Context-based Security Management for Multi-Agent Systems

Rebecca Montanari [1] Alessandra Toninelli [1] Jeffrey M. Bradshaw [2]

[1] Dipartimento di Elettronica, Informatica e Sistemistica (DEIS)
University of Bologna
Viale Risorgimento 2, 40136 Bologna, Italy
{rmontanari, atoninelli}@deis.unibo.it

[2] Institute for Human and Machine Cognition (IHMC)
40 S.Alcaniz Street, Pensacola, FL 32502, USA
jbradshaw@ihmc.us

Abstract

Policies are being increasingly used for controlling the behavior of complex multi-agent systems. The use of policies allows administrators to specify both agent permissions and duties without changing source code or requiring the consent or cooperation of the agents being governed. However, policy-based control can encounter difficulties when applied to agents that act in pervasive environments characterized by frequent and unpredictable changes. In this case, policies cannot be all specified a priori to face any operative run time situation, but require continuous adjustments to allow agents to behave in a contextually appropriate manner. Current approaches to policy representation have been restrictive in many ways, as they typically follow a subject-centric model, which assigns agent permissions and obligations on the basis of agent role/identity information. However, in the new pervasive scenario the roles/identities of interacting agents may not be known a-priori and most important, may not be informative or sufficiently trustworthy. We claim that the design of policy-based agent systems for pervasive environments requires a paradigm shift from subject-centric to context-centric policy models. This paper discusses some issues concerning the specification and enforcement of context-driven policies and presents a novel context-based policy approach that considers context as a first-class principle to guide both policy specification and enforcement. In this perspective, "context" explicitly appears in the specification of security policies and context changes trigger the evaluation process of applicable agent permissions and obligations.

1. INTRODUCTION

Novel programming paradigms based on the agent computing model are increasingly being adopted for the engineering of complex distributed applications [1]. Software agents acting on behalf of end-users can autonomously fulfill user-specific goals, can execute in dynamic and uncertain environments, and can operate within changing organizational structures by responding in a timely fashion and by adapting their behavior to environmental variations.

However, research on agent systems still has to address a number of issues. With rare exception, today's agents have not been deployed in critical, long-lived, secure, or high-risk tasks. Nor do they undertake missions that require widespread agent collaboration among large numbers of agents interacting in complex, unpredictable ways [2].

The non-transparent complexity and inadequate directability of agents remain a major reason for the slow uptake of new research results in fielded agent systems [3]. In response to this concern, agent researchers have focused increasingly on developing means for controlling aspects of agent autonomy in a fashion that can be both dynamically specified and easily understood. Policies represent a promising technique to dynamically regulate a system's behavior without changing code or requiring the cooperation of the components being governed [4]. Through policy, people can precisely express the dynamic bounds within which an agent is permitted to function autonomously and limit the possibility of unwanted events occurring during operations. Policies can be exploited to control agent-to-resource and agent-to-agent interactions (authorization policies) and to impose requirements upon agents to perform given actions (obligation policies), thus ensuring that an adequate level of predictability, security and responsiveness to human control are maintained.

However, policy-based control of agent behavior can encounter difficulties when applied to agents that act and cooperate in pervasive and other highly dynamic environments. The design of agent systems needs to take into account the high heterogeneity of the agent execution environments in terms of resources and computing devices and the high dynamicity that causes frequent changes in the agent execution context. In order to function adequately, agents need to sense and reason about their current context and dynamically adapt their behavior to rapidly changing situations—in other words, they need to be context-aware.

In this case, policies cannot be all specified a priori to face any operative run-time situation, but require dynamic and continuous adjustments to allow agents to act in any execution context in the most suitable way and to follow a contextually-appropriate behavior. Controlling context-aware agents by means of policies requires the specifica-

tion of actions that agents can or should perform whenever the agent context of execution changes. Thus, whenever the context changes, a new policy should be specified in order for agents to adapt their behavior to meet the requirements of the new context.

Current approaches to policy representation have been restrictive in many ways. The specification of policies typically follow a subject-based approach where permissions and obligations are assigned depending on the identity or role of the agent requesting authorization or being required to perform an action.

However, pervasive scenarios often consist of previously unknown agents that move and come dynamically and whose identity may be underspecified or known to be insufficiently trustworthy to allow decisions to be made about what agents can or must do. In fact, it is almost impossible for policy-based agent system administrators to know in advance the identities or roles of all agents that are likely to interact and to request access to their managed resources and services. Instead, administrators can more easily define the conditions for making resources available to agents and for determining agent resource visibility and access according to the context of their operating conditions.

We claim that the design of policy-based agent systems for pervasive and other highly dynamic environments requires a paradigm shift from subject-centric policy models to context-centric ones. In this perspective, *context* explicitly appears as a key building element in the specification of security policies and the context of execution should determine the set of applicable agent permissions and obligations. Instead of managing agent principals and their policy constraints individually, administrators define for each context the set of applicable permission or obligations. When an agent operates in a specific context, it instantaneously acquires the set of permissions active for the related context. When it changes its operating context, its previous permissions are automatically revoked and new permissions are acquired.

In this paper, we compare three approaches to policy representation, reasoning, and enforcement, that have been specifically designed and extensively tested for management of multi-agent and distributed systems. In particular, we highlight similarities and differences between Ponder, KAoS, and Rei, and sketch out some general criteria and properties for more adequate approaches to context-aware policy specification and enforcement. The paper also present the implementation of these criteria and properties within our context-aware policy framework.

