know about policies such as the applying circumstances or effects of a policy, and the Axiomatics Policy Auditor will retrieve the PR and related environment knowledge in response to users.
  Benefiting from the open industry standard XACML, all access polices are written in a unified format, which enables various components in the access control system to understand one another, thus saving the interpretation efforts to some extent. Besides, newly added applications can work with the existing system easily, since their policies all comply with the XACML standard.
  Axiomatics is distinguished from the above-mentioned tools in that it is specially tailored for the XACML standard.

3. Construction of policy languages

An abstract policy is implemented through the concrete rules that construct it. Usually these rules are stored in a PR as basic management units, for the policy makers to use and re-use to form multiple policies. Generally speaking,

rules in most policy languages are based on the following paradigms: *Event–Condition–Action* paradigm and *Condition–Action* paradigm.

3.1. Event–Condition–Action Paradigm

The ECA rule is originally a technique from the research area of active database, [28] and has gradually been adopted as a domain-independent paradigm for various policy languages, such as the cases of PDL [29] and Ponder [30]. This ECA paradigm differs from the *Condition–Action* paradigm in that it needs an explicit Event element to trigger the execution of an action under certain conditions. Thus, in the ECA paradigm, a typical rule is composed of three key parts:

ON (Event) IF (Condition) THEN (Action)

- *Event*: The Event, which is under monitoring of the system, is the part to be specified when the rule should be triggered. (e.g. *the arrival of a new packet* as an event). Event is either primitive or composite. And in PDL [29], primitive Events are further classified into two basic types: system-defined primitive Event and policy-defined primitive Event. The former is pre-defined by the system, and the latter is tailored by the user. A composite Event is made up of conjunction of single events, negation of an event, or a sequence of events occurring in a predefined order.
- *Condition*: When the Event is caught, the associated states or parameters of the system should be checked against the Condition contained in the policy rule. Condition is usually a predicate expression that can be evaluated to *True*, *False* or *Not Applicable*. A Condition can be time, an application type, an IP address, etc. Conjunction, disjunction and negation of predicates can form a Condition as well. If the Condition is met, the corresponding actions should be executed.
- *Action*: The Action part of a policy rule can be a single action or a sequence of actions. As a consequence of the action(s), the current system state, once its attributes meet the Condition, may transform to another state. Note that the resulting state of the system is not explicitly specified, thus consideration should be taken by policy makers while designing the action(s)-triggered transition pattern from state to state.

3.2. Condition–Action paradigm

In both IETF and DMTF's PCIM [5] and DMTF's CIM [31], a policy is applied by using a set of rules, and each policy rule is constructed by a set of conditions and a set of actions. Thus this kind of policy rule follows an *IF (Condition) THEN (Action)* paradigm. When conditions associate with a policy rule are evaluated to be true, corresponding actions can be executed, either to maintain current state of an object, or transit to another state.

3.3. Comparison of paradigms

In general, policy languages which follow an *IF (Condition) THEN (Action)* paradigm are also event-triggered, although lacking an explicit Event element in their declaration. However, we recommend the ECA paradigm because of the extra Event element it brings, and it outmatches the *IF (Condition) THEN (Action)* policies at least in these aspects:

- The Event element in the ECA paradigm could cover more events in variety. In most cases of the *IF (Condition) THEN (Action)* paradigm, a rule's implicit event element is limited in scope, because these policies are not designed to catch various events. Here we distill the two most typical implicit events the Condition–Action paradigm can offer:
  - Request-triggered rule evaluation.
    In access control languages using the *IF (Condition) THEN (Action)* paradigm such as XACML [12], the Condition of a policy rule is evaluated when requests come, and these requests submitted by the subject can be seen as implicit events.
  - Periodically-triggered rule evaluation
    In some rules of the Condition–Action paradigm, the Condition is periodically evaluated to check the status of a system element, at regular intervals.

If encoded in ECA, these implicit events of Condition–Action paradigm can be readily defined as specific Events. Besides, ECA can also cover events that are related to environment attribute changes, to provide an asynchronous notification mechanism, whereas Condition–Action paradigm cannot.

- The Event element could be separated from Condition and Action, to provide a more flexible and composite Event. For example, in CIM-SPL [32], the specification and evaluation of Event is separated from the specification and evaluation of the rest of the policy, while an additional event correlation engine or invocation method is provided to trigger policies.

In this survey, we classify a policy language to be ECA rule-based only if it declares to be so. Basically this means its Event part supports various events, providing an asynchronous notification mechanism to the system; and this Event part is incorporated with the Condition and Action part in an ECA rule at the same time.

4. Policy languages for network and security management

According to their application scenarios, policy languages can be grouped into different classifications. Through our investigation, we select two typical categories which are policy languages for network management and policy languages for security management.

Although security management could be considered part of network management, here we separate policy languages for security management from those for network management, because of the different emphasis their modeled policies lay on. As for the network management category, we mean policy languages whose aims are to allocate resources, e.g. bandwidth, CPU time, and the configuration of devices within a network according to the system

requirement, e.g. SLA assignments. And for the security management, the policy languages we discuss focus on security only, especially access control and privacy. Although several general-purpose policy languages such as Ponder [30] could express policies for both network management and security management, still many policy languages focus on security aspects only such as access control, e.g. the OASIS standard XACML [12], privacy policies, e.g. APPEL [14], and security goals, e.g. VALID [33].

The following languages we choose to introduce are either proposed by standard organizations such as IETF and OASIS; or widely referred to, such as Rei [34].

