

Rule based Security Policy management for Web Service Integration

Hui Dong, Zhimin Wang, Robert A, Morris
University of Massachusetts, Boston
Department of Computer Science
100 Morrissey Blvd Boston, MA 02125, USA
hdong1, wangzm, ram@cs.umb.edu

Abstract

With the prevalence of loosely coupled web service composition in the service oriented infrastructure, distributed security management has become important to offer flexible web security. In our project affiliated with NatureServe [2], we developed a XACML [3, 4] policy oriented security system to provide trust data service to 75 network partners in the Natural Heritage Program [1]. A Horn Logic based rule inference engine extension to XACML model is used to solve the possible policy conflicts over context and semantics in the key decision making step. Approaches to facilitate such decision making process in various ways by using the extension are illustrated.

1. Introduction

With the rapid development of the web and related technology, web service paradigms and standard protocols have become the key approach in distributed environments to relate services through well-defined contracts between different parties across the network. As demand increases for online automated data integration, XML has played an important role in data representation, exchange and information dissemination in web service composition. This interoperability infrastructure allows service providers to publish and aggregate new services more easily based on business flows, however, it also causes new security concerns to protect sensitive data against unauthorized users. In the distributed environment, different data providers have their access rules linked to their data items, and trusted data integration services may fetch data from different providers and send clients responding documents compliant to their own integration schema. The data mediator has to ensure exposure of sensitive information only to classes of users who satisfy the security characteristics and qualifications set by the data providers.

In an access control system we developed for Nature-

Serve [2], we filter and integrate the returns of distributed services of biodiversity data served by the 75 partners of the NatureServe Natural Heritage Network members. Each Natural Heritage partner provides data conforming to two specific schemas: one describes the attributes of species and the other provides attributes of biodiversity population surveys of species and natural communities and ecosystems-in set locations according to set ecological monitoring protocols designed by NatureServe and the partners. Data providers also create data access policies to hide sensitive information from unprivileged users. Most of these are concerned with protecting the exact location of rare or endangered species, but some also protect the privacy of land owners who have given permission for these ongoing surveys on their property. A trusted data integration service retrieves data from those providers and repackages it according to security policies. In [12], we describe a schema-driven security model to generate a security filter dynamically and accommodate to fast changing data integration scenarios. Data providers use a friendly application that generates security policies represented in the eXtensible Access Control Markup Language (XACML) [3]. Along with the integration schemas, these are used to dynamically generate an XSLT [6] filter to perform structure transformation required to deliver the response document satisfying all security constraints. Generally, the Natural Heritage providers hold data about species and ecosystems located in a particular state, Canadian province, or one of a few Latin American countries.

Although XACML maintains a concise syntax and is semantically rich enough to create data access policies, used alone it has difficulties handling complex multiple policy combinations and policy conflicts due to limited expressive power. In addition, features such as support for data access delegation control in a portable way – which will could assist applications of regional collaborations with shared access to sensitive information – is not provided in the current standard. We extend the policy combination mechanism and the XACML decision making process in a declar-

ative way with more expressive power to support various adaptive distributed service integration. Automatic inference from knowledge about policy issuers and data integration requirements along with rule effects will benefit reaching proper conclusions for resource access. In this paper, we describe a Horn Logic derived rule language extension to XACML to allow flexible policy combination and policy conflict resolutions. We show priority between rules can be inferred based on requirements of integrated data service and facts in policies introduced by different policy issuers by using rule inference. Rules can be reused to relate different services and can be used to create data access delegations for policy level administration. To achieve the interoperability in distribute environment, structure transformation program is used to convert the rule syntax in XML encoding compliant to RuleML [13] syntax.

The remainder of this paper is organized as follows. In Section 2 we review XACML model. Our rule based extension to XACML security model with examples is described in Section 3. In Section 4 we discuss the possible improvement to our engine by introducing more expressive logic dialect. Section 5 discusses related works for security policy management and concludes the paper.

2. Background

With policy management becoming popular means of providing flexible web security, a standard policy language protocol is an important requirement to integrate heterogeneous data with different access controls.. In this section, we briefly introduce the XACML architecture and describe some distribute policy conflicts which can not be handled well in its current specification.

