Proteus: A Semantic Context-Aware Adaptive Policy Model

Alessandra Toninelli[1], Rebecca Montanari[1], Lalana Kagal[2], and Ora Lassila[3],

[1]*Dipartimento di Elettronica, Informatica e Sistemistica*
Università di Bologna
Viale Risorgimento, 2 - 40136 Bologna - Italy
{atoninelli, rmontanari}@deis.unibo.it

[2]*MIT CSAIL*
32 Vassar Street, Cambridge, MA 02139, USA
lkagal@csail.mit.edu

[3]*Nokia Research Center Cambridge*
3 Cambridge Center, Cambridge, MA 02142, USA
ora.lassila@nokia.com

Abstract

The growing diffusion of portable devices enables users to benefit from anytime and anywhere impromptu collaboration. Appropriate policy models that take into account the dynamicity and heterogeneity of the new pervasive collaboration scenario are crucial to ensure secure sharing of information. Collaborating entities cannot be predetermined and resource availability frequently varies, even unpredictably, due to user/device mobility, thus complicating resource access control. Policies cannot be defined based on entity's identities/roles, as in traditional security solutions, or be specified a priori to face any operative run-time condition, and require continuous adjustments to adapt to the current situation. To address these issues this paper advocates the adoption of a semantic context-aware paradigm to policy specification. Context-awareness allows operations on resources to be controlled based on context visibility whereas semantic technologies allow the high-level description and reasoning about context/policies. The paper describes Proteus that, as a key feature, combines these two design guidelines to enable dynamic adaptation of policies depending on context changes. In particular, the paper shows how ontologies and logic programming rules can be used to leverage policy adaptation.

1. Introduction

The increasing diffusion of portable devices with wireless connectivity enables mobile users in physical proximity of each other to spontaneously and opportunistically form ad-hoc communities. Mobile file sharing, mobile e-campus, emergency response, and vehicle coordination are typical collaborative application examples that illustrate the novel opportunities promoted by mobile technologies.

However, the complex security challenges that arise from the increased degree of openness and dynamicity of the pervasive collaborative scenario are currently limiting the widespread uptake of anywhere and anytime collaboration. Collaborating participants cannot be statically pre-identified; they usually change frequently, forming continuously varying ad-hoc coalitions with entities entering and leaving the coalition dynamically. While roaming, entities can establish opportunistic collaboration with dynamically discovered partners of interest without having a long-term pre-established trust relationship with them. In addition, the high dynamicity of the operative conditions under which the interactions between entities take place complicates security management. Traditional security systems operate in environments where changes in the set of both service clients (users/devices) and accessible resources are relatively small, rare, or predictable. On the other hand, security solutions for pervasive environments have to take into account the frequent changes caused by user/device mobility in physical user location, in accessible resources, and in the visibility and availability of collaborating partners. The conditions defined at design time to control and govern resource operation and sharing can be unpredictably different from the ones that actually hold at execution time when entities attempt to access some resources.

To protect ad-hoc collaborations, there is the need for appropriate security models/systems that consider the high unpredictability, heterogeneity, and dynamicity of pervasive environments. To address these issues, we advocate the adoption of a context-centric policy model that treats context as a first-class principle for policy specification and enforcement unlike traditional subject-centric solutions where context is an optional element of policy definition that is simply used to restrict the applicability scope of security policies. Hereinafter, at a high level, the term "context" is defined as any

information that is useful for characterizing the state or the activity of an entity or the world in which this entity operates [1]. In our scenarios, it is not possible to define a policy without the explicit specification of the context that makes that policy valid. Drawing inspiration from the RBAC model that exploits the concept of role as a mechanism for grouping subjects based on their properties [2], we state that, the same as with role, the concept of context can provide a mechanism for grouping policies and for evaluating applicable ones that simplifies policy management, increases policy specification reuse and makes policy update and revocation easier. In pervasive environments, instead of associating policies directly to the subjects and defining the contexts in which these policies should be considered valid and applicable, a system administrator defines the contextual conditions that govern entity's operations on it. When an entity operates in a specific context, she automatically acquires the ability to perform the set of actions permitted/obliged in the current context.

This paper focuses on the advantages stemming from a context-centric approach to policy management to leverage policy adaptation. Policy adaptation is crucial in pervasive environments, being the conditions that characterize user-resource interactions largely unpredictable: policies cannot be all specified a priori to face any operative run-time situations, but may require dynamic adjustments to be able to govern operations on resources even in presence of unexpected changes. We use the generic term "adaptation" to describe the ability of a policy-based management system to adjust context and policy specifications in order to enable policy enforcement in different, possibly unforeseen situations. For instance, consider the case of an access control policy. If adaptation support is lacking, a request for a controlled resource is typically resolved only if the context specified for activating the policy exactly matches the query's context. However, this is too restrictive in pervasive environments where the representation of the contexts collected from various sources is generally different from the one in the policy specifications, even though often sharing equivalent meanings. Instead, it is desirable to resolve a resource query by analyzing the relationships, e.g., semantic equivalence, between the query's and the policy's contexts and not relying on their exact match.

