A Quality of Context-Aware Approach to Access Control in Pervasive Environments

Alessandra Toninelli, Antonio Corradi, and Rebecca Montanari

DEIS – Università di Bologna, Viale Risorgimento 2, 40136 Bologna, Italy
{alessandra.toninelli,antonio.corradi,rebecca.montanari}@unibo.it

Abstract. The widespread diffusion of wireless-enabled portable devices creates novel opportunities for users to share resources anywhere and anytime, but makes access control a crucial issue. User/device mobility and heterogeneity, together with network topology and conditions variability, complicate access control and call for novel solutions to dynamically adapt access decisions to the different operating conditions. Several research efforts have emerged in recent years that propose to exploit *context-awareness* to control access to resources based on context visibility and changes. Context-based access control requires, however, to take into account the *quality* of context information used to drive access decisions (QoC). Quality of context has in fact a profound impact on the correct behavior of any context-aware access control framework. Using context information with insufficient quality might increase the risk of incorrect access control decisions, thus leading to dangerous security breaches in resource sharing. In this paper we propose a QoC-aware approach to access control for anywhere, anytime resource sharing. The paper describes the design, implementation and evaluation of the Proteus policy framework, which combines two design guidelines to enable dynamic adaptation of policies depending on context changes: context-awareness with QoC guarantees and semantic technologies to allow high-level description of context/policy specification and reasoning about context/policies.

1 Introduction

Technological advances in telecommunications and mobile device capabilities are paving the way towards an integrated pervasive scenario where users access services anywhere and anytime, even when they are on the move, and engage in opportunistic and temporary resource sharing with other users in absence of a fixed network infrastructure. In the new pervasive scenarios, characterized by high heterogeneity and dynamicity in terms of available services, computing devices/mobile users properties and executing environments, controlling access to shared resources becomes a crucial problem.

Traditional systems rely on a relatively static characterization of the operating conditions where changes in the set of users, devices and accessible resources are relatively small, rare, or predictable. By contrast, user/device mobility causes

C. Giannelli (Ed.): Mobilware, LNICST 0007, pp. 236–251, 2009.

frequent changes in physical user location, in accessible resources, and in the visibility and availability of collaborating partners. Access conditions defined at design time to control resource management and sharing can be unpredictably different from those holding at execution time, when entities actually attempt access to resources. To address this issue, some research efforts have emerged in recent years that propose context-aware access control policy models to control access to resources based on context visibility and changes [1]. The term context can be broadly defined as any information that is useful to characterize the state or activity of an entity or the world in which this entity operates [2]. Differently from traditional solutions where context is an optional attribute used to restrict the applicability scope of security policies, context-aware solutions adopt context as the main design principle for policy specification and enforcement. Context-aware access control policies exploit not only identity/role information, but also other contextual information, such as location, time and ongoing activities, to allow access to resources, thus adapting to dynamically changing conditions. The exploitation of context as a first-class principle to regulate access control brings several advantages. First, it is an example of active security model, i.e. it is aware of the context associated with an ongoing activity and thus distinguishes the passive concept of permission assignment from the active concept of context-based permission activation. Moreover, similarly to the concept of role in role-based access control (RBAC) models, which serves as a mechanism for grouping subjects based on their properties [3], the concept of context provides a grouping mechanism for policies that simplifies policy management by increasing policy reuse and making policy update and revocation easier.

Existing context-aware access control models, however, typically rely on the implicit assumption that context information used to take security decisions is correct and trustworthy. This hypothesis is clearly not compatible with real operating conditions in pervasive scenarios, where context data are acquired by heterogeneous (and possibly not trustworthy) sensors via variable network connections, represented according to different models, and aggregated through various procedures that might introduce additional biases. Controlling access based on context thus requires a careful analysis about the *quality of context* information (QoC) used to take access decisions.

