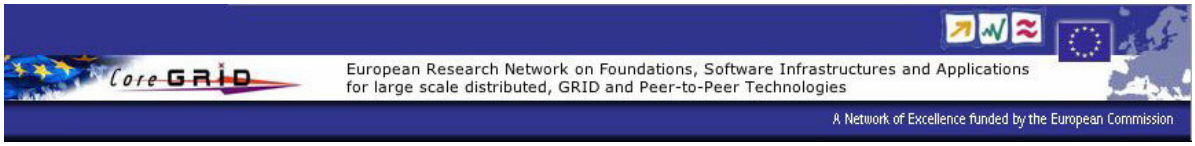

A Study of Languages for the Specification of Grid Security Policies

Syed Naqvi, Philippe Massonet
Centre d'Excellence en Technologies de l'Information et de la Communication
CETIC, 8 Rue Clément Ader, B-6041 Charleroi, Belgium
{syed.naqvi, philippe.massonet}@cetic.be

Alvaro Arenas
CCLRC Rutherford Appleton Laboratory
Chilton, Didcot, OX11 0QX, United Kingdom
a.e.arenas@rl.ac.uk

CoreGRID Technical Report
Number TR-0037
April 14, 2006

Institute on Knowledge and Data Management

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme

Project no. FP6-00426

A Study of Languages for the Specification of Grid Security Policies

Syed Naqvi, Philippe Massonet
Centre d'Excellence en Technologies de l'Information et de la Communication (CETIC)
8 Rue Clément Ader, B-6041 Charleroi, Belgium
{syed.naqvi, philippe.massonet}@cetic.be

Alvaro Arenas
CCLRC Rutherford Appleton Laboratory
Chilton, Didcot, OX11 0QX, United Kingdom
a.e.arenas@rl.ac.uk

CoreGRID TR-0037
April 14, 2006

Abstract

This report presents a study of various semantic languages. Our objective is to identify the most suitable language for the specification of grid security policies. A policy-driven approach to security requirements of grid data management systems (GDMS) heavily relies on the choice of some appropriate language for the expression of the policies. In this report, we have first identified a number of semantic languages that can be employed for the specification of grid security policies followed by a comparative analysis of these languages. We have used several criteria for the comparison of these languages – such as their suitability to specify grid security policies, their expressiveness, their representation techniques, their reasoning strength, and their deployment characteristics.

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

1. Introduction

The size and the complexity of real world security policies make it impossible to handle *flat* policy representations such as plain authorization lists [1]. Policy complexity is increased by the interplay of multiple, heterogeneous requirements, that must be grouped together into a coherent policy. The concept of grid computing – *to enable coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations* [2] – poses a number of new challenges for the specification and consequent implementation of the security policy. The list of these challenges includes:

- **Resource sharing:** Resources being used are still owned by their respective organization and subject to its policies.
- **Coordination:** Resources contributed to VO (virtual organization) need to have a coherent policy in order to interoperate.
- **Dynamicity:** Users and resources may be large, unpredictable, having distinct roles, and changing at any point. This situation does not allow static configuration of the policy.
- **Multi-institutional issues:** Resources and users have respective local policies and technologies that cannot be replaced by the VO. Moreover, cross-organizational trust relationships may not already exist.

Our main aim is to derive security policies from the security requirements in grid systems. To facilitate this objective, we have analysed a number of security specification languages to identify the most suitable one that can be used to express the security policies of grid systems. We first selected a number of known security specification languages. Then we short listed these languages according to their suitability to specify security, representation techniques, abstraction levels etc. These short listed policy specification languages are elaborated followed by a detailed comparative analysis. Finally some conclusions of this study are drawn at the end of this report.

2. Policy Specification Languages

A policy is a statement of the intent of the owner or controller of some computing resources, specifying how he wants them to be used [3]. Policy specification languages are an attempt to formalize the intent of the owner into a form that can be read and interpreted by machines [4]. Thus a policy may give certain rights to entities (programs, users, communications, et cetera) that fulfil some criteria and deny other rights. Each language must define the entities and their attributes it considers as well as the actions or permissions that can be given. A single policy is then some binding of entities and their attributes to specified actions. What entities, attributes, actions and combinations thereof can be represented depends on the language used.

For the study of current policy specification languages, we have considered the following languages at the first instant:

ASL (Authorisation Specification Language) [5] is a formal logic language for specifying access control policies. The language includes a form of meta-policies called integrity rules to specify application-dependent rules that limit the range of acceptable access control policies. Although it provides support for role-based access control, the language does not scale well to large systems because there is no way of grouping rules into structures for reusability. A separate rule must be specified for each action. There is no explicit specification of delegation and no way of specifying authorisation rules for groups of objects that are not related by type.