Policy adaptation requires appropriate modeling of context information and of policy elements and requires expressive reasoning about the relationships between the elements of a pervasive system, i.e., the context and the management policies. Semantic technologies represent a key building block for supporting expressive context/policy modeling, reasoning and adaptation.

This paper describes an implementation of these ideas in the Proteus[1] policy model that integrates the context-awareness design principle and ontological technologies to support context/policy modeling and reasoning and that provides support for dynamically adapting policies to varying contexts. The Proteus model extends the model presented in [3] in its support of policy adaptation.

2. Motivating Example

In the remainder of the paper, to point out the characteristics of the Proteus model we consider the case of an international exhibition, like the MotorShow held in Bologna once a year. On that occasion there are hundreds of ongoing ad-hoc meetings opportunistically created among co-located people where different sets of information/data need to be delivered to a wide number of on-site community members, e.g., journalists, to improve effective collaboration on a joint project, such as fund raising activities to support MotoGP Championship. Suppose that the journalist Alice attending the meeting is willing to share some of her personal data relevant for the project with other co-located meeting participants. Proper access control policies govern operations on Alice resources. Once the meeting is terminated, Alice needs to revoke the permissions that allow participants to view her documents.

Suppose that also the following additional policies apply to Alice: during the meeting Alice's boss tries to call her on her corporate mobile phone. When she is in meeting with a client, Alice cannot normally answer the mobile phone. Her mobile phone has to automatically deviate the call to Alice's answering service. However, in case her boss is calling during work time and she is not able to answer the call, Alice is obliged to send him an SMS to explain why she cannot answer the phone.

As it stems from the example, Alice's related policies depend on several contextual information: access should be granted to those who are currently located in the same room where the resource owner is located, if they actually participate in the activity/project relating to the meeting, and for the scheduled duration time of the meeting. Alice's obligation policies also depend on contextual information.

This simple example demonstrates the need for a new approach to policy specification that not only defines policies based on context information, but also allows the seamless adaptation of policies based on current context. For example, let us consider the case of a meeting that continues beyond its originally scheduled

[1] Proteus is the name of a marine god of the ancient Greek mythology that was able to change his shape into different forms.

end time. It is essential to ensure that meeting participants can continue to access each other's resources as long as the meeting is actually taking place. It is therefore necessary to adapt previous policies to prolong resource access permission beyond meeting scheduled duration. In addition, suppose that Alice cannot send the SMS message to her boss because of a poor GSM network coverage in the meeting room or because of she has exhausted her mobile phone monthly corporate credit. Policies ruling phone answers and SMS sending should be adapted to allow Alice to find a way to communicate with her boss even in the unexpected above described condition.

Adequate context and policy adjustment mechanisms give the policy framework the maximum opportunity for dynamic adaptation to unforeseen situations while assuring the security administrator that the entity behavior will be kept within desired bounds. In the absence of policy adaptation support, for example, in the meeting scenario, access to the policy owner's resources would be denied after the scheduled time, since the conditions that limit the applicability of the policy, specifically the condition concerning time, would be evaluated to be false.

3. The Proteus Framework

Proteus is a semantic context-aware policy model that is centered around the concept of context. We consider context to be any characterizing information about controlled system entities and about their surrounding world. Contexts act as intermediaries between entities and the set of operations that they have to and/or can perform on resources. For each context, policies define operations on resources. In particular, policies can be viewed as one-to-one associations between contexts and allowed/obliged actions. Entities should and/or can perform only those actions that are associated with the contexts currently in effect (*active context*), i.e., the contexts whose defining conditions match the operating conditions of the requesting entity and of the environment as measured by specific sensors embedded in the system. We define policy activating contexts as those contexts relevant to specific policies: the activation of a context either causes the activation of permissions or determines the actions that should be performed. Activating contexts of interest are determined by the defined policies (*authorization activating contexts* by authorization policies, *obligation activating contexts* by obligation policies).

We note that there may be a great number of policy activating contexts defined in the system, with each one defining specific sets of permissions/obligations. Proteus activates those contexts by following an approach similar to the one presented in [4]. In particular, resource access requests trigger the evaluation of the authorization contexts in effect, whereas any relevant change in the conditions defining obligation activating contexts determines the activation of obligation contexts.

3.1 Semantic Context Model

The Proteus context and policy model is based on an underlying system model that describes the interactions occurring in a system using the concepts of entities and actions. An entity represents any actor or resource in the system and it is logically characterized by a number of properties that are expressed as attribute-value pairs. An action describes an activity an actor is able to perform on another entity. The action is performed within a specific operating situation, which we call the action context. The action context includes attributes that qualify the action, such as the target on which the action is performed, and the entity that is performing the action. An interaction defines an association between an entity and an action.

A Proteus activating context consists of all information considered relevant for policies, logically organized in parts that describe the state of controlled resources, such as availability or load (the *resource* part), the actors operating on resources (the policy/resource owner and the requestor), such as their roles, identities or security credentials (the *actor* part), and the surrounding environment conditions, such as time, or other available resources (the *environment* part). In particular, Proteus models an activating context as a set of attributes and predetermined values, labeled in some meaningful way and associated with desirable semantics [3]. Instead of a single value, an attribute could also define constraints for a range of allowed values. An attribute value can be assigned to a fixed constant or can be a variable over a value domain. The current state of the surrounding world is also represented in terms of attribute/value pairs where the attribute values represent the output of sensors (with the term "sensor" used loosely). For an activating context to be "in effect", the attribute values that define the current state of the world have to match the definition of the context (as given above) [6].