The paper is organized as follows. Section 2 highlights the novel policy requirements for context-aware agent systems and analyses how some relevant well-known approaches to policy representation and enforcement, i.e., Ponder, KAoS, and Rei, deal with context information [5], [6], [7]. The comparison allows us to sketch out some general criteria and properties for more adequate approaches to context-based policies. Section 3 presents our proposed context-based policy model and its applicability is discussed in Section 4 through the implementation of a common example of authorization policy. Finally, in Section 5 we present some conclusions.

2. CONTEXT-BASED POLICIES

Novel Policy Requirements for Context-aware Agents

Some initial research efforts are starting to recognize the importance of using context information to secure pervasive and other highly dynamic distributed applications [8], [9]. Context-based security is an emerging approach to cope with the new security problems introduced by the high dynamicity and heterogeneity that characterize pervasive and highly dynamic computing environments.

Traditional security solutions seem inadequate to rule access to resources and interoperation among entities in cases of frequent context changes. In particular, traditional subject-based access control systems exploit user identity or role information to determine the set of user permissions. Permissions are tightly coupled to the identity or role of the subject requesting a resource access, whereas context information can only further limit the applicability of the available permissions. This coupling requires security administrators to foresee all contexts client users are likely to operate. In pervasive environments where client users are typically unknown and where context operating conditions frequently change even unpredictably, the traditional approach to specify access control policy may lead to a combinatorial explosion of the number of policies to be written and may force a long development time and can induce potential bugs. This approach also lacks flexibility: new access control policies need to be designed and implemented from scratch for any user when new context situations occur.

Novel security solutions are required that consider context as the criteria to deduce which policies to apply to a subject acting in the system. Novel approaches should draw inspiration from the RBAC model that exploits the concept of role as a mechanism for grouping subjects based on their properties [10]. In RBAC systems, permissions are first associated with roles, and subsequently subjects are assigned to roles. We claim that, in a fashion analogous to roles, contexts can provide an additional level of indirection between users and permissions. Permissions and obligations are first associated to contexts and subsequently subjects are associated to the contexts they are currently operating in.

The exploitation of context as a mechanism for grouping applicable policies (not as a limit to applicability of already retrieved policies as in traditional access control solutions) simplifies access control management by increasing policy specification reuse and by simplifying policy update and revocation.

The adoption of a context-based policy approach to control agent systems requires the definition of a complete context-based policy model where with complete we mean that the model can precisely identify the basic types of policies required to control agents, can specify how to express and represent the supported policies and how to enforce them. Some general properties have to be considered during the development of context-based policy models for controlling agent systems. We consider the following as basic requirements:

- the ability to model and represent the contexts in which agents operate and to which policies are associated to.
- the ability to define what actions are permitted or forbidden to do on resources in specific contexts (authorizations policies);
- the ability to define the actions that must be performed on resources in specific contexts (*obligations*).
- the ability to dynamically associate agents with contexts.

Considering context in security is a very recent research direction with only a few novel context-based policy model proposals. Belokosztolszki et al. [8] define context as a particular section of an access control application and exploit this notion in order to control information flow during policy evaluation. In this way, they intend to prevent the accidental or malicious overflow of information towards non authorized entities. Covington et al. [9] propose to generalize traditional role-based access control (RBAC), by allowing policy designers to specify environmental context through a new type of role called *environmental role*. Because permissions are associated to roles, which may be both traditional roles or environmental ones, this model aims at overcoming the inherent subject-centric nature of RBAC.

However, the proposed solutions focus on the problem of access control and cannot be exploited to govern and control agent behavior. The aim of this paper is not to provide a general survey of the state-of-the-art in context-based policy representation, but to focus on the analysis of how context information is treated in Ponder, KAoS and Rei. In particular, we first present Ponder which is one of the most well-known policy-based systems for network management that is being evaluated in several universities and industrial organizations also for the control of agent systems, followed by KAoS and Rei, both of which were originally designed for governing agent behavior.

Context-awareness in Policy Models for Agent Control

Ponder

Ponder is a declarative, object-oriented (OO) language that supports the specification of several types of management policies for distributed object systems, such as security policies [5]. Ponder has been extensively deployed and tested in several applications, including large enterprise information systems, and is being evaluated also for the control of agent systems [10]. A Ponder policy is a rule that defines a choice in the behavior of a system. The rule establishes a relationship between a set of subjects and a set of targets by defining how subjects can or must operate on targets. Ponder uses the term subject to refer to users, principals, or automated manager components, which have management responsibilities, i.e., they have the authority to initiate a management decision. A subject can operate on target objects (resources or service providers) by invoking methods visible in the target interface. Ponder distinguishes between basic and composite policies. The fundamental basic policy types are authorizations and obligations. In particular, authorizations policies define "what operations a subject is authorized to do on target objects" while obligation policies define "the actions that policy subjects must perform on target entities when specific relevant events occur".

Ponder offers various possibilities to define contextual constraints within policies, i.e., in which contexts policies should be considered valid and applicable. The simplest way to specify contextual information within a Ponder policy is to exploit the Ponder language construct named policy constraint. Policy developers can use the Ponder policy constraint to define the conditions that must hold for the policy to be applied. Developers can specify these conditions in terms of both application state and environment variables. Three different types of constraints can be specified on the basis of:

- Subject/target state – the constraint is based on the object state as reflected in terms of object interface;
- Action/event condition – constraints can be based on event parameter values in obligations or action parameter values in authorizations;
- Time constraint – used to specify the time validity of a policy.