We claim that QoC has a tremendous impact on the behavior of a context-aware access control system. Depending on the quality of used context data, granting access to a resource might be associated to a variable risk level: the less reliable context information is (i.e. the lower its quality), the higher risk is associated to any access action allowed based on that context information. Using context information with insufficient quality might therefore increase the risk of incorrect access control decisions, thus leading to dangerous security breaches in resource sharing.

The importance of considering QoC in designing and managing context-aware systems has recently started to be recognized [4, 5]. We believe, however, that QoC impact on context-aware access control models has been underestimated so far, mainly considered a low level issue dealing with raw data, sensor equipment,

and system performance. As such, QoC management is typically delegated to context provisioning platforms, while higher level applications are context-aware, but largely QoC-unaware. In particular, to the best of our knowledge, none of existing access control solutions addresses the issue of considering QoC when assigning permissions based on context.

In this paper we present a novel QoC-aware access control framework for sharing resources in pervasive environments. In our framework QoC is exploited as a filtering principle to both (i) discard context data whose quality does not comply with minimum QoC requirements and (ii) select applicable policies based not only on current context, but also on the extent to which that context can be considered true. This helps minimizing the risk of granting access to resources based on incorrect or ambiguous context information, while reducing policy evaluation/enforcement overhead by filtering out contexts and policies that are unapplicable due to their inadequate QoC. We have implemented this approach in the Proteus middleware architecture, which exploits QoC-awareness and semantic technologies for the specification and the evaluation of access control policies.

The paper is organized as follows. The Proteus QoC-aware policy model is presented in Section 2, while Section 3 details policy management with QoC-based filtering. Section 4 describes the Proteus middleware architecture and its prototype implementation, which is evaluated in Section 5 by providing some experimental results. Conclusions and future work follow.

2 Proteus QoC-Aware Policy Model

Proteus is a semantic context-aware access control model that is centered around the concept of context. Similarly to roles in traditional role-based access control models, contexts can act as intermediaries between entities and the set of operations that they can perform on resources. For each context, policies define allowed operations on resources. In particular, policies can be viewed as one-to-one associations between contexts and allowed actions. Entities requesting access to resources operate in one or more *active* context, i.e. 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 *protection contexts* those active contexts that have specified permissions associated with. Entities can perform on resources only those actions associated with the protection contexts currently in effect [1].

When activating a set of permissions, Proteus takes into account the quality of information making the policy protection context active. In particular, we exploit quality of context information to filter the set of potentially active contexts: if data describing the current state (as measured by sensors) do not satisfy certain quality requirements, such as freshness or accuracy, they are not considered eligible for activating protection contexts (and associated policies). This approach brings two advantages. First, it minimizes the risk of granting access to resources based on incorrect or ambiguous context information. In

addition, it helps reducing policy evaluation/enforcement overhead by a-priori filtering out policies whose protection contexts cannot be activated because of their insufficient QoC level.

2.1 Context and Policy Model

A protection context in Proteus consists of all characterizing information that is considered relevant for access control, logically organized in parts describing the state of the resource associated with the protection context, such as availability or load (the resource part), the entities operating on the resource (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). A protection context is a set of attributes and predetermined values (called hereinafter *context elements*), labeled in some meaningful way and associated with desirable semantics [6]. Instead of a single value, an attribute could also define constraints for a range of allowed values. Let us note that 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 (called *context assertions*), where the attribute values represent the output of sensors- with the term sensor used loosely. For a protection context to be in effect (active), the attribute values that define the current state of the world have to match the definition of the context. More details on Proteus context model can be found at [1].

A policy is represented as the association of a protection context and an access action. Table 1 shows an example of protection context and policy related to a pervasive healthcare scenario. The policy states that, in case of health emergency, Alice's health protected information (HPI) is accessible by any physician, provided that (s)he is located in the healthcare center and owns a valid credential (e.g., an official certificate).