ISPS (IPsec Security Policy Specification) [6] apply policy constraints on entities (such as security gateways and router filters) along the path of a communication. Current IP security protocols and algorithms can exchange keying material using IKE and protect data flows using the AH and/or ESP protocols. The scope of IKE limits the protocol to the authenticated exchange of keying material and associated policy information between the end-points of a security association. ISPS specifies confidentiality and integrity rules; however, it lacks some important features such as authentication, audit, delegation etc.

KAoS [7] is a collection of services and tools that allow for the specification, management, conflict resolution, and enforcement of policies. KAoS uses ontology concepts encoded in OWL [8] to build policies. The KAoS Policy Service distinguishes between authorization policies and obligation policies. The applicability of the policy is defined by a set of conditions or situations whose definition can contain components specifying required history, state and currently undertaken action. In the case of the obligation policy the obligated action can be annotated with different constraints restricting possibilities of its fulfilment.

LaSCO (Language for Security Constraints on Objects) [9] attempts to express constraints on objects. LaSCO policies are specified as logical expressions and as directed graphs. The auditing operations and control aggregation problems are indirectly expressed by LaSCO. However, there is no direct way of specifying confidentiality and integrity in LaSCO. From the grid's point of view, the delegation is not supported by the LaSCO syntaxes.

PCIM (Policy Core Information Model) [10] is an object-oriented information model for distributed policy information. This approach to policy specification treats the policy-based system as a state machine in which the policies determine the manner in which state transitions occur. The PCIM specification provides an abstract model for defining the structure of policies and relationships between policy objects. However, as PCIM is only an information model, there is no built-in support for policy analysis or refinement.

PDL (Policy Description Language) [11] is an event-based language originating at the network computing research department of Bell-Labs. Policies in PDL use the event-condition-action rule paradigm of active databases to define a policy as a function that maps a series of events into a set of actions. The language has clearly defined semantics and an architecture has been specified for enforcing PDL policies. The language can be described as a real-time specialised production rule system to define policies. PDL does not support access control policies, nor does it support the composition of policy rules into roles, or other grouping structures.

PMAC (Policy Management for Autonomic Computing) [12] is a framework that uses policy-based management to simplify the management and automation of products and complex systems. Policies in the PMAC framework are defined by using XML schema. The framework includes a general constraint specification language which is also specified using XML. PMAC provides support for policy analysis, refinement operations, and conflict resolution. However, the analysis process does not take into account the behaviour of the managed system and therefore cannot detect inconsistencies. These inconsistencies are surfaced when the enforcement of one policy causes a state change that yields conflicts among the other policies.

Ponder [13] is a declarative, object-oriented language developed for specifying management and security policies. Ponder permits to express authorizations, obligations, information filtering,

refrain policies, and delegation policies. Ponder can describe any rule to constrain the behaviour of components, in a simple and declarative way.

PPL (Path-based Policy Language) [14] is designed to support both the differentiated as well as the integrated services model and is based on the idea of providing better control over the traffic in a network by constraining the path (i.e. the links) the traffic must take. However, except access control expressions, PPL does not adequately cover the other security functionalities.

Rei [15] is a policy framework that integrates support for policy specification, analysis and reasoning. It allows users to express and represent the concepts of rights, prohibitions, obligations, and dispensations. In addition, Rei permits users to specify policies that are defined as rules associating an entity of a managed domain with its set of rights, prohibitions, obligations, and dispensations. Rei provides a policy specification language in OWL-Lite that allows users to develop declarative policies over domain specific ontologies.

Table 1 provides a coarse-grained comparison of these policy specification languages. This table is constructed to short list the number of policy languages for their elaborate presentation in the section 3.

Languages	ASL	ISPS	KAoS	LaSCO	PCIM	PDL	PMAC	Ponder	PPL	Rei
Access control	X	X	X	X	X	X		X	X	X
Identification	X		X	X			X	X		
Confidentiality		X								X
Integrity		X								X
Audit				X				X		
Delegation			X					X		X
Constraints	X		X				X	X		X
Abstraction	X	X	X	X	X	X			X	
Semantics	X		X							X
Reasoning			X					X		
Deployment			X				X	X		
Editing tools			X				X			

Table 1: Coarse-grained comparison of the policy languages

It is evident from the table 1 that KAoS, Ponder and Rei are the languages that support a good number of the features required for the grid security policy specification.