Context is used to limit the applicability of the policy. For example, let us consider an authorization policy that states that from 2 p.m. to 6 p.m. Doctor Green's patients can invoke a music playing service while they are located in the waiting room of the doctor's. The corresponding Ponder authorization policy is shown in Figure 1a. In this policy, several elements are related to context, such as location and time information and kinds of elements that characterize the involved entities, namely the qualification of the subject as "a patient of Doctor Green" and the type of service (playing music).

In addition, Ponder allows to specify parameterized context conditions through the definition of policy types. Ponder policy types favor policy reusability in different application deployment scenarios. Policy templates can be used to encode common choices in agent behavior and thus facilitate their reuse. An example of policy type is shown in Figure 1b. It defines an authorization policy that rules the access of a subject to a certain kind of service (target) on

the basis of location and time conditions. Once defined the appropriate policy type, the developer will only have to instantiate it for every combination of context conditions, thus avoiding the burden of specifying a new policy for each possible case.

The second possibility that Ponder offers to specify context-based policies is to exploit the *group* policy type. Ponder group policies aggregate authorization and obligation polices together on the basis of some unifying criteria. For instance, we could group together all the policies that govern the provisioning of services within the doctor's waiting room (see Figure 1c), thus establishing the context as the guiding grouping principle.

a)
```
inst auth+ playMusic {
  subject s = /patients/doctorGreen_patients;
  action invoke_service();
  target /services/music_play;
  when s.location = "waiting room" and time.between ("1400", "1800"); }
```

b)
```
type auth+ invokeService (subject s, target t, string t1, string t2) {
  action invoke_service();
  when   s.location = "waiting room" and time.between (t1,t2); }
```

c)
```
inst group waitingRoomGroup {
  inst auth+ playMusic {
    subject  s = /patients/doctorGreen_patients;
    action   invoke_service();
    target   /services/music_play;
    when     s.location = "waiting room" and time.between ("1400", "1800");
  }
  inst auth- playVideo {
     subject s = /patients/doctorGreen_patients;
     action  invoke_service();
     target  /services/video_play;
     when    s.location = "waiting room" and time.between ("1600", "1800");
  }
}
```

Figure 1. Ponder authorization policies

While Ponder generally proves to be founded on a powerful expressive model, especially for its capability to enable the flexible reusability of policy components, some limitations may arise during the deployment and enforcement of context-based policies. In fact, according to Ponder distribution model, authorization policies have to be installed close to target objects, while obligation policies have to be installed close to subject objects. Therefore, even if the developer is allowed to define a set of policies that can be *conceptually* driven by context, the Ponder policy deployment model does not reflect this approach being essentially subject/target oriented [12].

Let us consider, for example, the policy group shown in Figure 3c. and the run time situation of a patient of Doctor Green trying to invoke the music playing service while waiting within the room at 5 p.m. From a context perspective, both policies should be enforced because of the location and time condition. However, policy evaluation is triggered when the patient tries to invoke the music service. The Ponder Access Controller Agent evaluates the contextual conditions only at user access request time because it is not actually *context-aware*. If then the patient tries to invoke the video service, the same conditions will be evaluated again by the Access Controller Agent responsible for the video service management without any possibility for the Ponder policy framework to be aware that the same context is still active. The subject/target oriented policy deployment model does not avoid this second unnecessary context condition check.

To summarize, Ponder seems to offer a powerful language model that allows for the specification of context information within policies and the conceptual management of context-based groups of policies. However, the subject/target-oriented deployment scheme prevents the actual enforcement of a truly context-centric policy model. Moreover, because Ponder does not adopt a semantic-based approach, its expressive capabilities may not be flexible enough to properly model context information and to control the context-driven behavior of agent systems.

KAoS

KAoS is a framework that provides policy and domain management services for agents and other distributed computing platforms [13], [14]. It has been deployed in a wide variety of applications. KAoS policy services allow for the specification, management, conflict resolution and enforcement of policies within agent domains. In KAoS, a policy constrains the actions that an agent is allowed or obliged to perform in a given context. Policies are currently represented in OWL [15] as ontologies. In particular, the KAoS policy ontologies distinguish between authorizations and obligations. Each policy controls a well defined action, while the subject, the action target and other conditions that are used to narrow the scope of the action are all included within the action description, defined as property restrictions on the action type.

In KAoS, context conditions must be directly specified as property restrictions within the action definition. This means that context information is used to limit the scope of the action that is going to be authorized or obliged. Figure 2 shows the example policy we have previously described: from 2 p.m. to 6 p.m. Doctor Green's patients can invoke a music playing service while they are located in the waiting room of the doctor's. Context-related elements, i.e., time and location, and agent qualification (Doctor Green patient), are all defined as property restrictions on the controlled action type. Once specified the action type, the administrator can only define the corresponding policy to allow/forbid or oblige/not oblige the agent to perform that action. Let us note that the ontology-based approach of KAoS allows for flexible definitions of context conditions. For example, the restriction on the subject (denoted by the property *performedBy*) includes both the location and the qualification, i.e., patient, information within a single term. The choice on how to express and possibly aggregate context conditions is left to the policy designer through the use of ontologies. It is worth noting that the action subject must always be defined: in case it is not explicitly stated within the action, then the framework assumes that the policy applies to every possible subject.