2.1. The eXtensible Access Control Markup Language

XACML is an OASIS [3] standard which describes a general-purpose policy language and a protocol that defines the syntax of XACML compliant request and response. The policy language is used to describe general access permissions over information classified in a syntax described in the XACML core schema [4]. That schema defines sets of standard datatypes, functions and combination algorithms, and also supports extension mechanisms to allow user to integrate customized security control operations. Figure 1, illustrates the common XACML model in which some service wants to authorize an action on a resource. The original request is sent to a module called a Policy Enforcement Point (PEP). The PEP will fetch all the related information (the role or identity of the requestor, the resource, the action, etc.) and create a XACML request message and forward it to a Policy Decision Point (PDP). Once the PDP receives

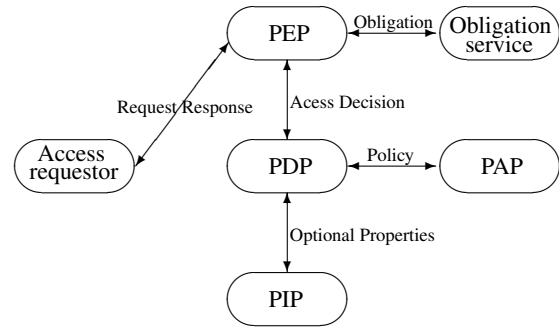


Figure 1. XACML request processing model

the request, it will retrieve respective policies and optional environment attributes from a Policy Administrative Point (PAP) and a Policy Information Point to make a decision on what kind of access may be granted with optional obligations to be imposed on the PEP. This answer will be sent back to the PEP, which will enforce the decision and provide the obligatory actions. We give a simple example below.

In a web service environment, every data provider can define its own rules to control access to its data. Using a standard language, such as XACML, insures compatibility and interoperability during service aggregation. A XACML policy can be abstracted as a quintuple $\langle \text{subject, object, action, decision, obligation} \rangle$. All the member items are defined by resources of different types and can be extended to fulfill specific application needs. The core modules of XACML model, the PDP and the PEP can also be services that are accessed in a distributed way. Different combinations (central PDP/distributed PEPs, distributed PDPs and PEPs etc) can satisfy those complicated scenarios in a distributed computing environment and can make the service aggregation easier and more seamless. Furthermore, policies about a single resource class can be distributed over the network and different groups can manage the sub-piece of the policies by themselves. For example different state agencies may have variant access definitions for a protected species; hence they may monitor their own policy sets as needed. The PDP will refer all those sub-pieces during the query time and make the right decision for PEP to enforce. The extension mechanism is a powerful feature to allow the security system to support usage more generic than this specific case.

2.2. Policy Conflict

The semantics of policy combination can be considered as depth first expression tree processing. Leaves of the tree are different rules or policies. Intermediate nodes are combination operators and decide the intermediate result of combining all the child effects. In a bottom up fashion, a final decision will be reached upon a particular query. XACML includes some combination operators such as “first permit” or “deny override”, however, the standard operators are not semantically rich enough to make complicated decisions in our data integration and service composition scenarios, let alone more complex ones. For example, suppose a particular user of the Natural Heritage data, A, has two assigned roles given by membership in two groups, “Scientific Researchers” and “Massachusetts Observers”. Authenticated membership in the former group might entitle the user to all precise latitude and longitude data of all species data throughout the network, while in the latter only to such data in Massachusetts. During data integration a PEP may issue a query to a PDP and ask whether user A has the right to access precise location of species in California, designated as resource R. Two relevant policies applicable to those roles and permissions are fetched from the PAP as depicted below:

Example 1:

```
Policy1
{‘researchergroup’, ‘resourceR’, ‘read’, ‘Permit’, ‘copy’}
Policy2
{‘observationgroup’, ‘resourceR’, ‘read’, ‘Deny’, ‘eliminate’}
```

Here “copy” and “eliminate” represent additional obligations of the system. While “copy” is perhaps superfluous, the “eliminate” obligation represents one of at least two ways one could contemplate acting on a “Deny” decision: abort the query or scrub the offending data while still passing the rest. The latter is clearly preferable, and a requirement imposed on our application. In this kind of scenario, security requirements usually will assign different priority to roles, and the combination algorithm will pick the one with higher priority value. Simply imposing an order quickly leads to a number of problems:

1. Policy issuers at one provider may be unaware of policies at other providers, each of which may maintain different Policy Administration Points as services in a distributed environment
2. Modification of a policy could impact all the policies of higher priority in the global order.
3. The policy order itself might depend on the applications that collaborating data providers might deploy. For example, for some applications it might be reasonable to declare that the authorization is to be against

the weakest policy in the set, but in some other against the strongest.