3. Description of the most Relevant Policy Specification Languages

3.1. KAoS

KAoS policy specification language is a set of platform-independent services that let people define policies ensuring adequate predictability and controllability of both agents and traditional distributed systems. KAoS services and tools allow for the specification, management, conflict resolution, and enforcement of policies in the specific contexts established by complex organizational structures represented as domains. Policies in KAoS are represented in a semantic web language called OWL (Web Ontology Language). It employs OWL-encoded ontology concepts to build policies. The core of the KAoS Policy Ontology defines concepts used to describe a generic actor's environment and policies in this context. The core is loaded at the KAoS bootstrap phase. It then loads additional ontologies on top of this core layer that extend concepts from the core ontology with notions specific to the particular controlled environment and application domain.

The KAoS Policy Service constraints that permit or forbid some action are called positive and negative *authorizations.* Likewise, constraints that require some action when a state- or event-based trigger occurs or that serve to waive such a requirement are called positive and negative *obligations*. Other policy constructs (for example, delegation or role-based authorization) are built from the basic domain primitives plus the four policy types:

PositiveAuthorization,
NegativeAuthorization,
PositiveObligation,
NegativeObligation.

The type of policy instance determines the kind of constraint KAoS should apply to the action, while a policy's action class is used to determine a policy's applicability in a given situation. OWL restrictions are used in the action class to narrow down scopes-of-action properties to the needs of a particular policy. Every action contains a definition of the range of actors performing it. Policies in KAoS are represented without conditional rules. They instead rely on the context restrictions associated with the action class to determine policy applicability in a given situation. Action instances – that actors intend to take or are undertaking – are classified by an action class. It then obtains a list of any policies whose action classes are relevant to the current situation. Now, KAoS determines the relative precedence of the obtained policies and sorts them accordingly to find the dominating authorization policy. For positive dominating authorization, KAoS collects obligations from any triggered obligation policies in order of precedence. Finally, KAoS returns the result to the interested parties that are generally the enforcement mechanisms. These mechanisms are collectively responsible for blocking 'not allowed' actions and for assuring the performance of obligations.

KAoS facilitates reasoning about the controlled environment (domain structure and concepts exploiting the description logic subsumption and instance classification algorithms); policy relations and disclosure; policy conflict detection and harmonization through underlined algorithms. Other features of KAoS include homogeneous policy representation; support for both authorization and obligation policies; use of services in conjunction with a wide range of applications and operating platforms; and ease of loading platform-specific and application-specific ontology on top of the core concepts.

3.2. Ponder

Ponder is a declarative and object-oriented language for specifying security and management policies for distributed systems. Besides, policy specification, Ponder can group policies into roles and relationships; and then define configurations of roles and relationships as management structures. Ponder implementation is not platform specific; rather it can map to and co-exist with one or more underlying platforms. Ponder supports an extensible range of policy types such as: authorisation policies; obligation policies; refrain policies; delegation policies; composite policies; constraints; meta-policies; etc.

Generally, a security policy is dubbed as an authorisation policy which is related to access-control and specifies what activities a subject is allowed / disallowed to perform. Authorization policies are designed to protect target objects from the malicious entities. Obligation policies specify responsibility of a subject by defining its duties – i.e. what activities the subject should perform to a set of target objects. Refrain policies are the converse set of rules – i.e. they specify what a subject must refrain from doing. Delegation policies specify the actions which subjects are allowed to delegate to others. Therefore, a delegation policy specifies an authorisation to delegate. Composite policies are used to harness a set of related policy specifications within a syntactic scope with shared declarations in order to simplify the policy specification task for large distributed systems. There are four types of composite policies: groups, roles, relationships and management structures. Constraints are specified in order to limit the applicability of policies. They are based on objects' parameters (such as time or attributes' values) to which the policy refers. Meta-policies are the policies of policies. They are policies about which policies can coexist in the system or what are permitted attribute values for a valid policy. Policies specified in Ponder depend on the key concept of management domains. These domains provide a mechanism for grouping objects to which policies apply and can be used to partition the objects in a large system as preferred. Consequently Ponder policy subjects and targets are defined in terms of domain scope expressions that enable the grouping of domains to form composite set of objects.