Table 1. (a) Proteus protection context definition example and (b) Proteus QoC-aware policy example

(a)
PersonalEmergencyContext ≡
ProtectionContext ⊓ ∃ owner.Alice ⊓ ∃ requestor.InHospitalQualifiedPhysician ⊓
∃ resource.AliceHPI ⊓ ∃ environment.PersonalEmergency
InHospitalQualifiedPhysician ≡
Physician ⊓ ∃ has_credential.ValidQualification ⊓ ∃ located.HealthcareCenter
(b)
HPI_Access_Policy ≡ AccessControlPolicy ⊓ ∃ controls.ReadAction ⊓
∃ context.PersonalEmergencyContext ⊓ ∃ policy_qoc.Policy_QoC
Policy_QoC ≡ QualityOfContext ⊓ ∃ has_value.QoC_over0.85

2.2 Quality of Context Model

Proteus model handles QoC at two distinct levels: (i) context elements and (ii) policies.

Proteus associates each context element with a quality attribute. In general, it is possible to define several attributes to evaluate the quality of context information, depending on both context sources and context collection/aggregation mechanisms [4]. In our model we define a base quality attribute, called *Quality of Context* (QoC), which is specialized in several different attributes including freshness, precision, correctness, trustworthiness, relevance and resolution, as shown in Figure 1. Each quality attribute is also associated to a numeric value.

Similarly to context, the required quality of a context element is represented as a set of attributes and constrained (range of) values. For instance, a QoC constraint might require that any assertion about location must be up to date to be considered as part of the current state (e.g., its normalized threshold value for freshness is 0.8, as shown in Table 2). It is also possible to express QoC constraints on specific context element values (e.g., the assertion "Dr. Green is located in the Emergency Room" must have a QoC value higher than 0.6") or to any context element that is asserted in the current state knowledge base (e.g., any context assertion must have the freshness value: very high"). Any context assertion composing the current state is provided with a certain QoC level, represented in terms of attribute/value pairs. For instance, the context assertion "Dr Green is located in the Emergency Room" might be provided with a QoC value of 0.7.

In addition, each access control policy is associated with a QoC threshold value (see Table 1). This value represents the minimum quality level that any

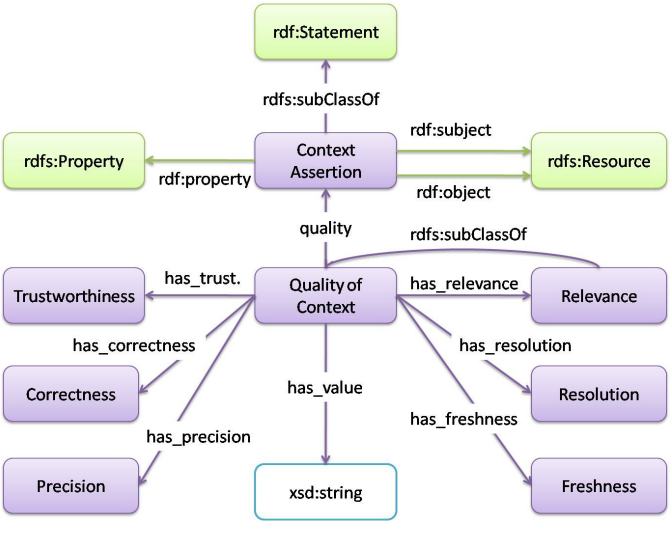

Fig. 1. Quality of Context Ontology

state of the world activating the policy protection context must exhibit. In other words, only if the current state has an overall QoC value exceeding the threshold, the protection context and the associated policy will be considered active. Let us note that this differs from QoC constraints expressed on single context elements, which are not bound to any specific policy.

2.3 Context and QoC Representation

We adopt description logics (DL) and associated inferencing to model and process protection context data. A protection context is defined as a subclass of a generic context and consists of the resource, the actor and the environment context elements. Each context element is characterized by an identity property, and a location property defining the physical or logical position of an entity. Single context elements are characterized by specific additional properties. Figure 2 depicts Proteus base context ontology.