We profile the investigated languages according to six critical features listed in Table 1. These features are selected as evaluation criteria because of their respective merits:

- *ECA*: This refers to whether a language follows the ECA paradigm to express its policies. If the language is not ECA, then it is based on Condition–Action paradigm.
  As is mentioned in Section 3, the ECA paradigm can deal with more complicated events, such as events with asynchronous notification, and composite events. Thus, it is a good construction to design a general-purpose policy language.
- *XML*: This refers to whether the representation of this language is encoded in the XML format.
  Since XML is an international standard which is expressive and extensible, we view it beneficial to encode a policy language in this format, to advance usability and facilitate interchange.
- *Index*: This refers to whether there exists a special item for policy engine to retrieve the required policies more efficiently.
  This item could help to increase retrieval efficiency, thus to improve the performance of the PDP.
- *RBAC*: Role-Based Access Control [35] is widely used in the policy-based access control management. RBAC uses roles to bridge subjects and permissions. A subject can be assigned roles, and roles can be assigned permissions. Due to the stability of roles in a system or organization, the task of the administrators will be simplified by RBAC. Thus, a policy language supporting RBAC can better help administrators in the security management.
- *Obligation*: This refers to whether the policy language can trigger tasks that *must* be performed, once some Event occurs and the related Condition is met.
  These actions are triggered like *side effects* of the occurring Event or applied Condition. For example, when the access to some resources is permitted, the subject is obliged to return the resources in time, and forbidden to leak it out.
  In most cases, policies express *what an element can do*, which is an authorization, and *what an element must do*, which is an obligation. Thus obligations are necessary under some circumstances. As a result, we think a policy language is not expressive enough if it lacks obligation policies.
- *Formalization*: This refers to whether this policy language has a formal foundation. It can either directly express policy rules in formulas of some logic such as first-order logic, or it can be translated into a formal language.
  This feature directly decides whether this policy language could leverage automated reasoning, for conflict check, goal refinement, etc.

4.1. Policy languages for the network management

From the perspective of network operation, the policy-driven network management aims at reducing the complexity of network management tasks. With policies, the system gains some autonomy, saving the administrator's manual work to some extent. A typical usage in this domain is the QoS management.

IETF and DMTF have jointly proposed a model to express policy information used in a network, i.e., PCIM (Policy Core Information Model) [5]. The PCIM can be mapped into network device configurations, and it can also be utilized as a support for higher-level languages. Some existing commercial tools adopting the PCIM policy framework have been discussed in Section 2. Besides these comprehensive tools in the industry area, several languages targeted at the policy-driven network management have also been proposed by researchers, and the typical ones are listed as follows.

4.1.1. PFDL

PFDL (Policy Framework Definition Language) [36] is a network administration definition language, which is originally proposed as a necessary part of the PCIM model. However, the IETF policy group then suspended further work on PFDL, because they realized that the language part

Table 1
Features in policy languages.

	ECA	XML	Index	RBAC	Obligation	Formalization
PFDL						
PDL	✓				✓	✓
Ponder	✓			✓	✓	
CIM-SPL						
KAOS						✓
XACML		✓	✓		✓	
Rei					✓	✓
EPAL		✓	✓	✓	✓	
P3P/APPEL		✓				
ASL				✓		✓
VALID						✓

of the PCIM is too flexible to reach a consensus within the available time.

As a general-purpose language proposed to fit into the PCIM model, the PFDL rule is specified through a management console (PAP) and used in a prototypical policy server (PDP), to implement a centralized view of administrator with regards to QoS, access control, and customized web applications management.

PFDL is an aggregation of a set of policy rules, which are organized in a *IF (Condition) THEN (Action)* paradigm. In the hierarchy defined in IETF's draft [37], the classes from higher level to lower level ones are: ComplexPolicy, SimplePolicy, PolicyRules, in which the former classes can contain one or more of the latter classes. And each PolicyRules class contains a set of PolicyConditons and PolicyActions classes.

4.1.2. PDL

The Policy Description Language (PDL) [29], proposed by Bell Labs, is one of the first policy-based management languages, mainly for network administration. It is declarative, domain-independent, and formulated as ECA rules. In its ECA rule, Condition is just a set of simple predicates, Events are divided into primitive ones and composite ones. The latter events can be constructed by the former events by using boolean operators (AND, OR, etc.) or temporal operators. A formal framework is developed, to evaluate policies specified in PDL, and a PEP is proposed to communicate between devices and policy servers [38]. An approach to translate PDL policies into logic programs is also proposed to detect and resolve conflicts among actions, according to action constraints [39].

There are two main flaws in PDL. On the one hand, Rules cannot be hierarchically grouped, i.e., a policy has a flat structure merely, without role or any other concept. On the other hand, there is no specific order of rules in a PDL policy.

4.1.3. Ponder

Ponder [30] is a declarative and object-oriented policy language from Imperial College. It can specify security policies using Role-Based Access Control, and general management policies for distributed systems. Unlike PDL, a key feature of Ponder is that related policies are highly grouped into roles. And the interaction between roles are defined as relationships. This structure could better facilitate the reuse and flexibility of policies.

There are five types of policies in Ponder: Authorization policies, Filter Policies, Refrain policies, Delegation policies, and Obligation policies. The former four types are targeted at defining access control policies, and Obligation policies are used to support ECA paradigm to perform management actions in the policy-driven management, i.e., when specific events occur and conditions are applied, a subject *must* perform some actions on target objects. In this aspect, Ponder's Obligation policies are similar to PDL. In comparison we can see Ponder has a broader scope in the variety of policies.

Although it is not XML-based, Ponder is easy to be translated into XML representation, for a standard XML browser or for exchange of policies across different domains [30]. As for conflict resolution, Ponder utilizes meta-policies to prevent conflicting actions [30]. In addition, a PEP is provided, taking the form of an enforcement agent in a deployment model of Ponder written in Java [40].

4.1.4. CIM-SPL

CIM-SPL (Simplified Policy Language for CIM), [41] is a policy language proposed specially to complement and render the policy model included in CIM (Common Information Model) standards of DMTF (Distributed Management Task Force), and is a DMTF standard submitted by the DMTF Policy Working Group.

CIM [31] is an object-oriented model defined by the DMTF to describe various components of IT infrastructure, such as systems, networks, applications and services. It also includes a policy model to define policy objects in the systems, and these policy objects can be grouped and structured.

CIM-SPL is an abstract model of policy language, which follows an *IF (Condition) THEN (Action)* rule structure. It is designed to be compliant with the CIM and CIM Policy Model, hence the special feature of CIM-SPL is that it can be fully incorporated into the CIM data structures, and its implementations can naturally draw support from existing methods designed for CIM. Therefore, in each CIM-SPL rule, there is an additional Import field besides the Condition and Action. The field could refer to existing CIM classes relevant to the policy.

CIM-SPL has an external Event Model [42], to define the Event-related abstractions, and describe the CIM Indication hierarchy which is used to model Events that can be detected. In its mechanism, when the policy server receives a CIM InstIndication, a subclass of CIM Indication which denotes the occurrence of an action, the evaluation of the corresponding policy will be initiated.