To effectively deal with policy conflicts while making access decisions in a PDP, a systematic extension to the XACML combination algorithm is needed so that we need not write combinatory code for every different request.

3. Rule based Extension for XACML

In this section we define basic syntax of our rule language and use examples to demonstrate using rule extension to solve general policy conflicts at the policy combination step in order to help a PDP to make an access query decision. Specified rulebases dealing with role priorities, obligation differences, forward delegation chains and external computation attachment are discussed. An XML based rule encoding compliant with the RuleML schema is produced using a simple XML model

3.1. Basic Horn Rules

To use declarative clauses to describe service requirements and metadata information, we use a language with simple syntax derived from Horn Logic [18] with sufficient expressive powers for handling XACML policy management. We reason over this with a light-weight engine based on the efficient SLD-resolution (Linear resolution with Selection function for Definite clauses) [7] inference algorithm, purpose-built for this language The BNF definition of the rules and facts in our rule language are provided by a subset of those permitted by Horn Logic and which is follows:

```
< clause > ::= < head > | < body > → < head >.
< head > ::= < atom >
< body > ::= < atomlist >
< atomlist > ::= < atom > | < atomlist >, < atom >
< atom > ::= < predicate > (< termlist >)
< termlist > ::= < term > | < termlist >, < term >
< term > ::= < constant > | < variable >
< predicate > ::= < constant >
```

Principally this language restricts Horn Logic by its absence of functions.

3.2. Example

A service mediator can create a rule set to solve the policy conflict between different roles such as is described in Section 2.2. consider the rules:

```
Superior(?x, ?y), Effect(?x, ?z) → Result(?z).
RoleHasHigherPriority(?r1 ?r2), PolicyAppliesTo(?x, ?r1),
PolicyAppliesTo(?y, ?r2) → Superior(?x, ?y)
RoleHasHigherPriority(“researchergroup”, “observationgroup”)
```

The first rule says that whenever policy x has higher priority than policy y, policy x's access decision is the final result of combination of two policies. The second rule indicates that whenever role r1 has higher priority than role r2, all those policies which apply to role r1 have higher priority than those which apply to role r2. The third rule is a trivial rule which doesn't have a body, and it reflects the fact for this particular service that the researcher group has higher priority than the observation group. During the PDP processing step described in Section 2.2, the PDP fetches all relevant policies and above rule set from the PAP and PIP respectively, then presents them to our XACML rule extension engine. Based on retrieved policies, more facts are generated about the constants arriving in policies retrieved from the PAP, such as the examples below:

```
Policy("policy1"), Role("policy1", "researchergroup"),
Effect("policy1", "Permit"),
Policy("policy2"), Role("policy2", "observationgroup"),
Effect("policy2", "Deny")
```

At the combination step, the extension engine will execute a query "*Superior(?x, ?y), Effect(?x, ?z) →*" to get the final combination result using a typical SLD unification algorithm, and the result("Permit") will be generated and sent back to the PDP.

3.3. Dealing with Enforcement Conflict

Policies are defined by different providers in the network, and there may exist disagreement between their approaches to protect sensitive information. In Figure 2, we show the expected result of different policy enforcement issued from different providers to protect sensitive geographic coordinates of species occurrence. Those approaches are elimination, encryption or substitution. Example 2 shows multiple policies with different obligations applied to a GIS polygon containing the sensitive coordinates:

Example 2:
 Policy1
 {'publicuser', 'occurrencePolygon', 'read', 'Deny', 'encryption'}
 Policy2
 {'publicuser', 'occurrencePolygon', 'read', 'Deny', 'substitution'}

The decision passed to a PEP for enhancement will be unpredictable since XACML doesn't specify a canonical way to handle this kind of conflict when in general the access mode for both policies is the same ("Deny").