An activating context can be *minimal*, i.e., formed by a single attribute/value pair, or *composed* of minimal contexts, i.e., defined by multiple attribute/value pairs. Over attribute/value pairs of minimal contexts it is possible to operate with *unary* relationships. Unary relationships include the:

- *is-a/part of* relationship that allows to create is-a/part-of hierarchies of minimal context attributes and/or of their values.
- *negation* relationship that allows to define a minimal context by constraining the values that the attribute should not assume to make that context in effect.

Two or more minimal contexts can be combined to form a composed context by using *n-ary* relationships that are:

- the logical *conjunction* relationship that forms a composed context by logically "and"-ing all the attribute/value pairs of the minimal constituent contexts. The composed context is active if all attribute/value pairs are in effect.
- the logical *union* relationship that allows to define a composed context by logically combining all the attribute/value pairs of the minimal constituent contexts. The composed context is active whenever some of its attribute/value pairs are in effect.

Proteus also allows the definition of constraints over different activating contexts to deal with possible conflicting situations. Conflicts may arise for various reasons. For example, two activating contexts may have common attributes that have incompatible values when contexts are both active; two activating contexts cannot be simultaneously valid according to application-specific requirements, similar to the separation of duty in traditional RBAC models. These constraints are intended to be used at design time to support proper modelling and layout of activating contexts. Proteus distinguishes *overlapping* and *disjoint* activating contexts in order to support conflicts. Overlapping activating contexts share at least one context attribute, even through complex semantic path relations. Disjoint activating contexts do not have any common attribute. Overlapping and disjoint activating contexts may be compatible or incompatible. We define two overlapping activating contexts to be incompatible if they have an attribute intersection but they cannot be active together for any given state. This might happen either because of a dynamic conflict over the common attribute value domain, such as in the case of the location of an actor that cannot be obviously simultaneously in two different places, or for application-specific/management reasons, such as in the case of an actor role that cannot be a student and a teacher simultaneously. We define two overlapping activating contexts to be compatible if their common subset of attributes can be satisfied. Similar considerations apply to disjoint activating contexts.

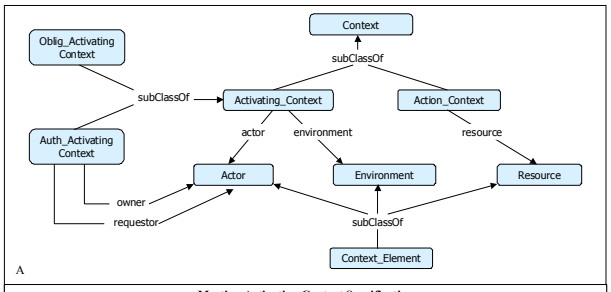
Context Representation. We adopt description logics (DL) and associated inferencing mechanisms to model and process protection context data. In particular, we use Web Ontology Language (OWL)-based ontologies as shown in Figure 1A. An activating context is defined as a subclass of a generic context. Each generic context consists of several context elements, with each element characterized by at least an identity property and a location property defining the physical or logical position of an entity, and eventually by other additional specific properties. For example, an action context is a subclass of a generic context that includes also the resource context element. In particular, a Proteus activating context consists of the actor and the environment context elements. In addition, an activating context can either be an authorization activating context or an obligation activating context.

Let us note that the use of DL-based modeling allows the representation of unary and n-ary relationships between contexts using the logical constructs provided by OWL. In particular, nested contexts are represented by means of the OWL property *subClassOf*. We can create OWL class hierarchies both for context attributes, i.e., context_elements, and for their values. Conjunction of contexts can be obtained using the OWL construct *intersectionOf*; union of contexts is represented by means of the OWL construct *UnionOf*. The contextual intersection of two activating contexts is determined by checking whether they define OWL restrictions on the same properties for the same context_elements. Incompatibility between activating contexts is represented by means of the OWL *disjointWith* property. Negation of contexts could be represented using the OWL construct *ComplementOf*. However, since most reasoners do not properly support negation, we have decided not to include the negation construct in our DL-based context model.

The use of DL in context modeling and reasoning has well-known benefits [6]. For instance, by considering activation contexts as classes and a set of sensor inputs (i.e., the current state of the world) as individuals, DL-based reasoning calculates the activation contexts that are in effect by verifying which activation context classes the current state is an instance of, and by figuring out how defined activating contexts relate to each other (nesting, etc.).