The KAoS action-based approach to policy specification, which is particularly suited to control the behavior of agents that are performing some tasks, may limit policy

reusability in scenarios where agents are unknown and act in environments with frequent changes. Let us now consider, for instance, a policy that only slightly differs from the one described in Figure 2a: Doctor Smith's patients are allowed from 4 p.m. to 6 p.m. to access a music service when they are located in the doctor's waiting room.

```
<owl:Class rdf:ID="ContextExample1Action">
  <rdfs:subClassOf rdf:resource="& action;AccessAction"/>
  <rdfs:subClassOf>
    <owl:Restriction>
          <owl:onProperty
rdf:resource="&action;performedBy"/>
          <owl:allValuesFrom
          rdf:resource="#DoctorGreenPatientLocatedInWaitingR
oom"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
          <owl:onProperty
rdf:resource="&action;accessedEntity"/>
          <owl:allValuesFrom rdf:resource="#MusicService"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
          <owl:onProperty rdf:resource="#startingTime"/>
          <owlhasValue rdf:resource=""#14"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
          <owl:onProperty rdf:resource="#endingTime"/>
          <owl:hasValue rdf:resource=""#18"/>
    </owl:Restriction>
  </rdfs:subClassOf>

< policy:PosAuthorizationPolicy rdf:ID="ContextExample1">
  < policy:controls rdf:resource="# ContextExample1Action"/>
  <policy:hasSiteOfEnforcement rdf:resource="#ActorSite"/>
  <policy:hasPriority>10</policy:hasPriority>
</policy:PosAutihorizationPolicy>
```

Figure 2. KAoS policy specification

If compared with the previous policy, this one controls the same action, i.e., accessing a music service, but context conditions have changed as the time constraints apply from 4 p.m. instead of 2 p.m. and the location is Doctor Smith waiting room. In this case, a new action, and consequently a new policy, would have to be written from scratch to take the new context conditions into account. As a general rule, the definition of a new action is required for every possible context, thus leading to the specification of an increasing number of policies when context conditions are supposed to change frequently and even unpredictably.

To solve the problem of how to govern the behavior of previously unknown agents KAoS provides an additional mean to specify policies. In particular, this problem is addressed in KAoS through the use of domains. In fact, KAoS domain services enable to define domain-based policies, i.e., policies that apply to all subjects belonging to a certain domain. In particular, membership in domains can be extensional, i.e., defined by explicit registration, or intensional, i.e., defined by dynamic conditions or states or properties that the agents belonging to the domain must own. Hence, for instance, if a domain is defined of all agents in a given location at a given time, any agent that is in that location at a given time will automatically become a member of that domain for so long as those conditions hold for it, and will be subject to any policies defined on that domain. The inherent subject-centric definition of intensional domains can be useful to handle context information but may limit the types of context that can be specified. In fact, some context conditions, like for instance properties regarding the state of an application, e.g., the number of running threads, or the state of the surrounding world, e.g., date or weather information, cannot be expressed as agents properties or states because they are not directly related to the agent activity. For this kind of context conditions, KAoS domain specification does not enable the flexible definition of context-based policies.

As far as the policy enforcement model is concerned, at present KAoS framework supports the enforcement of both authorization and obligation policies by adopting an action-based approach. The deployment model basically consists of some basic components, i.e., Guards and Enforcers. Each actor in the system is associated with a guard, which is responsible to interpret policies and pass them on enforcers. In particular, when the guard receives a new policy, it determines the types of actions controlled by this policy and takes appropriate decisions in order to install an enforcer for each type of action. The enforcer is in charge of intercepting every tentative action of that type and of determining whether the agent is authorized or not to perform that action.

Rei

Rei is a policy framework specifically designed for policy specification, analysis and reasoning in a pervasive environment [7].[1] Rei policy language allows to define the deontic concepts of right, prohibition, obligation and dispensation, which are formally expressed using the first order logic.

A policy consists of several *has* rules.

has(Subject, PolicyObject)

The field *Subject* may be a URI identifying a certain entity or a variable, allowing any entity that satisfies the policy object conditions to own the corresponding policy object. The construct denoted as *Policy Object*, which corresponds to the concepts of right, prohibition, obligation and dispensation, is represented as

PolicyObject(Action, Condition)

The field *Condition* within a policy object is used to express constraints about the actor, the action or the environment, e.g., context conditions.

[1] Rei is currently being redesigned. The current analysis is based on the available version of Rei.

Let us now examine Rei approach to policy specification, with particular regard to the specification of context conditions. Figure 3a represents the example policy stating that Doctor Green patients are allowed from 2 p.m. to 6 p.m. to access a music service when they are located in the doctor's waiting room. In the first row, a policy object modeling the right is created, while in the second row this right is associated to the variable *Var*, so that every entity that is a patient of Doctor Green and is currently located in the waiting room is granted the permission to invoke the music service. This construct may also be used to define group or role based policies, by specifying the role or group characteristics, which are typically domain dependent, as part of the condition of the policy object.

right (invokeMusicService, (and(patient(doctorGreen, Var)), (located(waitingRoom, Var), (timeBetween(14, 18)))	**a)**
has(Var, right invokeMusicService, (and (patient(doctorGreen,Var), located(waitingRoom, Var), timeBetween(14, 18))))	**b)**
newConstraint (patient, [doctor:String, patient:String], [2]) *has(john, right (phoning, located(john, room5)))*	**c)**