Like any other object-oriented programming language, Ponder syntax is compiled by its compiler at the end of the policy specification phase. The Ponder compiler compiles the policy into a Java class and then represented at runtime by a Java object. However, policy can not be changed during the runtime. Although the specification of the interfaces for the enforcement agents is provided by Ponder, but no implementation is yet provided. Some examples of Ponder policy deployment model in different application domains are given in [17].

3.3. Rei

Based in OWL-Lite, Rei also includes logic-like variables that provide flexibility to specify relations such as role value maps that are not directly possible in OWL. Rei allows policies to be specified as constraints over allowable and obligated actions on resources in the environment. Meta-policy specifications are used for conflict resolution, speech acts for remote policy management and policy analysis specifications. Rei is composed of both domain dependent ontologies and domain independent ontologies. Delegation management is supported by the Rei's policy engine that makes it useful for dynamic systems consisting of transient resources and users, and distributed systems. Rei supports two kinds of delegation by providing a standard way of controlling and propagating access rights through delegation. It allows individual policies as well as group and role based policies to be specified.

Rei provides a way to specify actions and conditions by assuming that the meanings of actions or conditions are domain dependent and that their complete processing is outside the policy. Therefore, by using these actions and conditions, a policy maker can create policy objects. There

are four kinds of policy objects: rights, obligations, prohibitions, and dispensations. Rei models four speech acts that can be used within the system to modify policies dynamically: delegate, revoke, cancel and request. The Rei policy language contains meta-policy specifications for conflict resolution. It has a policy engine that is used to interpret and reason over the policies and speech acts so that decisions about users' rights and obligations can be made. Rei policy engine is queried for each request of a certain action by an agent. It checks if the requester has the right, by checking its policy objects, or if the right has been delegated to the requester by another entity with the right to delegate. It also checks for prohibitions and revocations. The meta-policy associated with the agent and the requested action is used to resolve conflicts if any. If the policy decision yields positive response, Rei informs the owner of the action and allow the owner to interpret the action and handle its execution.

Policy specification in Rei is flexible and is easy-to-use policy language. It includes few constructs, based on deontic concepts that are powerful because they can be used to describe several kinds of policies including security policies. For example, security policies restrict access to certain resources in an organization. Rei can be used to create actions on the resources and to describe role based rights and prohibitions for the users in the organization. Conversely, management policies define the role of an individual in terms of his duties and rights. Rei maps this directly into obligations and rights. Likewise, other policies can be described in the same way in terms of deontic principles that makes Rei versatile.

4. Comparative Analysis of KAoS, Ponder and Rei

In this section we provide a comparison of KAoS, Rei and Ponder according to policy representation (abstraction level), policy reasoning (analysis and verification), and policy deployment. However, we first specify a simple natural language policy in all of these three languages, so that the difference of policy specification becomes clear before presenting their comparison.

4.1. Example Policy

The following natural language policy is considered in this section:

> *Permit the access to the files if the user is in the group of employees authorized for this access.*

This policy is now expressed by KAoS, Ponder and Rei languages:

KAoS

```
<owl:Class rdf:ID="FileAuthAction">
<owl:intersectionOf rdf:parseType="owl:collection">
  <owl:Class rdf:about="&action;AccessAction"/>
  <owl:Restriction>
  <owl:onProperty rdf:resource="&action;#performedBy"/>
  <owl:toClass
    rdf:resource="&domains;MembersOfAuthorizedEmployee"/>
  </owl:Restriction>
  <owl:Restriction>
  <owl:onProperty rdf:resource="&action;#performedOn"/>
  <owl:toClass
```

```
      rdf:resource="&domains;MembersOfFilesServer"/>
   </owl:Restriction>
</owl:intersectionOf>
</owl:Class>
<policy:PosAuthorizationPolicy rdf:ID="AccessAuthPolicy1">
   <policy:controls rdf:ID="AccessAuthAction"/>
   <policy:hasSiteOfEnforcement rdf:resource="#FileStorageSite"/>
   <policy:hasPriority>1</policy:hasPriority>
</policy:PosAuthorizationPolicy>
```

Ponder

```
type auth+ FileAccess { subject Employee,
                        target CompanyFiles}
   {
      action read, write;
      When
           IsAuthorizedEmployee(TRUE);
   }
```