To model QoC, we exploit the support for reification provided by RDF (and inherited by OWL)[1]. Each context assertion can be thought as a triple connecting a subject, a predicate and an object. For example, the assertion "Dr Green is located in the E.R." can be modeled as the following triple:

`(Dr. Green, located, E.R.)`

where the subject is `Dr. Green`, the object is `E.R.` and the predicate is `located`.

Such triple can be considered itself as a piece of information by means of the RDF statement (`rdf:Statement`) abstraction. An RDF statement is the statement made by a token of an RDF triple: the subject of an RDF statement is the subject of the triple; the predicate is the predicate of the triple; the object is the object of the triple.

We rely on this modeling structure and extend the `rdf:Statement` class with a `Context_Assertion` class, which inherits the `rdf:subject`, `rdf:predicate` and `rdf:object` properties. To represent QoC information associated to each context assertion, we add a `quality` property connecting each assertion to a `Quality_Of_Context` class. This class is connected via property to several subclasses of the `QoC_attribute` class representing different QoC attributes, as shown in Figure 1. QoC constraints on context elements are represented as classes whose restrictions define the required quality. Table 2 shows an example of a QoC constraint defining a minimum threshold value for any context assertion about the `located` property.

To calculate active protection contexts based on current state, we rely on DL-based reasoning [6]. For instance, by considering protection contexts (i.e., sets of context elements) as classes and a subset of the current state of the world (i.e., context assertions) as individuals, DL-based reasoning calculates the protection contexts that are in effect by verifying which protection context classes the current state is an instance of, and by figuring out how defined protection contexts relate to each other (nesting, etc.). We also exploit DL-based reasoning to verify

[1] `http://www.w3.org/TR/rdf-schema`

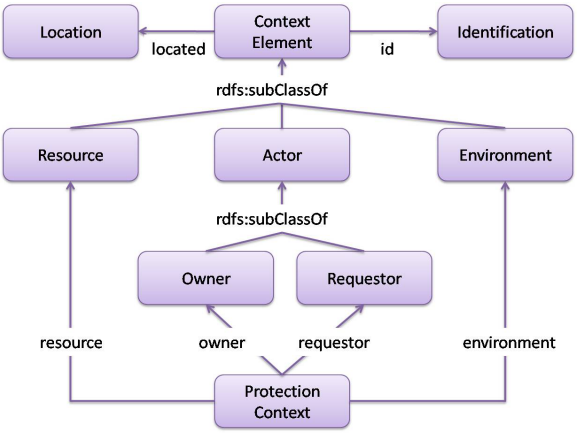

Fig. 2. Proteus Context Ontology

Table 2. (a) Proteus QoC constraint over a context element and (b) Context assertion with QoC value

(a)
QoC_over0.8 ≡ QualityOfContext ⊓ ∃ has_value.{Over0.8}
QoC_constraint ≡ ContextAssertion ⊓ ∃ quality.QoC_over0.8 ⊓ ∃ rdf:property.{located}
(b)
$< Ctx\_assertion\_11, Dr.Green >$: rdf:subject
$< Ctx\_assertion\_11, EmergencyRoom >$: rdf:object
$< Ctx\_assertion\_11, located >$: rdf:predicate
$< Ctx\_assertion\_11, QoC\_value\_11 >$: quality
$< QoC\_value\_11, 0.7 >$: has_value

QoC constraints associated to both context element and policy definition. This is a consequence of our modeling choice, which provides a uniform ontological representation for both context and QoC data. Let us note that, in case QoC constraints are expressed as numerical values, the comparison between required (range of) values and actual QoC attribute values is performed via programming procedures (since it is not supported by DL reasoners).

3 Policy Management in Proteus

To allow access to resources based on both context and its quality, Proteus applies a two-step QoC-filtering process. Recalling the model described in Section 2.2, Proteus allows to define QoC constraints both on context elements and policies. Each type of constraint is exploited during a specific phase of the filtering process.