Although CIM-SPL is a general purpose policy language, it does not directly define access control policies. A method combining RBAC with the CIM-SPL is proposed as an extension of it [43].

Finally, an open source implementation of a CIM-SPL policy engine has been started in the Open Group's OpenPegasus project [44].

4.1.5. KAOS

KAOS (Knowledge Acquisition in autOmated Specification) [45] is a language initially designed for goal-directed software requirements analysis. It can also be selected by the policy-driven system to express high-level goals.

Distinguished from previously mentioned languages, KAOS is a typical Goal policy language, not an Action one, as is described in [45]. As a typical Goal policy language, KAOS provides the capability to assign system-level and organisational objectives rather than lower-level process- or action-oriented descriptions.

KAOS supports a formal specification of the system by using temporal logic. Based on this formal foundation, a necessary refinement pattern can be created, which is used to derive more domain-specific low-level policies, which are often in the form of conjunction or disjunction of subgoals, from high-level goals. Finally, through a goal-regression

process, a series of implementable actions can be acquired for enforcement.

4.2. Policy languages for the security management

Actually, security management is one of the most important aspects in the system and network management. Many policy languages were proposed to express the administrators' intention on the security management.

4.2.1. XACML

Nowadays eXtensible Access Control Markup Language (XACML) [12] is widely accepted both in industry and academia as a *de facto* standard. XACML is a declarative, XML-based policy language, mainly for access control management in distributed systems. It provides standards for both access control policies and access control request-respond format. Latest lease version 2.0 was approved by OASIS standards organization as an international standard on February, 2005, and now version 3.0 is under drafting.

A distinguished feature of XACML is its Target element that is attached to every rule, policy and policy set, for the purpose of policy indexing. Target is defined by the resources, actions, subjects, and environments where the rule or policy applies. In response to a request, a PDP can search for the applicable policies by evaluating their Targets.

Another feature of XACML is that when more than one policy is applied, the strategy to give the overall answer and to avoid conflict is a policy combination procedure, i.e., the combination of actions from multiple policies, with the support of several rule-combining and policy-combining algorithms [46]. Besides, PCL (Policy Combining Language), a formal language specially for policy combination is proposed, which improves the policy combining theory in several ways, particularly the ability to introduce new policy combining algorithms. It is primarily motivated by XACML, and has already been implemented and integrated with Sun's XACML Open Source Implementation [22], a commonly used tool available for the implementation and enforcement of policies encoded in XACML.

Besides, XACML is intended to fit in various application environments. The core XACML language is insulated from specific application domains by XACML context. And the external domain-specific inputs and outputs are out of the scope of XACML specification.

Furthermore, we do not categorize XACML to be ECA rule-based, because the event to trigger the PDP is only an access request, and environment attribute changes are not included. Some researchers have proposed methods to add an event-driven behavior to XACML [47].

Finally, XACML itself only provides limited support for RBAC. However, a policy language combining XACML and RBAC has also been proposed recently, which is xfACL (eXtensible Functional Language for Access Control) [48].

4.2.2. Rei

Rei [34,49] is a declarative policy specification language based on deontic logic. It is concerned with obligation, permission, etc., designed mainly for security and privacy in dynamic and open computing environments. In Rei, policies are defined as constraints over allowable and obligated actions on resources. Next, Rei provides a policy engine to dynamically either allow or deny a request from an entity, after reasoning over policies and related domain knowledge. Meta-policies are also included and enforced in decision making, to make sure the actions to be taken are consistent and conflict-free. In addition, given the characteristics of pervasive environment, Rei uses so-called *speech acts* which include delegation, revocation, request and cancellation, to make the security control of pervasive applications simple and decentralized. Furthermore, Rei is not strictly role-based, because it can specify individual, grouped, and role-based policies at the same time. Finally, implementation also exists to enforce policies defined in Rei [50].

4.2.3. P3P/APPEL

W3C's P3P (Platform for Privacy Preferences Project) [13] and APPEL (A P3P Preference Exchange Language) [14] are used for privacy negotiations between a web site and its users, including the collection of web user's preferences.

P3P is ratified as a W3C web standard in Aril 2002. By using P3P, web sites can express their privacy policies that can be retrieved automatically and interpreted easily by user agents [13]. Web sites may declare in P3P why they need users' personal data, what data they collect, how long they will retain them, and who will use these data, etc. These elements are standardized in a policy, which is applied to a specific set of data resources. Then the web users can be informed of web sites' data-collection policies, which are encoded in the machine-readable XML format, with the help of a user agent such as P3P-enabled web browsers. In this way the user can rely on his or her agent to read and evaluate web sites' policies on behalf of him or her, and to further opt-in or opt-out data sharing decisions.

APPEL can complement P3P in that it describes a collection of a user's privacy preferences regarding P3P policies, although it is not a W3C standard yet, and not required as a must in a user agent. Nevertheless, the user can adopt APPEL to express his or her preference rules, and then his user agent can make automated or semi-automated decisions on accepting or not accepting web sites' machine readable privacy policies from P3P-enabled web sites, by comparing web sites' declared practices with web user's preference rules. These decisions could be given out by simply informing the user, or prompting him for a decision.

XPref [51], another small preference language based on a subset of XPath [52], is proposed as an alternative to APPEL, to overcome some shortcomings in APPEL's fundamental design choices. A translator is also provided to translate APPEL to XPref.

Several tools exist for both user agents and server-side support [53]. For the user side, Internet Explorer has built-in P3P functionality, but mainly limited in cookie management: web sites may include compact policies in a P3P HTTP header, to declare their policies on how to use cookies; then the browser could refer to these compact policies whenever encountering a cookie. There are also full P3P user agents, such as AT&T's Privacy Bird [54], which has an additional support for APPEL as well. For the server side, the above mentioned tool, the IBM Tivoli

Privacy Manager for e-business, supports P3P privacy policies in its e-business applications and infrastructure. Other P3P policy editors adopted by many enterprises are P3PEdit [55], P3PWriter [56], etc.

P3P/APPEL are two XML-based languages, thus can be easily to accepted in the Web. But some important features, such as RBAC, Obligation, are not included.