Figure 3. Rei policy specification

As far as the specification of context conditions in concerned, Rei allows the policy administrator to define application specific constraints, since conditions are typically dependent from the particular domain of application. This is especially true for context conditions, which normally have to be defined with regard to the specific application they refer to. The definition of the constraint used as a policy object condition in Figure 3a is shown in Figure 3b. The *newConstraint* clause allows to describe the parameters of the condition and to specify the positions of those parameters that the associated agent can be bound to. In our example, the parameter *patient*, which is located in the second position, can be bound to the subject of the policy by means of variable unification. It is worth noting that all policy objects only have one value, i.e., *Subject*, that can be used for binding with variables in conditions. We believe that this restriction may impose some limitations to the definition of context conditions. In particular, in order to assign a policy object to an agent, the only types of condition that can be dynamically evaluated are those directly referring to the subject. For instance, let us consider the following policy: "John is granted some rights, e.g., using the printer or using the phone, depending on the room where he is currently located in". The rationale behind this kind of policy definition is to allow actions to be performed on the basis of conditions that are not directly related to a certain subject, not even a role, but are determined by a certain context. For the sake of simplicity, we consider here a constant, i.e., John, to be the policy subject; however, the following considerations are intended to hold even in case of a variable subject. In order to specify a Rei policy rule (a *has* association), we first need to know which are the rooms where John is allowed to perform some actions, for example using the phone. Let us suppose that this authorization is valid for room n.3, n.5 and n.7: for each of them, we will have to write a policy rule like the one shown in Figure 3d. Let us now suppose that the rooms where the policy is to be enforced are not three, but three hundred: then, we will have three hundred identical policy rules all referring to John as a subject. Moreover, if the building administrator decides to change the policy for some rooms, then the involved policy rules applying to John must be revised. The problem is that we are not enabled to define any variable within conditions except the subject, because a policy object can only be associated with an agent and not, for instance, with a place.

Rei offers another possibility to express context conditions. In fact, to allow more contextual information to be captured and permit a deeper understanding of the action and its parameter, Rei provides a representation of action that follows the pattern below:

action(ActionName, TargetObject, Pre-Conditions, Effects)

Pre-conditions are defined as the conditions that need to be true before the action can be performed in a safe, consistent and correct way. For instance, a pre-condition for the action of invoking a music service is that the service is currently available. Pre-conditions may possibly be used to express additional context conditions. However, this use of pre-conditions may result to be not coherent and even tricky with respect to their original purpose. In fact, because in Rei the execution of the action is supposed to be performed outside the policy engine, pre-conditions are not supposed to be directly evaluated by the policy engine. Therefore, the use of pre-conditions to express context information would require the forcing of the original model to evaluate pre-conditions inside the policy engine.

With regard to the deployment model, the Rei framework provides a policy engine in charge of reasoning about the policy specification, which currently supports both a Prolog and a RDF-S interface. The framework does not provide any enforcement model, since it has not been explicitly designed for this purpose.

3. A CONTEXT-BASED POLICY MODEL

We propose a policy model based on context to control the behavior of agents that exploits context as a guiding principle for both policy specification and enforcement. Figure 4 depicts the conceptual model of policy, which is defined as an association between a context and an action.

Agents enter a certain context, thus being automatically associated to the corresponding action. The association between a context and an action may be of two basic types, i.e., a context may authorize or oblige an action. This definition is logically represented in Figure 5a using a Cmap [http://cmap.ihmc.us/].

Context is a complex notion that has many definitions. Here we consider context as any information that is useful to characterize the state or the activity of an entity, in particular of an agent, and any useful information about the world in which this entity operates. This may include information about agent location, about the characteristics of the underlying device, about relationships with other agents and possibly past interactions, and many others. In addition, context may include information about the state of an application, e.g., the number of threads currently running or the amount of used resources, or about the state of the surrounding world, e.g., time, date or weather conditions. Let us note that, while the first type of context is specifically bound to the agent activity or state, the latter is supposed to hold regardless of the agent state because it depends on characteristics of the external world where the agent is currently situated.

The core ontology of context is depicted in Figure 5b. The *SimpleContext* class represents a group of information that all refer to the same kind of context, like for example location information, time or details about the acting agent. We have defined in our basic ontology some subclasses of a simple context, like for example *LocationContext*, *TimeContext*, *IdentityContext*, which respectively express information about the location of the agent, about the time and about the identity, possibly the role, of the agent. Other subclasses may be derived from the root class of context, depending on the requirements of the specific application. A *CompositeContext* corresponds to the concept of "set of contexts" and it is composed of different single contexts, which may be bound by an associative or disjunctive relation, or logically negated. We exploit the OWL constructs that map the AND, OR and NOT operators on classes, namely *unionOf*, *intersectionOf* and *complementOf*.

