# Flexible and Usable Policies

P.A. Bonatti[*]

## 1 Introduction

The needs of the new business models—in particular their demanding dynamicity and reliability requirements—have been fostering the creation of new open, service-oriented software architectures. Here the word "reliability" includes security, privacy, and trust issues. Lack of trust in the new software architectures is perceived as a major risk, that may hinder large-scale adoption of the new paradigm.

According to the service oriented vision, systems are deployed by composing dynamically (at run time) or offline (at design time) sets of services. In this perspective, a major issue from the point of view of security, privacy, and trust is how the heterogeneous policies adopted by the component services can properly interoperate and be harmonized in a way that preserves the security, privacy and dependability requirements of all involved peers and organizations.

Further and almost extreme needs for dynamic decision making and policy enforcement come from pervasive computing scenarios. The continuous and transparent interaction of small computing appliances with a changing environments pose hard security and privacy preservation problems. Access rights and information disclosure depend on variables such as location, time, nonfunctional service properties (QoS, privacy policy, cost, etc.) and so on.

Services often provide high-level abstractions, directly mapped to business-level concepts. This level of granularity facilitates the deployment of new applications and the support of short-lived and task-oriented virtual organizations. Similarly, stakeholders feel the need for business-driven and business-level policy specifications, to facilitate the formulation of security and privacy requirements in terms they can understand and manage.

More generally, there is a general need for greater user awareness of—and control on—the policies applied by their own systems and by the services they interact with. As policies and services become more and more complex and volatile, keeping security and privacy under control—without affecting usability—requires suitable tools and methodologies. Users cannot be supposed to be informed a priori about the security requirements of a service, and must be informed about the privacy policy adopted by the service and about the prerequisites needed to use it. The standard conservative approach (give as little information as possible to prevent misuse) is often not appropriate at the business level in open systems. To make services and business more competitive a *cooperative form of enforcement* is crucial (see Section 2.3).

The existing language standards related to security, such as XACML and SAML, are not (yet) rich enough to face the dynamic nature of decision making, the interoperablity issues, and the challenge of increasing user awareness and control over policies.

[*]Università di Napoli Federico II and coordinator of the WG on Policy specification, composition and enforcement of the European Network of Excellence REWERSE, IST-2004-506779. The author is a participant of the W3C RIF working group.

SAML is focussed on encoding authorizations rather than policies. XACML has two major limitations: (i) its rules cannot be used to define application-dependent concepts (whose importance is discussed in the next section); (ii) some fundamental aspects—such as the semantics of policy combinators—is not formalized in the standard. In particular policy combinators are seen just as pieces of arbitrary code, thereby mixing policies and mechanisms and creating obstacles to interoperability.

The trust negotiation frameworks being developed in several projects (e.g., [5, 6, 8] just to name a few) may provide ideas and suggestions for better languages and standards.

## 2 Some major requirements

We shall focus mainly on the language level, and on the mechanisms needed to interpret the policy language.

### 2.1 High-level and business-level policies

The requirement of a high-level policy language, capable of expressing security and privacy constraints in terms of business concepts calls for a flexible abstraction mechanism. The range and variety of possible business-related concepts is extremely large, so it must be possible to define specialized abstractions for each given application or business domain. Such abstractions should be machine understandable in order to support system interoperability (different, independent systems should be able to understand each other's requirements by reducing them to a small set of shared primitives, such as X.509 credentials, for example.)

While semantic web technologies are obviously relevant and potentially useful to this end, it is important to adopt a *lightweight* approach, both in terms of the difficulty and subtlety of the concept definition language adopted, and in terms of the computational complexity of inference.

The cost of the concept definition (or adaptation) phase should be moderate, and—ideally—accessible to people with the standard profiles found in software companies. An approach involving massive intervention of specialized knowledge engineers is likely to fail.

*Rule-based approaches* are appealing alternatives to description logics, in this context. Users spontaneously tend to formulate policies as rule sets. Moreover, semantics tends to be simpler and computational complexity lower.