4.2.4. EPAL

Enterprise Privacy Authorization Language (EPAL) [57] is an XML-based language proposed by IBM, and the latest version, EPAL1.2 was submitted to W3C in November 2003 for consideration as a W3C standard, but has not been approved so far. EPAL is similar to XACML to some extent, but the difference is that EPAL is targeted at privacy policies only, not general access control policies such as XACML, thus EPAL is not so broadly applicable and deployed as XACML, either. As a specific privacy policy language, a feature of EPAL is that rather than adopting traditional RBAC strategy, it uses purpose-based access control. The authorization decision is evaluated directly by the subject's purpose of using the requested resource. For example, your physician can access your health information while the physician's purpose is medical treatment. This access control mechanism simplifies the traditional RBAC, but requires a well-structured Purpose element. IBM has provided tools to implement and enforce EPAL, and some degree of policy refinement and conflict resolution has also been included [58].

4.2.5. ASL

ASL (Authorization Specification Language) [59] is an access control language based on first order logic. In its security model, ASL consists of essential RBAC components, i.e., users, roles, and objects.

An authorization policy rule expressed by ASL is a mapping from the four-tuple: *a user, a role set, an object, and an action*, to the access decision *authorized*, or *denied*. From this view we note the decision making mechanism of ASL lacks sufficient flexibility and reusability, because the authorization decision is encoded in the rule itself, attached with a role set, rather than dynamically determined by a PDP explicitly.

An authorization specification of ASL is composed of a set of rules, i.e., authorization rules, derivation rules, resolution rules, access control rules, and integrity rules. It is written in a stratified Datalog program, which is a formal language based on first order logic. These rules will be evaluated according to its semantics, once an access request comes. This checking process can be performed in linear time with respect to the number of rules in the authorization specification [60].

Given that different models may make different decisions on authorization derivation and conflict resolution, ASL allows the specification of arbitrary rules about these issues. Jajodia et al. [59] also demonstrated how various approaches can be represented in ASL.

4.2.6. VALID

VALID (Virtualization Assurance Language for Isolation and Deployment) [33] is a security assurance language to express high-level security goals, with the purpose to mitigate configuration problems in virtualized infrastructures like clouds. This language is based on the tool-independent IF (Intermediate Format) [61], which gives a formal foundation for VALID to facilitate automated reasoning.

The main feature of VALID lies in that it is the first formal security assurance language for virtualized topologies. It stands out from its counterparts for two reasons: on the one hand, it focuses on high-level security goals from a topology perspective, to complement the local policies. On the other hand, it takes virtualization into account, in addition to traditional complex environments, thus problems like virtual machine isolation are within the range of consideration.

VALID can be applied in three main scenarios. First of all, in the access control cases, it is adopted in combination with low-level XACML language to perform the task. Secondly, in the case of automated deployment, it should be used to enforce the high-level security goals automatically. Thirdly, in the verification case, the high-level security goals are a part of the verification target, against which the system should be evaluated.

5. Key issues

In this section, we discuss five key issues in policy languages and policy-driven management. The policy conflict issue commonly exists in all policy languages; the refinement is a key issue when Goal policies are translated into Action policies; the policy administration in distributed environments is a key issue concerning PAPs; and the last two issues are key issues about PDPs and PEPs respectively.

5.1. Policy conflict detection and resolution

Policy conflict is a general issue which widely exists in all policy languages. According to [37], there are two fundamental types of policy conflicts: intra- and inter-policy conflicts, with the former caused by an ill-defined policy and the latter caused by multiple applied policies leading to conflicting actions. An intra-policy conflict occurs when the conditions in two or more policies are simultaneously satisfied, but a PEP cannot simultaneously execute their actions in these policies [37]. For instance, in an access control scenario, one application policy allows the access request, but the other denies the access request. These two policies lead to the *intra-policy conflict*. In addition, an inter-policy conflict is defined as two or more applicable policies result in conflicting configuration commands and/or mechanisms to networked devices [37]. In the inter-policy conflict case, the conflicting policies do not conflict when compared to each other statistically, but do conflict when assigned to system elements at run time.

Accordingly, there are two ways to resolve this issue: the first intuitive idea is to avoid setting conflicting policies for a network or system beforehand, which is a static method to avoid intra-policy conflicts; the second one is to choose a result or to combine multiple results when the PDP faces conflict results at run time. This is a dynamic

method that could tackle both intra- and inter-policy conflicts.

Most importantly, the first idea should detect policy conflicts automatically. To this end, policies should be in the form of or can be translated into a formal expression. For example, ASL is a language written directly in the first order logic [59], while PDL can be indirectly translated into logic programs [29]. By either means, once in logical representation, methods already developed in this area can be readily adopted for conflict checking. Leveraging this conflict detection, the policy makers can avoid setting conflicting policies by adjusting policy conditions and/or actions before applying policies to the system.

As is shown in Table 1, however, some policy languages, such as PFDL, XACML, are not formal ones. Thus, the second idea is the more suitable way for these policy languages. Literature [37] proposed three methods to resolve the conflicts, whose principle is to choose one policy among others. They are applying a match-first criteria; applying a priority order criteria; using additional metadata. In the method of a priority order criteria, the priorities are inherently linear, whereas in the method of additional metadata, the priorities could be branched. Besides the choosing approach, another approach is policy/rule results combination, as is designed by XACML. Several combination algorithms are also defined: First Applicable, Deny-Overrides, Permit-Overrides, Only-One-Applicable [12].

The second idea can also be used in formal policy languages. For example, Ponder and Rei use additional metadata to resolve the policy conflicts [30,34].

5.2. Refinement for Goal policies

As is described in Section 2, policies are generally divided into high-level policies, which are mostly Goal policies from the use case point of view, including KAOS and VALID; and low-level policies, which are mostly Action policies, including PFDL, PDL, Ponder, CIM-SPL, XACML, Rei, P3P/APPEL, ASL. In the cases of Action policy-driven systems, a mapping is needed to assign actions to concrete objects and devices to finally realize the management. However, in Goal policy-driven systems, an additional refinement process is necessary before mapping into the concrete system elements, because a Goal policy has no detail guidance as low-level system actions, thus the Goal policy must be refined to low-level ones on how to achieve the goal.