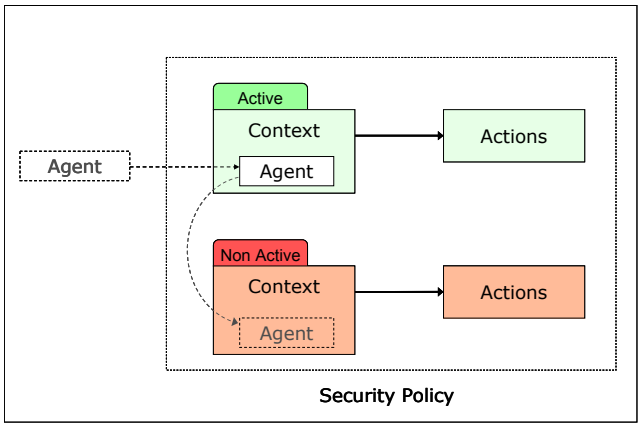

Figure 4. Context-based security policy

The basic ontology of action includes a definition of the action itself and the target on which the action is performed, which is not mandatory. Various types of actions may be sub-classed from the root class *Action*, on the basis of the specific applications. In addition, the ontology may be extended by defining properties that are directly related to the execution of the action, like for instance an encryption mechanism on a communication action or the quality of a printing action.

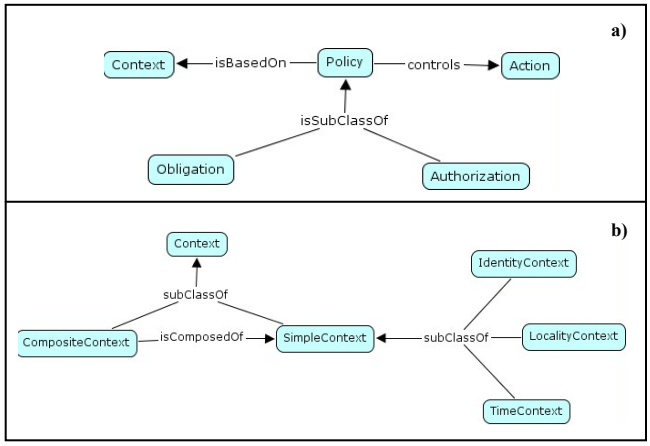

Figure 5. Context-based policy ontologies

We exploit the Web Ontology Language (OWL) to formalize the specification of concepts and properties within the model. Figure 6 shows an excerpt of the OWL specification of the example policy that we have previously discussed.

It is worth stating that our model defines context as a basic element within policy specification, since a policy actually consists of two building blocks, i.e., context and action. This approach significantly differs from the approaches we have previously discussed, where context information is essentially used to restrict the applicability scope of a policy. In those cases, context typically represents an optional, but not fundamental element, while in our model it is not possible to define a policy without a context specification.

Another key issue regards the specification of the action subject. Unlike the previously described models, our model doesn't allow for the explicit specification of the subject of a policy, because we consider policies to be associated with contexts and not with subjects. The subject of a policy is implicitly defined as the agent for whom the policy context conditions currently hold. For instance, the policy represented in Figure 6 applies to every agent that is located in Doctor Green's waiting room between 2 p.m. and 6 p.m., regardless of the agent specific identity This kind of approach is particularly suited to pervasive applications, where interacting agents and interaction modalities may not be always known in advance to the security administrator that is in charge of controlling the behavior of agents.

Moreover, the adoption of an ontological approach to the specification of context-based policies presents some peculiar advantages with respect to a simple attribute-based approach. In fact, it is possible to exploit the information expressed within ontologies and the reasoning capabilities of policy engines to define policies for general contexts that will automatically apply also in specific sub-contexts. Let us consider, for instance, a policy that apply to all stu-

dents of a faculty. This policy would automatically apply to the PhD students of the computer science department without the need of any further specification, because reasoning over ontologies would allow to recognize that PhD students are a subclass of students and the computer science department is part of the faculty structure.

Let us note that this context-based approach may also be used to model traditional role/identity-based policies as role/identity information can be represented as particular kinds of contexts.

Policy enforcement model

Our model includes five middleware components that are in charge of providing support for the specification, installation and enforcement of context-aware access control policies. They are a context manager, a policy specification and a policy installation module, a context-aware security manager and an authorization manager.

```
<owl:Class rdf:ID="ContextExample"
 <owl:UnionOf rdf:parsetype="Collection">
  <owl:Class rdf:resource="#IdentityContextGreenPatient"/>
  <owl:Class rdf:resource="#LocationContextGreen"/>
  <owl:Class rdf:resource="#TimeContext1418"/>
 </owl:UnionOf>
</owl:Class>
<owl:Class rdf:ID="MusicServiceAction">
 <rdfs:subClassOf rdf:resource="& action;AccessAction"/>
 <rdfs:subClassOf>
   <owl:Restriction>
    <owl:onProperty rdf:resource="&action;accessedEntity"/>
    <owl:allValuesFrom rdf:resource="#MusicService"/>
   </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
<policy:Authorization rdf:ID="AuthC271">
 <policy:activatedBy rdf:resource="#ContextExample"/>
 <policy:controls rdf:resource="#MusicServiceAction"/>
 ...
</policy:Authorization>
```

Figure 6. Context-based policy specification

The **Context Manager** (CM) is responsible for dynamically establishing the context conditions of any agent and of the execution environments where agents currently act, thus determining its currently *active* contexts. The agent context is determined on the basis of agent location and characteristics, device description and current conditions that hold within the locality, like for instance time and date. CM needs to coordinate with all the available sources of context information, e.g., a calendar or an agent profile, in order to define context for any agent that tries to perform some action.

The **Policy Specification Manager** (PSM) provides security administrators with editing and browsing tools that assist them in policy specification, revision, static deconfliction and application. We are currently implementing the PSM editor as a graphical user interface that automatically generates policy representation, thus relieving the user from dealing with the complexity of direct policy specification. Policy and context templates allow various classes of policy and context definitions to be expressed as high-level domain-specific abstractions.