Enforcement of obligations is similar to triggers in a relational database; when a query against a certain resource is processed, an associated transformation has to be performed. With the help of our XACML rule extension engine, the trusted data service can define a new reaction rule for policy selection based on issuer and resource. Thus, instead of using a role priority relationship to decide selection

< State code = 'NM' > < Countyname = 'RioArriba' zipcode = '35039' /> < Occurrencepolygon id='p23' latitude='3581N' longitude='10661W' /> < /State >
< State code = 'NM' > < Countyname = 'RioArriba' zipcode = '35039' /> < /State >
< State code = 'NM' > < Countyname = 'RioArriba' zipcode = '35039' /> < Occurrencepolygon id='p24' latitude='oAf32t' longitude='Ct47erf' /> < /State >
< State code = 'NM' > < Countyname = 'RioArriba' zipcode = '35039' /> < Countyname = 'Sandoval' zipcode = '35043' /> < /State >

Figure 2. Different access manipulation with obligations

order between policies, we specify the rule as:

```
IssuerPriority(?i1, ?i2), Role(?x, ?i1), Role(?y, ?i2)
→ Superior(?x, ?y)
LocalResource(?i1, ?r), DelegateResource(?i2, ?r)
→ IssuerPriority(?i1, ?i2)
```

Different declarative rules can be acquired from the PIP at query time, and dynamically integrated during the policy combination step, and consistent decisions will always be made

3.4. Administrative Policy Delegation

In order to provide a flexible security policy management, we need provision by which regional collaborations with shared access to sensitive information can delegate trust to one another in limited, possibly changing with time, circumstances. The draft standard for XACML administrative policy [5] provides a mechanism for this which we find cumbersome and potentially unscalable in the face of long trust chains or high service latencies in a distributed environment. Instead, we will describe how our rule language can reduce the trust establishment to a single query. In this discussion we refer to these three policies:

```
Policy1
{'researcher', 'ResourceR', 'read', 'DelegatetoIssuerC',
'inheritobligations', 'issuerA'}
Policy2
{'researcher', 'ResourceR', 'read', 'Permit', 'copy', 'issuerB'}
Policy3
{'researcher', 'ResourceR', 'read', 'DelegatetoIssuerB',
'inheritobligations', 'issuerC'}
```

Consider a scenario in which a PDP acquires Policy 2 from issuer B while acting upon a request to read a resource, "Resource R". This policy, issued by issuer B grants read permission on the resource. Now assume in this scenario that the PDP does not trust B to issue such policies on this resource. In the XACML draft procedure, an administrative request is constructed asking whether anyone delegates the right to issuer B directly or indirectly. This administrative query will get the result that issuer C will allow issuer B to make decisions on its behalf returned as Policy3, issued by

issuer C. But now imagine that C is also not trusted by the original PDP. This procedure must be iterated until the PDP acquires a policy from a trusted issuer asserting the delegation. We illustrate this with Policy 1, assuming issuer A is trusted. In the XACML draft protocol, this makes the original Policy 2 become, for the current query, suitable for enforcement. The current XACML standard ([3], [4]) makes no distinction between data access policies and trust delegation policies and in particular has no combination rules that allow trust chains to be resolved by a correctly functioning XACML implementation. The draft XACML 3.0 administrative policy standard [5] makes such a distinction, but does not provide for reading the delegation policies en masse. To avoid the service overhead of a chain of unknown length of these administrative requests, our rule based extensions can simply use declarative rules to make the delegation verification in one query. Delegation processing can be generalized to two rules:

```
TrustIssuer(?issuer), Policy(?pid, ?issuer),
    ?Effect(?pid, ?decision) → Result(?decision)
TrustIssuer(?x), Delegate(?x, ?y) → TrustIssuer(?y)
```

These rules guarantee that only policies issued by trusted issuers or issuers delegated by trusted issuers will be honored. At the policy combination step, facts such as *Delegate*("issuer A", "issuer C"), *Delegate*("issuer C", "issuer B"), *TrustIssuer*("issuer A"), *Policy*("policy 2", "issuer B") and *Effect*("policy 2", "Permit") will be generated and the result will be deduced in one run instead of multiple delegation queries. In practice, an application must make a trade-off about making a single query retrieving all policies about a given role, resource and permission, compared to chasing delegation chains. If there are many delegations involving no path between a trusted policy issuer and the target one, an application would suffer the consequences of retrieving many policies that don't participate in the reasoning. One could imagine a low-cost metadata registry which carries information that would allow whatever drives the reasoner to choose a strategy: chase paths or get everything. Indeed, the current implementation of the Natural Heritage access control system has no requirement for trust delegation, though some interest has been expressed. In reality, we believe that the delegation networks would be rather small, generally driven by collaborations between providers whose geographic boundaries adjoin one another, e.g. the providers from adjacent states.