Challenges in Goal policy refinement mainly lie in how to automatically derive low-level policies from a high-level policy, and how to validate them to be consistent with the original high-level one. To tackle these issues, two bases are necessary: a formal foundation to construct both the system and the policy model; and some refinement strategies to match the desired goals and concrete system behaviors.

Bandara et al. [62] introduces a policy refinement approach based on Goal-based requirements elaboration and the EC (Event Calculus). The target of this method is to refine high-level Goal policies into low-level ones, and then map them to concrete system elements. First of all, the system modeled by the UML should be translated into the EC, which is a formal language to represent the system. Then for the refinement from high-level goals to low-level operational policies, abductive reasoning is leveraged, to select appropriate strategies to elaborate goals to operations. And finally these operations are assigned to the system objects for enforcement. What is worth noticing is that the concept *strategy* refers to the mechanism by which a given system can achieve a particular goal. And when both the system and the goal have a formal specification, strategies can be inferred through abductive reasoning.

5.3. Policy administration in distributed environments

It is a major task of policy administrators to make policies by using policy languages. According to our investigation, existing policy languages have two main inconveniences for administrators in the process of policy-making.

- First of all, these languages are usually proposed based on a simple structured administration model, which is managed as a whole by a central PAP. However, in real implementations, distributed environments are often large-scale and heteromerous systems, built upon a physically large and topologically complex network. Taking convenience, efficiency and reliability of administration into consideration, large distributed systems are usually divided into various and separate domains, according to either physical locations, business objectives, or other standards. As a consequence, policies should also be administered by domains. This issue now meets two challenges which have had some advancement, but still require more deep researches.
  - *Make administration policies*: It is a hot spot in security policy languages to make administration policies, which assign who can manage the security policies in networks or systems, especially large distributed systems.
    Researchers have proposed some administration models for Role-Based Access Control [63], e.g. ARBAC 97 [64]. Security analysis methods are also proposed for RBAC [65,66] with ARBAC97. The theoretical analysis shows that general URA-SAP (Security Analysis Problem in Role-Based Access Control with the URA97 administrative scheme) is PSPACE-complete. And only some cases in URA-SAP are NP-complete[66]. Thus, it could be hard to develop a commercial tool to analyze the security of RBAC within a distributed administration model. In addition, delegation is widely researched as a special problem in policy administration in distributed environments [67]. To avoid the complexity of theoretical analysis, Han et al. [68] used measurable risk to strengthen the security of delegation in a role-based workflow system.
    The newest version of XACML is developing a profile to administrate its access control policies [69]. But no literature has ever reported the similar work for other policy languages mentioned above.
  - *Composite policies*: Policy composition refers to the problem which arises when distributed policies are authored and merged together.

This problem is especially important in access control policies. Bonatti et al. [70] modeled access control policies as a set of variable-free authorization terms, proposed an algebra to composite authorization specifications originating from different independent parties, and used logic programming and partial evaluation techniques to evaluate algebra expressions. However, their algebra only supports 2-valued (*Permit* and *Deny*) policies, which is not enough since the policy may give a third decision *Not Applicable*, and their approach did not discuss the internal mechanisms of the composition. Hence, Rao et al. [71] proposed a fine-grained composition algebra for language-independent 3-valued policies. Because these researches are highly independent of concrete policy languages, thus the languages, including PDL, CIM-SPL, and XACML can use these researches to compose their distributed policies.

- Secondly, policies from different domains could be reused. We propose this idea because existing languages actually require policy makers to have deep professional knowledge about these languages. However, in reality, the involved policy makers may not be so well-trained.

  To tackle this, it is a necessity to simplify policy authoring for common users. For example, an administrator can refer to the policies authored by other administrators in other locations. Just by justifying some parameters or a part of policies, the administrator can get his wanted policy set. This framework would be necessary for a not well-trained policy maker, such as a common user in APPEL privacy policy authoring.

5.4. Performance of PDP

A PDP, or a Policy Decision Point, is a core conceptual element in the IETF policy-driven management model. A PDP can be seen as a processing engine that can evaluate requests from subjects (in access control scenario), or currently occurring events (in ECA paradigm), and retrieve applicable policies stored in the PR and maybe other knowledge as reference, to give out a final decision to a PEP.

The PDP is so important that once it cannot deliver a high performance when processing complex policies, it would become a bottleneck of the system. Thus it is important to evaluate the policies with a low latency and high throughput.

First of all, it is a general way to add cache mechanism [72], including policy caching and result caching, to improve the performance.

Another method to enhance the PDP's performance is to add LPDP (Local Policy Decision Point) as an assistant to the central PDP, such as the case in the COPS [6]. LPDP stays in the same application or network node with a PEP, rather than in the server as a central PDP does. It frequently backups the decisions of the PDP, so when connection is interrupted between the PDP and the PEP, the LPDP can substitute the PDP to guide the PEP temporarily. But central PDP has the superior authority.

The above methods can be applied to policy languages such as PDL, CIM-SPL, XACML, and Rei. But if the number of policies, such as privacy policies (in APPEL) evaluated on a client platform, is small, the improvement mechanism could be redundant.

In addition, Xengine, a special method to improve the PDP of XACML is proposed by Liu et al. [73]. Xengine transforms the policies before PDP evaluation. After transformation, the Xengine only uses *first-applicable* combination algorithm, to improve the performance, because the PDP is only required to evaluate the first applicable policy rather than all applicable policies.

5.5. Execution in PEP

A PEP, or a Policy Enforcement Point, is another core conceptual element in the IETF model. It is responsible for sending requests to a PDP, and enforcing decisions made by the PDP, thus acting as a link bridge between the PDP and system elements.

To perform its task, on the one hand, it should collect requests from various system elements, and forward them to the PDP for decision-making. On the other hand, it should actually and physically enforce the actions determined and distributed by the PDP. Particularly, this enforcement task is complicated by obligation policies, because this type of actions are often actions that are obliged to be executed in the future, by subjects of either devices, applications, or human users. Enforcement platforms for obligation policies such as Heimdall [74] are proposed, the design principle of which is that although the PEP cannot enforce the execution of a future action, it can compensate for any previously executed actions. For instance, when a system has not complied with its QoS agreement with a client, it might be publicly exposed in a blacklist server until the client gets some refund.