The **Policy Installation Manager** (PIM) is in charge of translating the semantic level policies generated by PM into the platform specific representation, and to store them for subsequent retrieval. In particular, PIM installs security policies into the system by storing the specified associations between (set of) contexts and actions into a suitable format, e.g., hash tables. Let us note that, unlike PSM that relies on the use of OWL, PIM is a platform dependent module and must be implemented according to the specific requirements and characteristics of the underlying platform.

The **Context Aware Security Manager** (CASM) is responsible for dynamically computing the set of valid access control policies that apply to a mobile agent, on the basis of holding context conditions. In particular, CASM coordinates with the context management service (CM) in order to be provided with up-to-date context information. For each currently active context, CASM retrieves from the information stored by PIM the associated permissions. Among the whole set of retrieved permissions, CASM selects the ones applying to each agent that is currently located within the domain and generates a view of agents permissions whose visibility is propagated up to the application level. Different updating strategies for the permission view can be adopted, depending on how often context information is refreshed [16]. In particular, an *eager* strategy requires CASM to constantly coordinate with the context management (CM) service in order to update permission views whenever any significant change in the involved entity context occurs. This strategy is expensive especially in environments with frequent context modifications, but guarantees the prompt update of agents permissions. On the other hand, a *lazy* strategy updates the permission set on demand, i.e., when the agent requests access to a resource. This reduces CASM management load, but may impose a greater delay within agent-resource interaction due to context updating and permission recalculating. The choice of the most appropriate strategy to adopt depends on several factors, ranging from service requirements to the characteristics of the service deployment setting and to the trade-off tolerated between service provisioning optimal choices and context update overheads. Similar considerations guide implementation of protection domains in JDK 1.2 security architecture.

The **Authorization Enforcement Manager** (AEM) mediates agent-resource interactions by granting or denying users the permission to access resources basing on the active context-based policies holding in the system. Agents are not enabled to directly access resources, but their access requests are intercepted by AEM, which decides whether to accept or deny the requests. In particular, AEM coordinates with CASM in order to compute the set of permissions that currently hold for the considered agent. Let us note that, in

case CASM keeps the active permissions view always up-to-date (*eager* strategy), AEM only needs to check whether the requested action is currently included in the list of authorized actions, i.e., the permission view. Another key issue is concerned with users mobility, and specifically with possible inconsistencies arising when user context, e.g., location, changes while the user is performing an action. The most problematic case is when user is performing an access and the newly determined contextual information does not allow her to perform that access any more. In such cases, it is necessary to reach a trade-off between security guarantees and preservation of action consistency. At present we privilege the integrity of transactions, hence the newly determined context and its associated permissions become active only when the user has completed the action.

4. CASE STUDY

We have started to evaluate the feasibility of our proposed approach by implementing it in a prototype application, which supports access control for services available within a certain locality, like for instance a building or a shopping center.

Let us consider the following scenario. A newly opened music store has launched a promotional campaign: during the month of April, every Saturday afternoon, people visiting the store are allowed to download an MP3 music file on their portable MP3 reader.

Policy specification

According to our context-centric model, the corresponding access control policy can be informally described as below:

- Composite Context = AND (month is april, day is saturday, client is located in the store, NOT (client has already downloaded), device is mp3 reader)
- Action = access store music service
- Type of policy = authorization

The security administrator is allowed to specify the policy using the PSM service. Using PSM interface, the policy maker defines the context conditions that are going to drive the policy, starting from the simple contexts, i.e., time, location, history and device context, and subsequently defining their logical relation, i.e., AND, as the composite context. Then, the controlled action must be defined, possibly choosing from a set of application-specific predefined actions: in our case the action is a resource access action on a music service. Finally, the administrator must define the kind of policy that binds context and action, which in this case is an authorization policy. PSM is in charge of automatically producing the OWL specification of the policy, which is shown in Figure 7.

The OWL policy specification is then passed to PIM in order to translate it into a platform-specific format and to subsequently store the associated information. In particular, the prototype implementation of PIM parses the OWL policy description to extract the different kinds of information, i.e., context information and action information, then it encodes this information in a string format and finally store it into hash-tables that map the association between context and action.