However, DL-based reasoning may not always be sufficient. Our model needs more expressive context reasoning in order to be effective. On the one hand, we need to correlate contexts using not only class definitions (as in pure DL-based reasoning) but also property path relationships between anonymous individuals. For instance, in the meeting example we need to state that if the resource owner is located in a certain place and the resource requestor is located in the same place, the two are co-located. On the other hand, we need to bind the context attribute values to specific instances depending on application-specific context attribute/value relationships. For instance, to enforce the meeting-related authorization policies, we must be able to determine, at each moment, what the actual current project is, so that the corresponding resources belonging to each actor are identified and protected. To overcome some DL-based reasoning restrictions we combine it

with LP-based reasoning. In particular, following the approach described in [3], we define two types of rules: context *aggregation rules* to support reasoning using property path relationships and context *instantiation rules* to provide OWL assertions for attribute values. For instance, the condition of co-location between two collaborating entities at a meeting is expressed with an aggregation rule, whereas the condition of current project with an instantiation rule.



B	Meeting Activating Context Specification
Meeting_Context ≡ Auth_Activating_Context ⊓ ∃owner.Meeting_Actor ⊓ ∃requestor.Co-located_Meeting_Actor	
Meeting_Actor ≡ Actor ⊓ ∃is_currently_working_on.Current_Project ⊓ ∃located.Meeting_Space ⊓ ∃is_involved_in.Current_Project	
Meeting_Action ≡ Access_Action ⊓ ∃action_context.Meeting_Action_Context	
Meeting_Action_Context ≡ Action_Context ⊓ ∃resource.Current_Project_Resource	
Current_Project_Resource ≡ Project_Resource ⊓ ∃is_resource_of_project.Current_Project	
Co-located_Meeting_Actor ≡ ... (see Table 1)	

Figure 1. Context ontology model and an OWL authorization activating context specification example.

Both types of rules are expressed according to the following pattern:

if context attributes $C_1 \ldots C_n$ then context attribute C_m

The above pattern corresponds to a Horn clause, where predicates in the head and in the body are represented by classes and properties defined in the context and application-specific ontologies.

Authorisation Activating Context Example. As an example of context representationwe focus on an OWL-based authorization activating context representation related to the meeting scenario depicted in Section 2. As Figure 1B shows, this example assumes that each actor taking part in the meeting owns a set of resources that relate to the project/activity the meeting is about and shares these resources with the other participants. In particular, the authorization activating context shown in Figure 1B grants access to these resources under certain conditions: the resources must be specifically pertaining the project discussed at the current meeting; the resource owner must be involved in the meeting's project as "project partner", must be currently working on the project-related set of resources, and must be located in the place where the meeting is planned to take place to guarantee that he is attending the meeting. The entities requesting access to these resources must be involved in the project as "project partners", must be co-located with the resource owner, and must be currently working on project-specific resources on their devices.

The activating context may have attribute values assigned to constants or to variables. In the latter case, attributes are assigned proper values by combining DL-based and LP-based reasoning over the context ontology and the context aggregation and instantiation rules.

Colocated Meeting Actor Specification	
Colocated_Meeting_Actor ≡ ∃is_currently_working_on.Current_Project ⊓ ∃is_involved_in.Current_Project ⊓ ∃colocated_with.Resource_Owner	
Instantiation Rules to be applied in case of an ordinary scheduled meeting	
Current_Meeting_Rule	Scheduled_Calendar_Slot (?x) ∧ Meeting (?x) → Current_Meeting (?x)
Current_Project_Rule	Current_Meeting(?x) ∧ Project(?y) ∧ meeting_on_project(?x,?y) → Current_Project(?y)

Table 1. Colocated Meeting_Actor class specification and instantiation rules.

Table 1 shows the definition of the Co-located_Meeting_Actor context element and provides examples of LP rules. In the activating context of the meeting policy, shown before, the resource owner property must belong to the Co-located_Meeting_Actor class. Let us consider the restrictions applying to the properties is_currently_working_on and is_involved_in. These properties are restricted to a variable value, represented by the Current_Project class. This is intrinsically a variable value since the current project varies over time due to the changing activities of the resource owner and requestor, thus corresponding to different instances at different time instants. The instantiation rules in Table 1 are used to determine the correct instance of the current project class at access request time. In particular, let us consider the first couple of rules shown in Table 1. The first rule establishes that, if the user's calendar shows a meeting for the current time, then that meeting has to be considered the current meeting. The second rule states that the project discussed at the current meeting is the current project. Once the facts about the user's calendar are inserted into the refinement fact base, the first rule is triggered and the inferred current meeting instance is used as a new fact to trigger the second rule. For instance, if Ducati-Meeting is scheduled on the user calendar, and Ducati-Project is the corresponding project, then Current_Project is replaced by Ducati-Project in the Colocated_Meeting_Actor specification. A new activating context is thus instantiated with the Ducati-Project value and the corresponding policy generated with the instantiated activating context.

3.2 Semantic Policy Representation

Administrators specify OWL-based policies representing ontological associations between actions and policy activating contexts ontology definitions. Figure 2A shows an authorization policy example that controls access to the meeting resources andFigure 2B presents an obligation policy example.

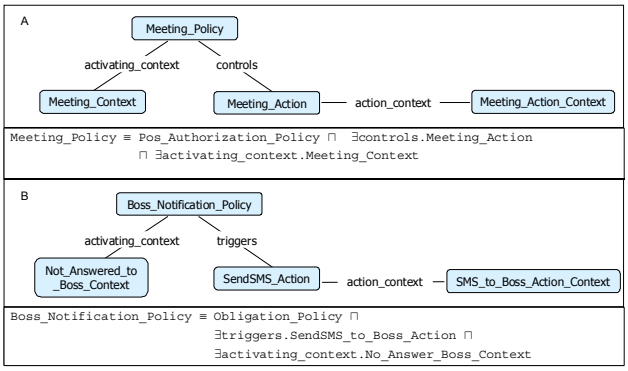


Figure 2. Proteus authorization and obligation policy examples.