Since the PEP is more domain-specific, it is usually built into the policy-driven system elements directly. The PDP and PEP can be located in the same or different applications and machines, depending on the deployed system model.

6. Future work of policy languages

Through the overview of existing policy languages and key issues, we come up with some ideas that outline the possible future research. There are mainly two aspects that would be further researched, from the basic level of the language itself, to broader level such as policies' applications.

6.1. Policy language

In this part we focus on policy language itself. We should pay attention to the listed aspects when we propose a new policy language or revise an existing language.

- *XML-based*: Taking a policy language's generality and usability into account, it seems that XML is the better choice to define policy languages, since it is an appealing international standard which is proved to be highly expressive and extensible, and many existing supporting tools already exist. Leveraging the XML format, we can analyze and edit policy specifications easily with

the support of a XML browser. In our survey we notice there are some languages adopting the XML format, such as XACML, and P3P/APPEL.

- *Formal expressivity*: As is mentioned in Section 5.3, a formalization feature can help system to detect policy conflicts before policy evaluation. Thus, advancing formal expressivity can strengthen the security of policy languages to some extent. The above-mentioned policy languages, including PDL, KAOS, REI, ASL, are formal ones, but XACML is not.
- *Privacy policy languages*: Privacy protection, as a subject within security management, is becoming increasingly important since resources are highly inter-connected today. Thus we view it as the future work to design privacy policy languages in a fashion that can be understood by common users without much professional knowledge. Moreover, it may be more beneficial if these common users can be engaged in the privacy policy design. This requires the language specifying policies to be not only readable by human, but also negotiable to some extent. Although W3C's APPEL [14], which collects user's preference captured from GUI, can make automated or semi-automated decisions about machine-readable privacy policies, further work should be done to ease the privacy policy authoring burden of common users.
- *Goal and Utility Function policy languages*: Goal policies declare the high-level system goal, such as SLA, and sweep the technical details under the rug. And Utility Function policies, which can always select actions to send the system to the state with the highest utility value, which is superior to the static boolean value set in Goal policies. Since they provide a high view of network and security management, Goal and Utility Function policies can easily be accepted by administrators and common users. However, the advancement in this field is slow. In today's existing languages, KAOS [45] is the most typical Goal language. And the recently proposed VALID [33] is another Goal policy language to express security goals in a Cloud architecture. However, to the best of our knowledge, there are few Utility Function policy languages, except for the Utility Function-based approach integrated into two products, developed by Kephart et al. [75,76].

6.2. Policy language application

Most recently, Cloud Computing and SNS (social networking services) are gaining momentum in both the academic and industry areas, and will probably play leading roles in the future. Thus we consider it promising to leverage the policy-driven approach to facilitate management of these types of large-scale systems, further extending the application spectrum of policy languages.

- *Cloud Computing*: a Cloud Infrastructure provider can leverage policies to govern its network in much the same way as traditional network management does, the Cloud Service provider can specify access control restrictions of its services in the paradigm of policies, and the Cloud Service users can declare their privacy rules in a policy language, too. Thus policy method and related languages may be adopted in various dimensions in the comprehensive Cloud platform. Related research already exists since Cloud Computing has become an important topic [4], such as the above mentioned VALID[33].
- *SNS*: The social networking services, especially the prevailing of websites like Facebook, have introduced a new paradigm of access control: Relationship-Based Access Control (ReBAC) [77], as the monitoring of access is based on the interpersonal relationship between the online owner and receiver, rather than traditional concepts such as role and condition. Hence the required information release pattern in digital social networking is very much like the one in our analog world, where people protect personal data they themselves consider private, and choose who they want to share the information with. Thus policies used to model this paradigm are more flexible and complicated. Most recently, some researchers have focused on the general-purpose ReBAC model and issues about ReBAC policies and related policy languages [78,79].

When policy languages are used in these applications, we argue it is a key problem to standardize policy languages for network management and security management. As is mentioned in Section 2.2, the current implementations except for Axiomatics, usually use their own policy languages as working languages. However, this practice is not suitable for a large and heterogeneous system. Although IBM Tivoli is adopting some standardized policy languages, the advancement of standardization of policy languages should be accelerated.

7. Conclusion and future work

This paper investigates the work on policy languages in the policy-driven management for large scale systems. Due to the mentioned advantages, many researchers and vendors have proposed various policy languages and related implementations. We, therefore, overview the construction of these policy languages, and introduce some softwares as well. Then we propose some indicates to show the features of the policy languages and introduce two types of them: policy languages for network management and policy languages for security management. We also discuss the key issues in policy languages and policy-driven management, and finally we outline the future work of policy languages and their applications.

According to our investigation, we conclude that first of all, policy languages could play an important role in today's large applications, such as the Cloud and the SNS. Secondly, more researches are needed in the area of the policy-driven management mechanism and policy languages. Finally, we notice that only a few implementations are using standardized policy languages, which is a practical problem that should be paid attention to.

In the future work, we will compare the commercial tools in policy languages and performances, investigate how the standardized policy languages are used in the

commercial tools, and explore more policy-driven applications.

Acknowledgements

This paper is supported by "211-Project Sponsorship Projects for Young Professors at Fudan", The Science and Technology Development Project of STCSM (Grant No. 09511500902), and a 863 high-tech Project (Grant No. 2011AA100701). The authors thank the anonymous reviewers of this paper for their helpful comments and suggestions. And we thank Mrs. Li for her English polish.