A language for business-level policy should also be properly integrated with a representation of business processes, including their dynamic aspects. There is increasing interest for frameworks that allow to talk about processes, business rules, security, privacy, QoS, and service level agreements (SLAs) in a uniform way, facilitating their harmonization. Since all these concepts involve some kind of decision process and behavior control based essentially on the same kind of data, it has been proposed to give the term "policy" a broad meaning, encompassing them all [5, 3].

### 2.2 Expressive languages for flexible policies

The level of trust in a peer or a service is essential for making sensible access control/privacy-related decisions in an open environment where interacting peers often have no prior relationship.

Trust can be "built" and enhanced by gathering a variety of properties and attributes of users, agents, and services, ranging from *strong* (cryptographically verifiable) evidence to *soft* evidence (such as a reputation measure originated from a whole community's experiences) and even *unsigned declarations*, such as the copyright agreements accepted by pushing an "accept" button on a pop-up window.

A careful choice of the trust level (and kind of evidence) is essential in balancing the risks involved in a transaction with respect to the cost of enforcing a policy and setting up the necessary prerequisites (such as a PKI.) Consider that in some scenarios it may be impossible to reach complete trust in a peer, but this is not always a good reason to prevent the interactions with that peer.

A flexible policy specification language should be able to integrate and combine all the spectrum of possible evidence in order to define for all applications an appropriate level of trust for each task.

A flexible language should also be able to interoperate with legacy software and data (because policies often need to refer to information that is already encoded in a company's information system). Even if the policy language is declarative, it should have a proper interface to external packages of various sorts.

Pervasive computing scenarios add more interoperability requirements: policies need to refer to measures of the current location and time, as well as similar dynamically changing pieces of information constituting a *highly dynamic context*.

## 2.3 Dynamic and cooperative enforcement

*Trust negotiation* is an appealing way of enforcing the security and privacy constraints of a set of independent peers that are interacting for the first time [7]. Some of the main related requirements concern of course interoperability (e.g., the peers have to explain to each other what they want).

Besides that, it is important to explain to users the policies adopted by a service to increase the user's trust in the service and to facilitate the use of the system. Similarly, it is important to explain the *decisions* made while enforcing the policy. Suppose that Alice sends her ID to a digital library to download a paper. The digital library service may reject the request. Without an explanation, it is hard to figure out whether the denial is due to a failure in the verification of the ID (which in turn may have several causes), to lack of trust in the issuer, to the fact that Alices' subscription does not cover the paper, or any of the other steps that may go wrong in the access control process.

A clean separation of policies and mechanisms, and the adoption of a declarative policy language enable the automatic generation of such feedback and documentation, thereby reducing significantly the cost of producing them and ensuring perfect alignment between the explanations and the policy actually enforced by the service.

*Provisional policies*, i.e., policies that specify actions, are very useful to help users in getting what they want, e.g. by starting workflows automatically, or directing users to suitable web services (such as credential repositories or certification authorities) in case the user needs to perform some preliminary operation before using the service.

In more general terms, we are advocating a *cooperative form of dynamic policy enforcement*. Never say only "no" (if confidentiality permits).

### 2.4 User-friendly front-ends

It is extremely important that users be able to personalize their policies and formulate their own rules, because no generic policy can be tight enough to get the right balance between protection and usability for all persons. Natural language technologies can be very useful for this purpose, too.

The design and implementation of such natural language front-ends is much simpler—and the user interface more effective—if the internal target language matches the granularity and the structure of the specifications that users spontaneously tend to formulate. Again, a declarative approach supporting high-level and business-level domain-dependent concepts seems the best approach to follow.

Graphical formalisms may provide further user-friendly approaches to policy formulation, possibly addressed to technical staff that—for example—might benefit from extensions of UML suitable for rule-based policies. In general, all aspects of security engineering are regarded as a major research issue.

## 3 Protune

Protune (*Pro*visional *tru*st *ne*gotiation) is the trust negotiation framework of REWERSE. It tackles simultaneously almost all the requirements mentioned in the previous chapters.

Protune's policy language is *rule-based* and supports limited forms of *actions*, with a semantics that naturally merges declarative and dynamic aspects. The language is expressive enough to define sophisticated access control policies and credential release policies, as well as simple *business rules*; this is a first step towards the integration of business rules with security and privacy policies. Actions can also be used to support *cooperative* policy enforcement.