Using OWL-based context and policy representation enables the simplification of policy specification and the evaluation, given a certain state, of which contexts are active of the world consequently determining the valid permissions/obligations on the basis of specified policies. In addition, the above-described relationships and constraints over contexts can be exploited to take the maximum advantage of OWL-based reasoning in the process of determining active permission and/or obligations. The logical constructs over contexts are used to infer new contexts and policies from existing ones. Proteus relationships and constraints thus lead to an increase policy reuse and a reduction in the number of policies that administrators need to define.

In particular, if some contexts are organized in a hierarchy, the instances of the more specific contexts are instances of the more general ones. Hence, if a policy associates some permission/obligation with the more general context, the same permission results associated to the more specific contexts in virtue of the subclass relationship between contexts.

In the case of a negative context, it is sufficient to specify one policy whose activating context is the context that must not be in effect to make the associated permission/obligation valid. In this way, the permission will be valid for any state of the world that does not make the defined activating context in effect, thus using only one policy and applying it in many possible different states.

If a context is defined by intersection it is in effect when each minimal context that it includes is in effect. Therefore, any permission/obligation that is associated to each of the constituent minimal contexts gets automatically associated with the context composed by intersection.

Finally, a context composed by union of minimal contexts is in effect if at least one of the minimal contexts that it includes is in effect. Therefore, the policy developer can specify just one policy to associate the union of contexts with a permission/obligation instead of several policies that associate the same permission to each constituent minimal context.

4. Semantic Context-aware Adaptation

The Proteus framework exploits context-awareness and semantic technologies to provide three different kinds of adaptation: policy adaptation, action adaptation and context adaptation. *Policy adaptation* consists of "instructing" the system such that, even though an activating context has changed, it should be still considered active if certain context conditions hold. Policy adaptation automatically prolongs the validity of an active policy even in presence of changes in its corresponding activating context. *Action adaptation* represents the ability to find alternative permitted/obliged actions in case the permitted/obliged actions as determined by the current state of the world cannot be performed. Finding alternative set of actions provides a powerful means of allowing an entity to continue to operate. *Context adaptation* consists of identifying an alternative context where permitted/obliged actions can be performed. Context adaptation can be useful in case of dynamic policy conflicts, such as when an entity is obliged to perform an action that it is not allowed to. Instead of changing the set of permitted/obliged actions, Proteus tries to identify a different activating context where the actions can be performed.

The following subsections describe how Proteus supports the various types of adaptation by using the meeting scenario.

4.1 Policy Adaptation

To describe how policy adaptation works we focus on the case of validity prolongation of the authorization policy of Figure 2A. Suppose that the meeting has gone beyond the allotted time. Given this state, the group of instantiation rules of Table 1 cannot be applied because there are no valid facts in the head of these rules. Therefore, a new set of instantiation rules has to be defined to cover the situation of an extended meeting (see Table 2). In particular, the first rule determines the owner's current project on the basis of her past and current activities, independently from her calendar schedule. For instance, if the last instance of current project (determined at pre-defined intervals or at access request time) was the Ducati-Project, if the calendar does not show any event for the current time, and if the actor

is working on the Ducati-Project, then the Ducati-Project is still the current project instance. The second rule checks for the last and the current scheduling in the actor calendar. If there is no current event, and the last event was a meeting, and that meeting was about the current project (as determined with the first rule), then the last meeting is also the current one. In our example, the current meeting instance is the Ducati-Meeting. The meeting prolongation causes new facts to be inserted in the fact base, which make the set of rules shown in Table 2 valid, while the rules in Table 1 are not valid any more. In particular, the first rule in Table 2 is triggered and the inferred current project instance is used as a new fact to trigger the second rule. Then, the authorization activating context is instantiated by re-writing it with the newly inferred context element values and the corresponding policy generated with the instantiated activating context. To evaluate the validity of the policy, an activating context instance is created that represents the current state of the world, and it is compared with the activating context of the meeting policy by making use of ontology classification to recognize whether the former is an instance of the latter. A more detailed description of the combined LP- and DL-based reasoning process is provided in [3].

4.2 Action Adaptation

Proteus is able to handle the temporary inability of an actor to perform an authorized/obliged action by finding alternative actions. To describe how Proteus supports action adaptation we focus on the case of obligation policy failure in our meeting scenario. In particular, consider the obligation policy stating that whenever an incoming call from the boss is not answered (activating context), an SMS must be sent (action) to the boss (action context). Suppose that in the system an authorization policy is defined stating that when Alice is in a meeting with a client and there is an incoming call on her corporate mobile phone (activating context), the phone software must redirect the call (action) on the answering service (action context). In addition, suppose that when the obligation activating context is valid, Alice is not able to send the SMS because there is very poor GSM network coverage in the meeting room. Therefore, she is currently not able to perform the obliged action, i.e., to send the SMS. Proteus provides adaptation support by looking for an alternative action to the one that Alice is not able to perform. For example, Alice could send an email to her boss with the same SMS content if an authorization policy is defined that allows this.