3.5. Procedure Attachment with Rules

In order to deduce the proper result for a query, the rule engine has to generate a set of corresponding atoms automatically based on policy context. It would be desirable to simply assert policies as axioms, but sometimes policy con-

flicts are context sensitive and data dependent, and external computation is needed. Suppose we have a policy that specifies that the observation group user is entitled to access only the name of the county containing an observation, not the detailed polygon which may be in the data. It may be desirable to have a delegation rule which allows a user to access the polygon level data if the area of the specified polygon is bigger than any county size, otherwise a deny is issued. External computation is required to compute the size of the polygon and generate proper axioms for later use. This procedure attachment information has to be saved together with the rulebase and be invoked upon generating relevant facts.

3.6. Compliant RuleML syntax

In order to exchange rules freely in a distributed network environment, an XML encoding of rules and facts is preferred. We use RuleML compliant syntax [13] and the rule *Superior*(?x, ?y), *Effect*(?x, ?z) → *Result*(?z) in example 1 can be coded as below:

```
< Implies >
  < head >
    < Atom >
      < Rel > Result < /Rel >
      < Var > z < /Var >
    < /Atom >
  < /head >
  < body >
    < And >
      < Atom >
        < Rel > Superior < /Rel >
        < Var > x < /Var > z < Var > y < /Var >
      < /Atom >
      < Atom >
        < Rel > Effect < /Rel >
        < Var > x < /Var > < Var > z < /Var >
      < /Atom >
    < /And >
  < /body >
< /Implies >
```

4. Future XACML rule extension enhancements

4.1. Defeasible Logic Extension vs. Fuzzy Logic Extension

In some policy combination scenarios, data mediation may demand many considerations in order to determine the priority order between policies. For example, relative importance of user roles, credentials of a policy issuer and attributes of resources may all need to be considered. To deal with such complexity, a Defeasible Logic [16] extension of original rule language can be used, namely adding a new production: Rule ::= clause, rank. With this, inferred results can be screened based on rule ranks and the highest

ranking result will be honored and treated as the result of decision tree.

Although ranking mechanisms provide a simple and efficient solution for reconciling multiple rule selection, they have some drawbacks. Ordering rule ranking is not always appropriate because policies are fetched at run time. Furthermore, the static coupling of the rank with any implication rule make the rule sets not suitable for reuse among different queries. Alternatively, in order to model the varying nature of rule ordering among various queries and possible imprecision, a weight can be assigned dynamically to generate fuzzy rules [19] like $IssuerPriority(?i1, ?i2), Role(?x, ?i1), Role(?y, ?i2) \rightarrow superior(?x, ?y) * 0.9$. A fuzzy rule XACML extension should be a better way of dealing with uncertainties and vague or incomplete information.

4.2. Using object oriented rule with ontology

Harold [9] has proposed a frame logic language [15] called Object Oriented RuleML. It provides basic constructs to allow object centric knowledge representation with named attributes instead of a sequence of propositional arguments. Moreover, OO RuleML defines order-sorted terms, which raise an integrity constraint on variables, individual constants and complex terms to assure that rules are only applied on the correct type of objects.

By using OO rules, we can actually link atoms into taxonomies such as RDF class hierarchies and lightweight OWL ontologies [14]. For example, we may have a hierarchical role structure maintained among all Heritage Network partners, and rules can be created based on the ontological information between different classes. Thus, a role “NationalScientist” might be a role for which all providers agree that a NationalScientist has all the privileges of the several “Scientist” roles assigned by each provider. In future implementations, during inference general rules will be replaced based on proper class subsumption and verified instance goals will be generated during tree building. Using OO rules with ontology will also simplify policies for different resources. Although the Natural Heritage network has no resource hierarchies, definitions about resource structure could be used to combine multiple special policies with a generalized policy and also allow permission set on a parent resource expression to be carried to child elements.

4.3. Negation as Failure (NAF)

In our rule language, we don’t have negative atoms. This makes our rule representation and query result generation very straightforward, at the cost of overhead due to limited expressive power. For example, suppose we have many

policies that can be applied to a resource access query. Based on our rules in Example 1, the rule extension engine can only process policy combination in a sequential manner, more explicitly it handles only two policies at one time. So N policies will need N-1 priority comparisons and also require the recording of intermediate states and generation of proper policy facts at the right time. With Negation as Failure (NAF, [17]) support, we could replace the result rule as below:

$$Superior(?x, ?y), Effect(?x, ?z), \neg Superior(?x1, ?x) \rightarrow Result(?z)$$

As the rule indicates that only the result from the highest priority policy will be honored as the result of the decision tree, all policy facts will be generated all at once and only one inference run will be executed to get result.