References

[1] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, M. Zaharia, Above the clouds: A berkeley view of cloud computing, Tech. Rep., 2009.
[2] J. Dean, Desins and Advice from Building Large Distributed Systems, 2009.
[3] D.C. Verma, Simplifying network administration using policy-based management, IEEE Network (2002) 20–26.
[4] P. Goyal, R. Mikkilineni, Policy-based event-driven services-oriented architecture for cloud services operation and management, in: 2009 IEEE International Conference on Cloud Computing, New York, USA, 2009, pp. 135–138.
[5] B. Moore, E. Ellesson, J. Strassner, A. Westerinen, Policy core information model – version 1 specification, RFC 3060, IETF, Febrary 2001. <http://www.ietf.org/rfc/rfc3060>.
[6] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, A. Sastry, The COPS (Common Open Policy Service) Protocol, RFC 2748, IETF, January 2000. <http://www.ietf.org/rfc/rfc2748>.
[7] J. Case, M. Fedor, M. Schoffstall, J. Davin, A Simple Network Management Protocol (SNMP), RFC 1157, IETF, May 1990. <http://www.ietf.org/rfc/rfc1157>.
[8] K. Zeilenga, Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map, RFC 4510, IETF, June 2006. <http://www.ietf.org/rfc/rfc4510>.
[9] J.O. Kephart, W.E. Walsh, An artificial intelligence perspective on autonomic computing policies, in: Proceedings 5th IEEE Workshop on Policies for Distributed Systems and Networks (Policy 2004), New York, USA, 2004, pp. 3–12.
[10] Delivering end-to-end security in policy-based systems, Tech. Rep., Cisco, 1998.
[11] H. Tao, A XACML-based access control model for web service, in: Proceedings of International Conference on Wireless Communications, Networking and Mobile Computing, 2005, pp. 1140–1144.
[12] OASIS, OASIS Extensible Access Control Markup Language (XACML) version 2.0, 2005.
[13] W3C, The Platform for Privacy Preferences 1.1 (p3p1.1) Specification, 2006. <http://www.w3.org/TR/P3P11/>.
[14] W3C, A p3p Preference Exchange Language 1.0 (appel1.0), 2002. <http://www.w3.org/TR/P3P-preferences/>.
[15] IETF, Geopriv Requirements, 2004. <http://www.ietf.org/rfc/rfc3693>.
[16] K.J. Turner, L. Blair, Policies and conflicts in call control, Computer Networks 51 (2007) 496–514.
[17] K. Turner, E. Magill, S. Reiff-Marganiec, L. Blair, J. Pang, Accent (Advanced Component Control Enhancing Network Technologies) Project, 2001–2005. <http://www.cs.stir.ac.uk/accent/>.
[18] Information technology-open distributed processing-reference model-enterprise language, International Standard ISO/IEC 15414, ISO/IEC, 2004.
[19] IBM, 2011. <http://www-01.ibm.com/software/tivoli/>.
[20] IBM Tivoli Security Policy Manager, Tech. Rep., IBM, 2009.
[21] WS-securitypolicy1.3, Tech. Rep., IETF, Febrary 2009. <http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.3/ws-securitypolicy. html>.
[22] Sun, Sun's XACML Open Source Implementation, 2011. <http://www.oasis-open.org/committees/xacml/>.
[23] G. Karjoth, Access control with IBM tivoli access manager, ACM Transactions on Information and System Security (TISSEC) 6 (2003) 232–257.
[24] G. Karjoth, A. Schade, E.V. Herreweghen, Implementing acl-based policies in XACML, in: 2008 Annual Computer Security Applications Conference, 2009.
[25] HP openview policyxpert2.1 product brief, Tech. Rep., HP, 2001.
[26] Openview policy-based network management, Tech. Rep., HP, 1999.
[27] Axiomatics, Axiomatics Products, 2011. <http://www.axiomatics.com/>.
[28] U. Dayal, E.N. Hanson, J. Widom, Active database systems, in: Modern Database Systems, ACM Press, 1994, pp. 434–456.
[29] J. Lobo, R. Bhatia, S. Naqvi, A policy description language, in: Proceedings 16th National Conference on Artificial Intelligence (AAAI-99), Orlando, USA, 1999, pp. 291–298.
[30] N. Damianou, N. Dulay, E. Lupu, M. Sloman, The ponder policy specification language, in: Proceedings of Policy 2001: Workshop on Policies for Distributed Systems and Networks, Bristol, UK, 2001, pp. 18–39.
[31] CIM Policy Model White Paper (CIM version 2.7), Standard, DMTF, 2003. <http://dmtf.org/sites/default/files/standards/documents/DSP0108.pdf>.
[32] D. Agrawal, S. Calo, K.-W. Lee, J. Lobo, Issues in designing a policy language for distributed management of IT infrastructures, in: 10th IFIP/IEEE International Symposium on Integrated Network Management (IM '07), 2007, pp. 30–39.
[33] S. Bleikertz, T. Gross, VALID: a virtualization assurance language for isolation and deployment, in: Proceedings of IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY 2011), 2011.
[34] L. Kagal, Rei: a policy language for the me-centric project, Tech. Rep., HP Labs.
[35] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman, Role-based access control models, IEEE Computer 29 (2) (1996) 38–47.
[36] J. Nicklisch, A rule language for network policies, Position Paper, 1999. <http://www.policy-workshop.org/1999/policy-99/pdf/26-Nicklisch.pdf>.
[37] J. Strassner, S. Schleimer, Policy framework definition language, Internet Draft, IETF, November 1998. <http://www.ietf.org/proceedings/43/I-D/draft-ietf-policy-framework-pfdl-00>.
[38] R. Bhatia, J. Lobo, M. Kohli, Policy evaluation for network management, in: Proceedings of 19th Annual Joint Conference of the IEEE Computer and Communications Societies, Tel Aviv, Israel, 2000, pp. 1107–1116.
[39] J. Chomicki, J. Lobo, S. Naqvi, A logic programming approach to conflict resolution in policy management, in: Proceedings 7th International Conference on Principles of Knowledge Representation and Reasoning (KR 2000), Colorado, USA, 2000, pp. 121–132.
[40] N. Dulay, E. Lupu, M. Sloman, N. Damianou, A policy deployment model for the ponder language, in: Proceedings IEEE/IFIP International Symposium on Integrated Network Management (IM 2001), 2001, pp. 14–18.
[41] CIM simplified policy language (CIM-SPL), International Standard DSP0231, DMTF, 2009.
[42] CIM event model white paper, Preliminary DSP107, DMTF, 2003.
[43] L. Pan, J. Lobo, S. Calo, Extending the CIM-SPL policy language with RBAC for distributed management systems in the WBEM infrastructure, in: Proceedings of the 11th IFIP/IEEE International Conference on Integrated Network Management (IM'09), 2009, pp. 145–148.
[44] T.O. Group, Openpegasus Open Source Project. <http://www.openpegasus.org/>.
[45] A. Dardennen, A. van Lamsweerde, S. Fickas, Goal-directed requirements acquisition, Science of Computer Programming 20 (1993) 3–50.
[46] N. Li, Q. Wang, W. Qardaji, E. Bertino, P. Rao, J. Lobo, D. Lin, Access control policy combining: theory meets practice, in: SACMATa ́f09, ACM Press, 2009, pp. 135–144.
[47] R. Laborden, T. Desprats, Dealing with stable environmental conditions in XACML systems, in: Second International Conference on Systems and Networks Communications (ICSNC 2007), 2007, pp. 63–63.
[48] Q. Ni, E. Bertino, xfacl: An extensible functional language for access control, in: The ACM Symposium on Access Control Models and Technologies (SACMAT 2011), 2011.
[49] L. Kagal, T. Finin, A. Joshi, A policy language for a pervasive computing environment, in: In IEEE 4th International Workshop on Policies for Distributed Systems and Networks, 2003, pp. 63–74.
[50] A. Patwardhan, V. Korolev, L. Kagal, A. Joshi, Enforcing policies in pervasive environments, in: Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitousa ́f04), 2004, pp. 299–308.