Instantiation Rules to be applied in case of a meeting prolongation	
Current_Project_Rule-2	Actor(?y) ∧ Last_Current_Project(?x) ∧ is_currently_working_on(?y,?x) ∧ Scheduled_Calendar_Slot(?z) ∧ Idle(?z) → Current_Project(?x)
Current_Meeting_Rule-2	Scheduled_Calendar_Slot(?x) ∧ Idle(?x) ∧ Past_Calendar_Slot(?y) ∧ Meeting(?y) ∧ Current_Project(?z) ∧ meeting_on_project(?y,?z) → Current_Meeting(?y)

Table 2. Instantiation rule examples to support policy adaptation.

Action adaptation requires the definitions of semantically equivalent actions. To represent alternative semantically equivalent actions, Proteus adopts a DL-based specification of the obligation policy where the obliged action is defined as a variable value, i.e., the currently possible communication action, and LP-based instantiation rules to instantiate the value (see Table 3). In particular, the first rule states that the communication action should be "send an SMS", if the actor is able to perform it. In the case it is not possible for the actor to send an SMS, then he could send an email. Let us note that this set of instantiation rules defines an explicit priority order between the alternative actions. Different solutions might be adopted to express a priority, e.g., setting a "mostly preferred" value or a preference hierarchy and encoding them as predicates in the rules. However, it is worth noting that a predicate that allows the selection of only one rule must be specified to prevent inconsistencies during DL-based reasoning.

The alternative action is determined by following the same reasoning process described in Section 4.1. In particular, once the facts about Alice ability/inability to send SMS or email are inserted into the fact base, the second rule in Table 3 is triggered, which causes the possible communication action to be instantiated as the action of sending email. The policy specification is then re-written with the instantiated action value and evaluated by means of DL classification.

Notification Policy Specification	
Boss_Notification_Policy ≡	Obligation_Policy ⊓ ∃triggers.Possible_Comm_Action_2Boss ⊓ ∃activating_context.No_Answered_Boss_Context
Possible_Comm_Action_2Boss ≡	Possible_Communication_Action ⊓ ∃target.Boss
Instantiation Rules to provide action adaptation	
Possible_CommunicationRule-1	SendSMS_Action(?x) ∧ Actor(?y) ∧ is_able(?y,?x) → Possible_Communication_Action(?x)
Possible_CommunicationRule-2	SendSMS_Action(?x)∧ Actor(?y) ∧ is_not_able(?y,?x) ∧ SendEmail_Action(?z) ∧ is_able(?y,?z) → Possible_Communication_Action(?z)

Table 3. Instantiation rule examples to provide action adaptation.

4.3 Context Adaptation

To describe context adaptation in Proteus we focus on the case of obligation policy failure due to a dynamic conflict with a defined authorisation policy [7]. In particular, we consider the obligation policy that forces Alice to send a SMS to her boss whenever an incoming

call from the boss is not answered. Let us also suppose that Alice has a fixed amount of monthly credit on her corporate mobile phone. Once this credit is exhausted, Alice must pay for her own calls. It is the end of the month and Alice has no more corporate credit available on her phone. Therefore, in order to make any call, she needs to explicitly agree to be charged the calls she will make from now on. In alternative, Alice could exploit a end-of-year benefit available to all employees: during the month of December, calls from corporate mobiles are free. Suppose that Alice cannot perform the obliged action not because of a technical impediment, but to the fact that Alice is not permitted to perform that action. For example, Alice is not authorized to send any SMS since Alice has exhausted her corporate monthly credit.

The above scenario can be described in terms of the authorization policies shown in Table 4. Let us note that we do not assume any default system behaviour, e.g., everything that is not explicitly permitted is prohibited or vice-versa. In particular, *(A1+)* states that if the phone user has valid credit, it is permitted to make calls and send SMS. *(A2-)* states that if the phone user does not have valid credit, it is not permitted to make any call, nor send any SMS. *(A3+)* states that during the month of December employees that chose the "Xmas Promotion" are always permitted to make calls and send SMS to local numbers regardless of their credit. Let us note that the Not_Valid_Credit and the Promotion_Activation activating contexts represent examples of compatible disjoint contexts.

(A 1+) Specification
A1_Policy ≡ Pos_Authorization_Policy ⊓ ∃controls.Call+SMS_Action ⊓ ∃activating_context.Valid_Credit_Context
(A 2-) Specification
A2_Policy ≡ Neg_Authorization_Policy ⊓ ∃controls.Call+SMS_Action ⊓ ∃activating_context.Not_Valid_Credit_Context
(A 3+) Specification
A3_Policy ≡ Pos_Authorization_Policy ⊓ ∃controls.Local_Call+SMS_Action ⊓ ∃activating_context.Xmas_Promotion_Context
Xmas_Promotion_Context ≡ Auth_Activating_Context ⊓ ∃environment.December_Env ⊓ ∃requestor.Promotion_Code_Employee

Table 4. Authorisation policy examples for the meeting scenario.