Rei

```
<constraint:SimpleConstraint rdf:ID="IsAuthorizedEmployee"
   constraint:subject="#RequesterVar"
   constraint:predicate="&example;memberOf"
   constraint:object="&example;AuthorizedEmployee"/>
<constraint:SimpleConstraint rdf:ID="IsFilesServer"
   constraint:subject="#FilesServerVar"
   constraint:predicate="&example;memberOf"
   constraint:object="&example;FilesServer"/>
<constraint:And rdf:ID="AreAuthorizedEmployeeAndFilesServer"
   constraint:first="#IsAuthorizedEmployee"
   constraint:second="#IsFilesServer"/>
<deontic:Permission rdf:ID="FilesServerPermission">
   <deontic:actor rdf:resource="#RequesterVar"/>
   <deontic:action rdf:resource="&example;access"/>
   <deontic:constraint
        rdf:resource="# AreAuthorizedEmployeeAndFilesServer"/>
</deontic:Permission>
<policy:Policy rdf:ID="AccessAuthPolicy1">
   <policy:grants rdf:resource="#FileServerPermission"/>
</policy:Policy>
```

4.2. Policy Representation:

Policy representation in KAoS can be made at any desired level of abstraction. This is a major advantage of using KAoS for policy representation. Policy elements such as groups, actors,

actions, action properties, conditions of applicability are described by corresponding ontology at the desired level of abstraction. This feature also facilitates the system to consistently reason about complex conflicts and be able to answer queries pertaining to the entities that can be defined with widely varying scopes and levels of description. These predefined ontologies can also be extended to accommodate specific requirements of an application.

Policy representation in Ponder requires the specific names of the entities to be controlled and their interfaces. Ponder users need to know these names beforehand. Use of Ponder in multi-agent systems has a potential limitation of policy definition: the low level of abstraction provided by method calls. In multi-agent systems, support for dynamic runtime changes and the need for simultaneously varying levels of description are by and large key requirements.

Policy representation in Rei exhibits the expressiveness of a logic language and the flexibility of an ontological description. This dual feature is a result of Rei's support of double policy specification in both the Prolog language (Rei constructs) and in the RDF-S language (Rei ontology). Double policy specification of Rei permits the combination of a simple and compact policy specification with an ontological description of the specification itself. However, there exist problems with this feature: Prolog-like syntax policy specification is not so easy for users with limited expertise in logic languages, even if it is more readable than a semantic web language.

4.3. Policy Reasoning:

Policy reasoning in KAoS greatly benefits from its declarative nature of policy definition that facilitates policy analysis and verification. Policy analysis and verification provide several features such as the detection of conflicting policies and enable their eventual harmonization; disclosure/publishing of related policies; reasoning about future actions based on knowledge of policies in force; etc. In general, KAoS can leverage the power and efficiency of description logics to their full extent.

Policy reasoning in Ponder is carried out by the rules which are translated into an event calculus representation that describes the semantics of the policy language. Ponder can therefore detect modality and application specific conflicts in policies, such as conflicts of duty. Ponder uses abductive reasoning techniques to analyse the policy specifications for the identification of existing conflicts and the corresponding explanations on how they might arise,

Policy reasoning in Rei relies on the Prolog reasoning engine. The Prolog reasoning engine analyzes the policies and can reason over domain-dependent ontologies. Unlike KAoS (where this conversion is hidden from the users), Rei users are empower to directly manage the intermediate form. Moreover, Rei users can access the policy instances in this (intermediate) form by submitting appropriate Prolog queries to the policy engine.

4.4. Policy Deployment:

Policy in KAoS is deployed in the abstract representation of actions that enables KAoS to be adapted for any platform that has the required *enforcement hooks*; however, such versatile deployment in any given platform requires the availability of a specific enforcer for the *policy action class*. Implementation of an enforcer (if such an enforcer does not already exist) is generally a difficult task as it requires its developers to have a thorough understanding of the action class and its mapping to the platform. With the development of libraries of enforcers for common actions across platforms, the implementation of enforcer is going to be less and less of an issue.

Policy in Ponder is deployed at a lower level of abstraction. Policies in Ponder can be directly implemented in Java with little additional effort. This feature significantly eases the overall policy deployment process and that's why Ponder is widely used in many existing systems.

There is no support of policy deployment in Rei. It is a major obstacle for exploiting this language for practical applications. Unlike KAoS that manages the policy in its original form, leaving to the implementing platforms the possibility to choose the most appropriate run-time representation, the Rei engine accepts ontology-based policy specification; yet the main policy management is over the Rei constructs.