Protune's rules are general enough to define domain-specific predicates, and hence *application-dependent ontologies*. Eventually domain-specific predicates are grounded on a small set of shared primitives: *X.509 credentials* and *unsigned declarations*.

A generic syntax for interfacing external packages can be used both for integrating *legacy software and data*, and for integrating (possibly numerical) *reputation measures*. Then Protune covers a wide spectrum of trust levels and forms of evidence.

Protune features an *automated trust negotiation* mechanism. Negotiations are automatically derived from the declarative policies of the involved peers. During negotiations, peers exchange cedentials, declarations, and policy rules (as a means to formulate a peer's requirements for accessing a resource or disclosing a credential—thereby tackling access control and privacy in a uniform way). Since policy rules may be sensitive, they can be protected by *filtering* the policy before releasing it. Negotiations can be adapted and controlled in a declarative way through *metapolicies*, that specify which rules are sensitive and must be protected, which predicates are associated to actions, which peers are in charge of executing each action, which actions should be delayed, etc.

Metapolicies and rule libraries constitute also a powerful *language extension* mechanism, to support new syntax, include new primitives (such as those needed for pervasive computing) and extend the negotiation mechanism as new requirements arise.

Protune has a novel advanced facility for *explaining* (in natural language) policies and the result of negotiations, including failures. Explanation facilities support also *what-if* queries to help users in validating the policies by crafting hypothetical scenarios. the adoption of a declarative policy language has been essential in enabling automated explanations.

Like the rest of the system, the explanation facility is *lightweight*: it requires very little effort to be instantiated in new application domains, and puts very little extra burden on servers because explanations can be built on the clients, making the approach scalable.

The current demo is being turned into a more robust system that we are planning to make publicly available on *sourceforge* by the end of the year. Reputation measures from popular web sites and services will be integrated in a few more months. The negotiation mechanism is implemented on top of *PeerTrust* [6], that in turn is currently grounded on Java technology. The explanation facility is completely new, instead. Rule formulation in controlled natural language is still under development; the final subsystem shall be based on *Attempto* (`http://www.ifi.unizh.ch/attempto/`). Both PeerTrust and Attempto are developed and maintained by members of the WG on Policy Specification, Composition and Conformance of REWERSE (`http://rewerse.net/I2/`).

Further features and details can be found in [5, 2, 4] and on `http://rewerse.net/`.

## References

[1] P. Bonatti, S. Vimercati, and P. Samarati. A Modular Approach to Composing Access Control Policies. In *ACM Conference on Computer and Communication Security*, Athens, Greece, November 2000.

[2] P. A. Bonatti and D. Olmedilla. Policy language specification. Project Deliverable D2, Working Group I2, EU NoE REWERSE, March 2005.

[3] P.A. Bonatti, C. Duma, N. Fuchs, W. Nejdl, D. Olmedilla, J. Peer, and N. Shahmehri. Semantic web policies - a discussion of requirements and research issues. In *Proc. of the European Semantic Web Conf. (ESWC 2006)*, pages 712–724, 2006.

[4] Piero Bonatti, Daniel Olmedilla, and Joachim Peer. Advanced policy explanations. In *17th European Conference on Artificial Intelligence (ECAI 2006)*, Riva del Garda, Italy, Aug-Sep 2006. IOS Press.

[5] Piero A. Bonatti and Daniel Olmedilla. Driving and monitoring provisional trust negotiation with metapolicies. In *IEEE 6th International Workshop on Policies for Distributed Systems and Networks (POLICY 2005)*, Stockholm, Sweden, June 2005.

[6] Rita Gavriloaie, Wolfgang Nejdl, Daniel Olmedilla, Kent Seamons, and Marianne Winslett. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *1st First European Semantic Web Symposium*, Heraklion, Greece, May 2004.

[7] William H. Winsborough, Kent E. Seamons, and Vicki E. Jones. Automated trust negotiation. DARPA Information Survivability Conference and Exposition, IEEE Press, Jan 2000.

[8] T. Yu, M. Winslett, and K. Seamons. Supporting Structured Credentials and Sensitive Policies through Interoperable Strategies in Automated Trust Negotiation. *ACM Transactions on Information and System Security*, 6(1), February 2003.