Policy enforcement

Let us now consider the case of a client that wishes to access the promotional music service on Saturday afternoon. When the agent running on her device connects to the store local network for the first time, the middleware components interact to create the agent context. In particular, CASM retrieves context information about the agent, e.g., location and used device, and about the environment, e.g., date and time, from CM. Then, on the basis of the policy association table, CASM extracts the actions that result to be authorized because the corresponding context is active. These actions are included in a permission view that CASM in charge of keeping up-to-date. For instance, if the client exits the store, she automatically looses the right to access the music service, due to her changed context conditions. In addition, at start-up, the client agent is provided with a reference to AEM that will be invoked in order to access the service.

```
<owl:Class rdf:ID="ContextPromo">
 <owl:UnionOf rdf:parsetype="Collection">
    <owl:Class rdf:resource="#LocationContextMusicStore"/>
    <owl:Class rdf:resource="#DeviceContextMp3Reader"/>
    <owl:Class rdf:resource="#TimeContextMonthApril"/>
    <owl:Class rdf:resource="#TimeContextDaySaturday"/>
    <owl:Class>
        < owl:complementOf >
            <owl:Class
            rdf:resource="#HistoryContextDownload"/>
        </owl:complementOf>
   </owl:Class>
 </owl:UnionOf>
</owl:Class>

<owl:Class rdf:ID="PromoServiceAction">
  <rdfs:subClassOf rdf:resource="& action;AccessAction"/>
 <rdfs:subClassOf>
     <owl:Restriction>
        <owl:onProperty rdf:resource="&action;accessedEntity"/>
        <owl:allValuesFrom rdf:resource="#PromoService"/>
    </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Figure 7. Case study policy specification

When the agent running on the mp3 reader tries to access the music service, its request is actually intercepted by AEM, which decides whether to authorize or forbid the action. In particular, AEM looks at the agent's list of permitted actions (permission view) that is maintained by CASM. If the access action results in the list, which means that all required context conditions are currently active for the agent, then AEM grants the agent the right to access the download service. Let us note that, in case the permission view is not currently up-to-date (*lazy* strategy), AEM first needs to coordinate with CASM in order to update the con-

text of the requesting agent. After the permission view has been refreshed, AEM can properly evaluate the agent request.

5. CONCLUSIONS AND FUTURE WORK

The high dynamicity and heterogeneity of agent execution contexts in pervasive scenarios raise new challenges for the policy-based management of agents behavior. A paradigm shift in the approach to policy representation and enforcement is needed, to move the focus from the identity/role of the acting agent to the context it is operating in. We have proposed a novel context-based policy model, which treats context as a first-class principle both in the specification and enforcement of policies.

From our analysis it seems that the new context centric approach to policy specification and management is starting to attract increasing research efforts with some initial proposals. However, further work is needed to investigate the expressive capabilities of the considered models and to evaluate their suitability and their extendibility to enable effective control of context-aware agents.

ACKNOWLEDGMENTS

Our work is supported by the MIUR FIRB WEB-MINDS and the CNR Strategic IS-MANET Projects. We thank Renia Jeffers, Andrzej Uszok for valuable discussions on context-based policy management for multi agent systems.

REFERENCES

[1] J.M. Bradshaw, An introduction to software agents, in Software Agents, J.M. Bradshaw, Editor. 1997, AAAI Press/The MIT Press: Cambridge, MA. p. 3-46.

[2] J.M. Bradshaw, et al., Making agents acceptable to people, in Intelligent Technologies for Information Analysis: Advances in Agents, Data Mining, and Statistical Learning, N. Zhong and J. Liu, Editors. Springer Verlag: Berlin. p. 361-400 (2004).

[3] J.M. Bradshaw, et al., Taking back cyberspace. IEEE Computer, July, pp. 89-92 (2003).

[4] S. Wright, et al. (eds.), Special Issue on Policy Based Networking. IEEE Network, Vol. 16, No. 2, March, (2002), 8-56

[5] N. Damianou, et al., The Ponder Policy Specification Language, Proceedings of the Workshop on Policies for Distributed Systems and Networks (POLICY 2003), Bristol, UK, 29-31 January, Springer Verlag, LNCS 1995 (2001).

[6] Bradshaw, J.M., et al., Representation and reasoning for DAML-based policy and domain services in KAoS and Nomads. in Proceedings of the Autonomous Agents and Multi-Agent Systems Conference (AAMAS 2003). 2003. Melbourne, Australia: New York, NY: ACM Press.

[7] L.Kagal, Rei: A Policy Language for the Me-Centric Projec, HP Labs Technical Report, HPL-2002-270 (2002). http://www.hpl.hp.com/techreports/2002/HPL-2002-270.html

[8] A. Belokosztolszki, et al., Policy Contexts: Controlling Information Flow in Parameterised RBAC, Proceedings of the IEEE Fourth International Workshop on Policies for Distributed Systems and Networks, (POLICY 2003), Lake Como, Italy, 4-6 June. IEEE Computer Society Press, pp. 99- 110 (2003).

[9] M.J. Covington, et al., Securing Context-Aware Applications Using Environmental Roles, Proceedings of the 6th ACM Symposium on Access Control Models and Technologies (SACMAT 2001), May 3-4, Chantilly, Virginia, USA. ACM (2001).

[10] Sandu, et al., Role based access control models, *IEEE Computer*, Vol.29, No.2, February (1996).

[11] A. Corradi, et al., Policy-driven Management of Mobile Agents Systems, Proceedings of the Workshop on Policies for Distributed Systems and Networks (POLICY 2003), Bristol, UK, 29-31 January, Springer Verlag, LNCS 1995 (2001).

[12] N. Dulay, et al., A policy deployment model for the Ponder language, Proceedings of the IEEE/IFIP International Symposium on Integrated Network Management 2001, 14-18 May, pp.529 – 543 (2001).

[13] J.M. Bradshaw, et al., Terraforming cyberspace, in Process Coordination and Ubiquitous Computing, D.C. Marinescu and C. Lee, Editors. 2002, CRC Press.

[14] A. Uszok, et al., KAoS policy management for semantic web services. IEEE Intelligent Systems, 2004. **19**(4): p. 32-41.

[15] F. van Harmelen, et al., OWL Web Ontology Language Reference, W3C Recommendation 10 February 2004. http://www.w3.org/TR/owl-ref/.

[16] A. Corradi, et al., Context-based Access Control for Ubiquitous Service Provisioning, Proceedings of the28th Annual International Computer Software and Applications Conference (COMPSAC 2004), Hong Kong, 28 – 30 September. IEEE Computer Society Press, pp. 444-501 (2004).