4.5. Comparison Table:

The comparative analysis carried out in section 4 is summarized in the table 2:

Languages	KAoS	Ponder	Rei
Abstraction level	High	Medium	Low
Ontology-based	Yes	No	Yes
Specification language	OWL (Web Ontology Language)	Ponder (declarative specification)	Rei (Prolog-like syntax + RDF-S)
Policy specification tools	KPAT	Graphical editor and compiler	Nil
Reasoning support	Java Theorem Prover	Event calculus representation	Prolog engine
Enforcement mechanism	Need to write the code of appropriate enforcers and to insert them in entities to control	Java interfaces for enforcement agents are provided	Action execution is outside the Rei engine

Table 2: Summary of the policy languages comparison

5. Concluding Discussion

Selection of an appropriate policy specification language is a crucial design stage of the policy driven systems. Consideration of the grid-specific characteristics is essential in the context of policy-driven approach for the specification of grid security requirements. In this report, we have carried-out a comprehensive analysis of the policy languages which are most relevant to the grid security policies. Although, there is no language that fully comply with the grid characteristics; however, KAoS is found to be the most suitable language. The runner-up language of this analysis is Ponder, which although adequately responds to the needs of grid security; but the non-availability of any tool for the formulation of a policy in Ponder semantics makes it difficult to use. Moreover, Ponder is not a semantic language that makes the policy specification a bit more cumbersome. KAoS on the other hand is a semantic language supported by some editing tools such as KPAT (KAoS Policy Administration Tool) [18].

References

1. Bonatti P., *Rule-based Policy Specification: State of the Art and Future Work*, Public Report # IST506779/Naples/I2-D1/D/PU/b1 of the European Project REWERSE (Reasoning on the Web with Rules and Semantics), September 2004
2. Foster I., Kesselman C., Tuecke S., *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*, International J. Supercomputer Applications, 15(3), 2001.
3. Pfleeger C., Security in Computing, Second Edition, Prentice Hall PTR, 1997, pp 574
4. Kangasluoma M., Policy specification languages, Report of Helsinki University of Technology, Helsinki, Finland, 11 November 1999
5. Jajodia S. et al., *A Logical Language for Expressing Authorizations*, Proceedings of the IEEE Symposium on Security and Privacy 1997, pp 31-42
6. Fu Z. et al., *IPsec/VPN Security Policy: Correctness, Conflict Detection, and Resolution*, Proceedings of Policy 2001, Lecture Notes in Computer Science, Vol. 1995. Springer-Verlag 2001 pp 39-56
7. Johnson M., Chang P., Jeffers R., Bradshaw J., et al., *KAoS Semantic Policy and Domain Services: An Application of DAML to Web Services-Based Grid Architectures*, Proceedings of the AAMAS 03 workshop on Web Services and Agent-Based Engineering, Melbourne, Australia, July 2003
8. The Web Ontology Language (OWL) – http://www.w3.org/TR/owl-ref/
9. Hoagland J. et al., *Security Policy Specification Using a Graphical Approach*, Technical report CSE-98-3, University of California, Davis Department of Computer Science. 1998
10. Moore B., Ellesson E., Strassner J., and Westerinen A., *Policy Core Information Model – Version 1 Specifications*, IETF Network Working Group – RFC3060, 2001
11. Lobo J. et al., *A Policy Description Language*, Proceedings. of AAAI'99, Orlando, Florida, USA, 1999
12. IBM Autonomic Computing Initiative – http://www.alphaworks.ibm.com/tech/pmac
13. Damianou N., Dulay N., Lupu E., Sloman M., *Ponder: A Language for Specifying Security and Management Policies for Distributed Systems*, Imperial College, UK, Research Report Department of Computing 2001, (2000)
14. Stone G.N. et al., *Network Policy Languages: A Survey and a New Approach*, IEEE Network, 2001
15. Kagal L., *Rei: A Policy Language for the Me-Centric Project*, HP Labs Technical Report, HPL-2002-270, (2002)
16. Tonti G., Bradshaw J., Jeffers R., Montanari R., Suri N., and Uszok A., *Semantic web languages for policy representation and reasoning: A comparison of KAoS*, *Rei and Ponder*, Proceedings of the 2nd International Semantic Web Conference, Sanibel Island, Florida, USA, Oct. 2003.
17. Montanari R., Tonti G., Stefanelli C., *A Policy-based Mobile Agent Infrastructure*, Proceedings of IEEE Symposium on Applications and the Internet 2003 (SAINT 2003), Orlando, Florida, US, (2003), pp 370-379
18. Uszok A., Bradshaw J., Tonti G., Jeffers R., and Olson L., Integration of KAoS Policy Services with Semantic Web Services, International Semantic Web Conference 2004 (ISWC2004), 7-11 November 2004, Hiroshima, Japan