[51] R. Agrawal, J. Kiernan, R. Srikant, Y. Xu, Xpref: a preference language for p3p, Computer Networks 48 (2005) 809–827.
[52] XML Path Language (xpath) version 1.0, Standard, W3C, 1999. <http://www.w3.org/TR/xpath/>.
[53] W3C, P3P 1.0 Implementations. <http://www.w3.org/P3P/implementations>.
[54] AT&T, Privacy Bird. <http://www.privacybird.org/>.
[55] L. Code Infusion, P3pedit. <http://p3pedit.com/>.
[56] P3PWriter, P3pwriter. <http://www.p3pwriter.com/>.
[57] IBM, Enterprise Privacy Authorization Language. <http://www.w3.org/Submission/2003/SUBM-EPAL-20031110/>.
[58] M. Backes, B. Pfitzmann, M. Schunter, A toolkit for managing enterprise privacy policies, in: Proceedings of ESORICSa¸f03, LNCS, vol. 2808, Springer, 2003, pp. 162–180.
[59] S. Jajodia, P. Samarati, V.S. Subrahmanian, A logical language for expressing authorizations, in: IEEE Symposium on Security and Privacy, 1997, pp. 31–42.
[60] S. Jajodia, P. Samarati, M. Sapino, V. Subrahmanian, Flexible support for multiple access control policies, ACM Transactions on Database Systems 26 (2) (2001) 214–260.
[61] AVISPA, The intermediate format, Automated Validation of Internet Security Protocols and Applications Deliverable D2.3, AVISPA, 2003. <http://www.avispa-project.org/delivs/2.3/d2-3.pdf>.
[62] A.K. Bandara, E.C. Lupu, J. Moffett, A. Russo, A goal-based approach to policy refinement, in: Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'04), 2004, pp. 229–239.
[63] N. Li, Z. Mao, Administration in role-based access control, in: Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security (AsiaCCS'07), 2007.
[64] R.S. Sandhu, V. Bhamidipati, Q. Munawer, The ARBAC97 model for role-based aministration, ACM Transactions on Information and Systems Security 2 (1999) 105–135.
[65] N. Li, M.V. Tripunitara, Security analysis in role-based access control, ACM Transactions on Information and Systems Security 9 (2006) 391–420.
[66] S. Jha, N. Li, M. Tripunitara, Q. Wang, W. Winsborough, Towards formal verification of role-based access control policies, IEEE Transactions on Dependable and Secure Computing (TDSC) 5 (2008) 242–255.
[67] Q. Wang, N. Li, H. Chen, On the security of delegation in access control systems, in: Proceedings of the 13th European Symposium on Research in Computer Security (ESORICS), 2008, pp. 317–332.
[68] W. Han, Q. Ni, H. Chen, Apply measurable risk to strengthen security of a role-based delegation supporting workflow system, in: IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY 2009), IEEE, London, UK, 2009.
[69] OASIS, XACML v3.0 Administration and Delegation Profile version 1.0, August 2010.
[70] P.A. Bonatti, S.D.C. di Vimercati, P. Samarati, An algebra for composing access control policies, ACM Transactions on Information and System Security 5 (1) (2002) 1–35.
[71] P. Rao, D. Lin, E. Bertino, N. Li, J. Lobo, An algebra for fine-grained integration of XACML policies, in: SACMAT, 2009, pp. 63–72.
[72] K. Borders, X. Zhao, A. Prakash, CPOL: high-performance policy evaluation, in: Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS), ACM Press, 2005, pp. 147–157.
[73] A.X. Liu, F. Chen, J. Hwang, T. Xie, Xengine: a fast and scalable XACML policy evaluation engine, in: Proceedings of the SIGMETRICS'08, 2008.
[74] P. Gama, P. Ferreira, Obligation policies: an enforcement platform, in: Proceedings of the Sixth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY' 05), 2005.
[75] J.O. Kephart, R. Das, Achieving self-management via utility functions, IEEE Internet Computing (2007) 40–48.
[76] R. Das, J.O. Kephart, J. Lenchner, H. Hamann, Utility-function-driven energy-efficient cooling in data center, in: ICAC 2010, 2010, pp. 61–70.
[77] C. Gates, Access control requirements for web 2.0 security and privacy, in: Web 2.0 Security and Provacy (W2SP 2007), 2007.
[78] P.W.L. Fong, Relationship-based access control: protection model and policy language, in: Proceedings of the First ACM Conference on Data and Application Security and Privacy (CODASPY'11), 2011.
[79] P. Fong, I. Siahaan, Relationship-based access control policies and their policy languages, in: Proceedings of The ACM Symposium on Access Control Models and Technologies (SACMAT 2011), 2011.

Weili Han is an associate professor at Fudan University. His research interests are mainly in the fields of Access Control, Digital Identity Management and Service Oriented Architecture. He is now the members of the ACM, IEEE, SIGSAC and CCF.
He received his PhD of Computer Science and Technology at Zhejiang University in 2003. Then, he joined the faculty of Software School at Fudan University. From 2008 to 2009, he visited Purdue University as a visiting professor funded by China Scholarship Council and Purdue.

Chang Lei is a graduate student at Fudan University. Her research interests are mainly in the fields of policy driven management, information security.