The QoC-based filtering process is composed of the following two steps:

1. *Context assertions pre-filtering.* Context assertions about the current state are provided with certain QoC values. Prior to evaluating a request access, Proteus retrieves all and only those context assertions that satisfy defined QoC constraints. The output of this filtering phase is a set of context assertions whose QoC values is compliant with imposed quality requirements, regardless of any specific policy.
2. *Protection contexts filtering to activate policies.* Proteus activates policies based on current state information. Each policy has a specific QoC threshold representing the minimum QoC value that any current state must satisfy to activate the policy protection context. Among all potentially active policies (because context assertions, as filtered at step 1, are instances of their protection context), Proteus selects only those policies, whose QoC threshold is reached by the QoC value of the current state. The output of this second step (and of the whole QoC-based filtering process) is a set of active policies compliant with both context element and policy-specific QoC constraints.

3.1 Policy, Context and QoC Specification

The application manager or the security manager can configure QoC constraints for context elements at application deployment time, e.g., by setting values for certain QoC attributes, such as freshness or accuracy, that apply to all context elements. It is also possible to set values for context element QoC constraints based on the specific application domain. For instance, if access control policies are mainly based on location conditions, then the security manager can define QoC constraints on any context element describing location information. Table 2 shows an example of context element QoC constraint definition.

The policy manager/security administrator defines Proteus policies by creating ontological associations between actions and policy activating contexts. Table 1b shows a policy controlling access to a patient's HPI. The policy manager also sets the QoC threshold for the defined policy. Such threshold value typically depends on the sensitivity of the access resource and on the kind of access action as well. For example, a read access on the HPI might be less critical than a write access. OWL-based reasoning over contexts and policies is used to infer new contexts and policies from existing ones, thus allowing policy reuse and simplifying policy evaluation [1].

Let us note that context elements QoC constraints serve as a coarse-grained filter since they apply to all context elements (and consequently to all installed policies), while policy QoC provide a fine-grained filtering mechanism to ensure that each policy is enforced under certain QoC conditions.

3.2 QoC-Based Policy Evaluation

In this section we describe in detail how Proteus evaluates access control policies by applying the two-step QoC-based filtering process.

Context Assertions Pre-Filtering. Each context element might be associated to one or more QoC constraints. These constraints are expressed as OWL restrictions on any RDF statement containing a specific subject, property or object, or any combination of these elements (see, for example, Table 2). On the other side, any context assertion (i.e., a triple composed of a subject, a predicate and an object describing the current state) is provided with a QoC value. Prior to performing policy reasoning, Proteus filters out information describing the current state by selecting only those context assertions whose QoC value satisfies any defined QoC constraint on context elements. The output of this pre-filtering process is a base of context assertions whose quality is compliant with QoC requirements.

Let us note that QoC values might be obtained according to different approaches and mechanisms, from sensor training to utility functions [7]. Being the aim of our present work the design of an access control framework, we do not focus on QoC determination low level mechanisms by relying on existing work on this topic.

Protection Contexts Filtering to Activate Policies. The second filtering step is specific to each policy. Once pre-filtered current state information, Proteus performs DL-based (subsumption) reasoning to determine which protection contexts and associated policies could be activated by the current state. For each protection context, Proteus needs to compare the policy QoC threshold value with the QoC value of the (subset of) the current state activating the protection context. The latter is calculated from the single QoC values of context assertions by means of a weighted sum. Weights assigned to the different context elements can be defined by the policy manager at policy definition time. In case weights have not been defined, Proteus assigns the same weight to each context assertion.

4 Proteus Middleware Architecture

The Proteus QoC-aware policy framework includes a middleware architecture that supports policy specification, semantic evaluation and enforcement based on current context and QoC conditions. Figure 3 shows the main components of Proteus architecture, namely: the Policy Installation Manager, the Reasoning Core, the Policy Enforcement Manager and the Context Manager. Hereinafter we particularly focus on the Context Manager that is mostly responsible for QoC management in Proteus.