To handle the situation of an actor not permitted to perform an obliged action, Proteus provides support for identifying an alternative context that makes the action permitted for that actor and that can be activated given the current state. In particular, to find the alternative activating contexts that permit the action (which we call hereinafter target contexts) Proteus first searches all defined positive authorization activating contexts that have the obliged action associated with and analyse their semantic relationship with the currently activating contexts in effects. On the basis of these relationships, Proteus can determine whether the found contexts can be activated given the current state of the world. Figure 3 depicts the algorithm executed by Proteus.

Let us now consider our example policies. The currently active contexts, which are determined by applying the combined LP and DL-based reasoning described in Section 4.1, are the activating contexts of the meeting policy, of the obligation policy and of the A2- policy. The target contexts are the contexts of A1+ and A3+. We will call these contexts AC2, AC1 and AC3, respectively.

The first step consists in verifying whether there are attribute values in the current state that cannot be modified to meet the constraints defined in F. For instance, if the month is currently June, AC3 cannot be activated. Let us now suppose that month is December. By further applying the algorithm, we compare AC2 and AC1. These contexts are overlapping since they both include an attribute about credit. In addition, their values are incompatible because a credit cannot be valid and not valid at the same time. Therefore, the activation of AC1 is possible only if there exists a transformation path from AC2 to AC1, such as a rule stating that if the monthly credit is exhausted and the user agrees to be charged the call costs, then the credit becomes valid.

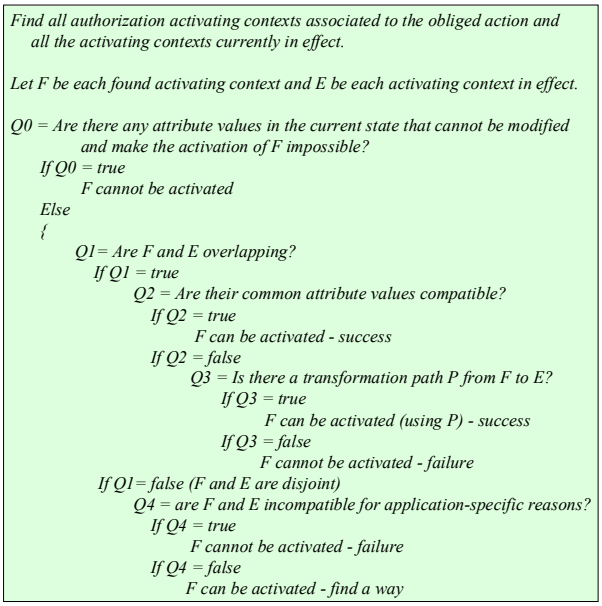


Figure 3. Proteus algorithm for context adaptation.

Let us now apply the comparison to AC2 and AC3. These contexts are disjoint because they do not share any context attribute. In addition, there is no incompatibility between them since A3 does not depend on the phone credit. Hence, Proteus decides that AC3 can be activated.

It is worth emphasizing that Proteus only focuses on determining whether a context can be activated and not

on how this can be achieved. Our model does not aim at providing a support for deriving strategies to achieve the activation of some context. Therefore, in order to find out and execute an appropriate transformation path from one context to another, additional support for goal-based reasoning, e.g., abductive reasoning, is needed. Proteus currently provides support for the specification of LP-rules to express transformation path, but it cannot handle goal-based reasoning over it. Planning techniques can thus be integrated with Proteus to elaborate strategies for the activation of a particular context.

5. Related Work

Several research efforts have addressed the issue of security in dynamic environments. We do not intend to provide a general survey of the state-of-the-art security and policy-based management solutions in dynamic environments, but only to focus on the research that either integrates context-awareness and semantic technologies into security policy frameworks for pervasive environments or addresses the need for dynamic adaptation of policies.

Considering context as a first-design principle is a very recent research direction with only few context-dependent policy model proposals, mainly in the field of access control. The importance of taking context into account for securing pervasive applications is particularly evident in [4] that allows policy designers to represent contexts through a new type of role called *environment role*. Environment roles capture relevant environmental conditions that are used for restricting and regulating user privileges. Permissions are assigned both to roles (both traditional and environmental ones) and role activation/deactivation mechanisms regulate the access to resources. Environmental roles are similar to our contexts in that they act as intermediaries between users and permissions. However, because environmental roles are statically defined in terms of attribute-constant value pairs they cannot be used for policy adaptation. In addition, unlike our approach, in [4] there is no integrated support for representing at a high level of abstraction and reasoning about environmental roles and policies.

The approach proposed in [8] overcomes the semantic gap between contexts specified in the policy at design time and contexts collected from dynamic context sources in pervasive environments: an access request is allowed if the query context is semantically equivalent to the context specified in the policy rule. This approach is similar to our proposed policy model in that it exploits semantic information contexts, but it only addresses access control issues. In addition, no adaptation support is provided to handle unforeseen context conditions.

The attribute-based access control model adopted in [12] is similar to our context-based approach in that it defines a level of indirection between entities and the set of operations they can perform on resources. Attributes represent the set of properties that can be associated to a system entity, e.g., subject, resource or environment, similarly to the Proteus definition of context. However, the approach proposed in [12] only focuses on attribute representation and does not provide any support for policy/context adaptation.