5 Conclusion

Many research efforts have schemes for document centric access controls. Bertino et al. [8] have studied issues like access granularity and role hierarchies. Bonatti et al. [10] propose a model for security and privacy management in cooperative systems. However their model is more static than ours, and doesn’t consider service composition in a distributed environment. Denker and Martin [11] propose using web ontology to define semantics of privacy policies, however OWL DL itself is not able to represent multiple joins across different predicates and it is hard to generate implication rules based on conjunctions of premises. Compared to those works, we use standard XACML as de facto policy language and a rule based extension to address possible policy conflicts and facilitate access decisions. We demonstrated methods to deal with resource level, operation level and semantic level conflicts. In future work, we plan to add more expressive power to our current rule language and extend the rule usage in other XACML contexts, for example policy fetching based on ontology and policy delegation based on asserted rules. A new unification algorithm is also needed to reflect integration of object oriented rules with other features to properly reflect query inference processes.

6 Acknowledgment

This research was partially supported by NSF grant DBI 0345400 to NatureServe. We are also grateful to Douglas Sellars for helpful criticism of the underlying access control architecture which this work extends.

References

- [1] *NatureServe*. <http://www.natureserve.org/>.
- [2] *NatureServe Natural Heritage Program*. <http://www.natureserve.org/visitLocal/>.
- [3] *OASIS eXtensible Access Control Markup Language (XACML) TC, XACML V2.0*. http://www.oasisopen.org/committees/tc_home.php?wg_abbrev=xacml.
- [4] *OASIS XACML policy schema*. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-policy-schema-os.xsd.
- [5] *XACML v3.0 administrative policy*. http://www.oasis-open.org/committees/download.php/15764/access_control-xacml-3.0-administration-wd-10.zip.
- [6] *XSL Transformations (XSLT) Version 1.1;W3C Working Draft 24 August 2001*. <http://www.w3.org/TR/2001/WD-xslt-20010824/>, 1999.
- [7] N. Bassiliades, G. Antoniou, and I. Vlahavas. *DR-DEVICE: A Defeasible Logic System for the Semantic Web*. 2nd Workshop on Principles and Practice of Semantic Web 2004.
- [8] E. Beritino, S. Castano, E. Ferrai, and M. Mesiti. *Controlled Access and Dissemination of XML documents*. In Proc. 2nd ACM Workshop on Web Information and Data Management, 1999.
- [9] H. Boley, M. Dean, B. Groszof, M. Sintek, B. Spencer, S. Tabet, and G. Wagner. *FOL RuleML: The First-Order Logic Web Language*. <http://www.ruleml.org/fol/>.
- [10] P. Bonatti. Rule languages for security and privacy in cooperative systems. *Computer Software and Applications Conference*, 2005.
- [11] G. Denker and D. Martin. Rule languages for security and privacy in cooperative systems. *Computer Software and Applications Conference*, 2005.
- [12] H. Dong, Z. Wang, R. A. Morris, and D. Seller. *Schema-Driven Security Filter Generation For Distributed Data Integration*. <http://alligator.cs.umb.edu/xmlsecurity.pdf>.
- [13] D. Hirtle, H. Boley, B. Groszof, M. Kifer, M. Sintek, S. Tabet, and G. Wagne. *Schema Specification of RuleML*. <http://www.ruleml.org/0.9/>.
- [14] M. Hori, J. Euzenat, and P. F. Patel-Schneider. *OWL web ontology language XML presentation syntax. W3C Note, 11 June 2003*. <http://www.w3.org/TR/owl-xmlsyntax/>.
- [15] M. Kifer and G. Lausen. *F-logic: a higher-order language for reasoning about objects, inheritance, and scheme*. International Conference on Management of Data, 1989.
- [16] J. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.
- [17] U. Nilsson and J. Maluszynski. *LOGIC, PROGRAMMING AND PROLOG*. 2000.
- [18] L. Sterling and E. Shapiro. *The Art of Prolog*. , MIT Press, 1994.
- [19] Y. Wang, H. Deng, and Z. Chen. *Adaptive fuzzy logic controller with rule-based changeable universe of discourse for a nonlinear MIMO system*. Intelligent Systems Design and Applications, 2005. ISDA'05.