The **Policy Installation Manager** (PIM) is responsible for the setup, configuration and management of the Proteus systems. In particular, PIM provides support to load context and policy ontologies, to install application-specific access control policies, and to define policy QoC constraints.

The **Reasoning Core** (RC) performs reasoning over context and policies to determine currently active policies, according to the QoC-aware policy model described in Section 3. In particular, by exploiting DL-based reasoning, RC determines which protection contexts and policies are active given the current state and its QoC.

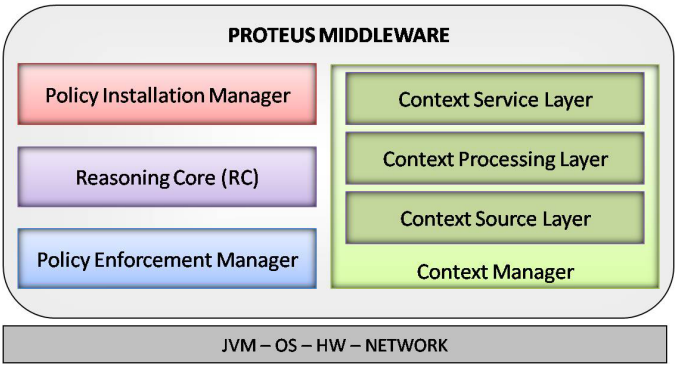

Fig. 3. Proteus Middleware Architecture

The **Policy Enforcement Manager** (PEM) is in charge of enforcing access control policies on protected resources. When a tentative access is performed on a resource controlled by Proteus, such as a file or a remote connection, PEM intercepts it, collects relevant information about the action and interacts with RC to verify whether access should be permitted or prohibited.

The **Context Manager** (CM) collects and manages current state and QoC information from available context sources, and provide them to the Reasoning Core. As shown in Figure 3, CM is designed as a layered component. More in detail:

- The *Context Source Layer* is in charge of interacting with context providers to acquire context data and their associated QoC. Context providers register to CM via the Context Source Layer, which implements provider-specific modules (called Context Sources) to translate context data from the provider's format to CM internal representation. Translation also includes the normalization of different quality parameter values (e.g., freshness and resolution).
- The *Context Processing Layer* acquires, stores, filters and reasons over context assertions by implementing QoC-based filtering, as detailed in the next section.
- The *Context Service Layer* is the mediator between CM and Proteus Reasoning Core. This layer implements a module (a Context Service) for each application that needs to be provided with context information according to defined QoC constraints. The Service Layer allows two different modalities for context/QoC provisioning, namely: (i) the context level agreement (CLA)-based approach establishes QoC requirements that apply to all context assertions provided by CM to RC; (ii) the query-based approach specifies associates QoC constraints to each context query from RC to CM.

4.1 Implementation Details

We have developed a Java prototype implementation of the Proteus middleware architecture. Our deployment setting is a wireless Internet scenario, i.e., a

computing environment where wireless solutions extend the accessibility of the fixed Internet infrastructure via access points, working as bridges between fixed and mobile devices.

For the sake of brevity, we only provide implementation insights about the Context Manager, which is the middleware component that is actually in charge of managing QoC in Proteus. Additional details and experimental results about the prototype can be found at `http://lia.deis.unibo.it/research/Proteus`.

Both the Context Source and Service layers are designed as modular components, thus allowing CM to interact with multiple context providers and consumers via specific modules. At present we have implemented the Proteus Context Service module, which interacts with Proteus RC and PIM, and the Contory Context Source module, which interacts with the context provisioning and management framework Contory [8]. In the current implementation, we support context acquisition by means of a context server. Contory is queried via its SQL-like declarative language.