The importance of adopting a high level of abstraction for the specification of all policy building elements (such as subjects, actions, and context) is starting to emerge in well-known policy frameworks, such as KAoS and Rei [9]. KAoS and Rei represent, respectively, significant examples of DL-based and LP-based policy languages. In particular, KAoS uses OWL as the basis for representing and reasoning about policies within Web Services, Grid Computing, and multi-agent system platforms [10]. Contextual information is represented as ontologies and is used to constrain the applicability of policies. The KAoS approach, however, relying on pure OWL capabilities, encounters some difficulties with regard to the definition of certain kinds of policies, specifically those requiring the definition of variables. Rei adopts OWL-Lite to specify policies and can reason over any domain knowledge expressed in either RDF or OWL [11]. A policy consists of a list of rules expressed as OWL properties of the policy and a context represented in terms of ontologies that is used to restrict the policy's applicability. Though represented in OWL-Lite, Rei allows the definition of variables that are used as placeholders as in Prolog. In this way, Rei overcomes one of the major limitations of the OWL language, i.e., the inability to define variables. On the other hand, the choice of expressing Rei rules similarly to declarative logic programs prevents it from exploiting the full potential of the OWL language. In particular, the Rei engine is able to reason about domain-specific knowledge, but not about policy specification. Our policy model shares some commonalities with regard to context/policy representation with both KAoS and Rei, but differs in how it deals with context. Our approach considers context as the primary basis that allows one to deduce which policies apply to a subject acting in the system whereas KAoS and Rei, similarly to traditional approaches, exploit context to build filtering mechanisms for policy applicability.

The policy model in [8] also exploits semantic technologies. In particular, contexts and policies are defined by adopting an OWL-based representation, and OWL inference rules are exploited to derive relationships among contexts.

6. Conclusions and Future Work

The dynamicity and heterogeneity of pervasive scenarios call for context-centric policy models. We propose a semantic context-aware policy model, which treats context as a first-class principle for policy specification and adopts policy definition approach based on DL ontologies and LP rules. We are currently working on implementing a prototype for the meeting scenario using OWL to specify ontologies and SWRL to encode rules. We are also working on the design of a deployment model that includes different components in charge of monitoring contexts, installing policies into the system, performing policy refinement and evaluation, and enforcing policies.

Future work will include integration of additional techniques to identify and execute appropriate transformation path that allow the proper change of contexts. Proteus currently provides support for the specification of LP-rules to express transformation path, but it cannot handle goal-based reasoning that is required find and execute an appropriate transformation path from one context to another. We are currently evaluating the possibility of integrating Proteus with planning techniques to elaborate strategies for the activation of a particular context.

7. Acknowledgement

This work is partly supported by MURST PRIN Project "MOMA: a middleware approach to Mobile MultimodAL web services".

8. References

[1] Dey, A, et al., "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications.", Human-Computer Interaction, Vol. 16, 2001.

[2] Sandu, R., et al. : "Role based access control models", IEEE Computer, Vol.29, No.2, 1996.

[3] Toninelli A., et al., "A Semantic Context-Aware Access Control Framework for Secure Collaborations in Pervasive Computing Environments", Proc. of the Fifth International Semantic Web Conference (ISWC), Athens, GA. LNCS 4273, Springer, 2006.

[4] Covington, M.J., et al.: "Securing Context-Aware Applications Using Environmental Roles", Proc. of the 6th ACM Symposium on Access Control Models and Technologies (SACMAT 2001), Chantilly, Virginia, USA. ACM, 2001.

[6] Lassila, O., et al., "Contextualizing Applications via Semantic Middleware", Proc. of the Second Annual Conference on Mobile and Ubiquitous Systems (MobiQuitous '05). IEEE Computer Society Press, 2005.

[7] Lupu E., et al., "Conflicts in Policy-Based Distributed Systems Management, IEEE Transactions on Software Engineering, Vol 25, No. 6. 1999.

[8] Ko H.J., et al., "A Semantic Context-Aware Access Control in Pervasive Environments", ICCSA 2006, Glasgow, LNCS 3981, Springer-Verlag, 2006.

[9] Tonti, G., et al., "Semantic Web languages for policy representation and reasoning: A comparison of KAoS, Rei, and Ponder", Proc. of the Second International Semantic Web Conference (ISWC2003), LNCS 2870. Springer-Verlag, Berlin, , Sanibel Island, Florida, USA, 2003.

[10] Uszok, A., et al.: "KAoS policy management for semantic web services". IEEE Intelligent Systems, Vol. 19, No. 4, 2004.

[11] Kagal, L., et al.,"A Policy Language for Pervasive Computing Environment" In: Proc. of IEEE Fourth International Workshop on Policy (Policy 2003). Lake Como, Italy, IEEE Computer Society Press, 2003.

[12] Priebe, T., Dobmeier, W., and Kamprath N., "Supporting Attribute-based Access Control with Ontologies". In: Proc. of the IEEE First International Conference on Availability, Reliability and Security (ARES '06). Vienna, Austria, IEEE Computer Society Press, 2006.