The Context Processing Layer is composed of four main sub-components, each one addressing a specific functionality, and a context repository storing all available data about the current state.

Query. This unit allows Proteus to install and remove context queries, which are executed by the Proteus Context Service to retrieve context assertions. In particular, it supports three query modalities, namely: *single* query, *time-based* query (executed at fixed intervals), *event-based* query. Queries are encoded in SPARQL and the prototype includes a user-friendly query specification tool for non-expert users.

Contract. This unit allows the Proteus Context Service to define a context provisioning contract defining supported context queries and required QoC constraints. Proteus currently adopts a CLA approach, where QoC constraints are defined at system start up and stored in a configuration file.

Collection. This is the most important unit since it is directly responsible of managing QoC at the context element level. The Context Processing Layer collects data from available context providers via the Context Source Layer, manages their quality and keeps the current state repository up to date. Each context assertion is provided with certain quality attributes values, which have been normalized by the Contory Context Source module. For example, in Table 2, the assertion "Dr.Green is located in the E.R." is provided with freshness = 0.7. The Collection unit exploits these single quality attribute values to determine a global QoC value for each context assertion according to the following scoring function:

$$QoC_{ctx\_assertion} = \sum_{i=0}^{n} w_i * QoC\_attr_i.$$

$0 \leq w_i \leq 1,$
$0 \leq QoC\_attr_i \leq 1$

Weights w_i represent the contribution of different quality attributes to the global QoC value. Their value is set in a configuration file at system installation time. QoC attribute values are provided by sensors, except for freshness, which is calculated by Proteus using appropriate timestamps. Let us note that setting a specific weight to zero means that the corresponding attribute will not be considered when calculating QoC. Conversely, if a quality attribute value is not provided by the context source, the Context Processing Layer sets it to a predetermined value. If the QoC values of a context assertion do not satisfy CLA requirements, that assertion is not stored in the repository.

Reasoning. The Reasoning unit is in charge of managing the Repository by allowing ontology installation and removal, and periodically executing a back-up transfer (currently on file). Ontologies include both concepts describing context (TBox) and context assertions (ABox): while the former generally remains constant unless the application domain is changed, the latter is frequently updated due to changes in the current state of the world. To increase efficiency, the Reasoning unit therefore performs periodical checks on the ABox repository, and removes those context assertions whose QoC is not compliant with the required level, for example because their freshness has decreased over time. It is worth noting that such QoC-based check not only allows to reduce the repository size for improved efficiency, but it also filters out context information that are not reliable enough to support access control decisions. In addition, this unit performs DL-based reasoning to answer context queries and to ensure that the current state knowledge base is consistent. This is important whenever context assertions are added and especially when they are removed (due to their decayed QoC). The current prototype exploits the DL reasoner Pellet (version 1.5)[2], with support for incremental reasoning, accessed via OWL-API and SPARQL queries.

5 Evaluating QoC-Aware Access Control in a Pervasive Scenario

The exploitation of a QoC-aware semantic middleware for access control introduces different forms of overhead, depending on both the deployment environment and the performance of middleware facilities. The most critical aspects for the performance and feasibility of our approach are (i) the introduction of a QoC modeling and evaluation model, and (ii) the exploitation of semantic technologies. In particular, the overhead due to these design choices should be considered at two different levels: when acquiring and managing context information (i.e., at the Context Manager level), and in policy management and evaluation (i.e., at the Reasoning Core level). We hereinafter provide some evaluations about CM performance. Additional implementation insights and evaluations, e.g., about context and policy reasoning in Proteus RC, are available at `http://lia.deis.unibo.it/research/Proteus`.

[2] `http://clarkparsia.com/pellet`

To build a test setting for our evaluations, we considered a mobile healthcare scenario, where access control policies are needed to regulate access to private health information (PHI) of patients, such as their electronic medical record. Example policy, protection context and QoC constraints are represented in Table 1 and 2 according to a concise DL format. For our case study, we developed an application specific ontology, including concepts like Physician, Healthcare center and PHI, to integrate with Proteus policy, context and QoC ontologies.

5.1 Performance Evaluation

Our tests were executed on a AMD Athlon 2800+ processor @2.08GHz, with 1024 MB RAM, running Windows XP SP2, Java SE 1.6.0_03 and Pellet 1.5. We also performed tests in a distributed deployment setting. To avoid a biased evaluation of QoC-related overhead due to variable network conditions, however, we do not consider them in this evaluation.

Context Repository Management. Managing the context Repository requires to add and remove context assertions provided by context sources, to keep the current state KB up to date and QoC-compliant. We have measured the time needed to add/remove a new context assertion in the Repository with the repository dimension growing up. After each context assertion addition or removal, the CM Reasoning unit performs a consistency check. Pellet 1.5 provides support for incremental reasoning (IR), i.e., optimized reasoning when variations in the knowledge base only involve assertions (ABox) and not ontology concepts (TBox). Results show that Pellet incremental reasoning significantly increases efficiency in case of context assertion addition: by enabling IR, addition times tend to keep constant, around 10 ms, with repository size varying from 100 to 25k context assertions. As for removal times, IR does not bring any substantial advantage: for example, with 1k context assertions in the repository, times keep below 5.2 seconds, both with and without IR.

Query. Another critical evaluation regards the Reasoning unit response time to context queries. We particularly focus on queries executed to periodically remove context assertions based on their QoC. We measured the response time to four different types of SPARQL query with increasing complexity: each query basically retrieves context assertions having one/two/three constraints on quality parameters, while the trivial query has no constraints on QoC (thus returning all context assertions in the repository). Below is shown the most complex query:

```
 SELECT ?ca WHERE
{ ?ca context:hasQuality ?q.
?q context:hasCorrectness ?qp1.
?q context:hasPrecision ?qp2.
?q context:hasFreshness ?qp3.
FILTER (?qp1 > 0.5 && ?qp2 > 0.5 && ?qp3 > 0) }
```

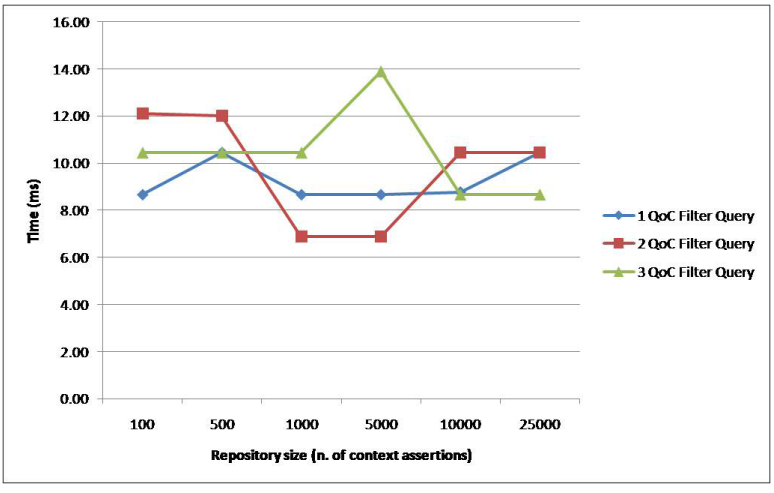

Fig. 4. Average times to answer QoC-based filtering queries

In this test, the repository was initialized with a number of assertions having the same predicate and different subjects, objects and QoC attribute values (randomly determined). As shown in Figure 4, queries with QoC constraints are answered within 10 ms, while response times for the query with no QoC constraints (not shown here) indicate a linear dependence with the repository size. These results show that QoC-based context assertion filtering brings a significant advantage also in terms of performance.

6 Related Work

Several research efforts have addressed the issue of ensuring security in mobile/pervasive environments, particularly access control to shared resources. Considering context as a design principle is a novel research direction with few emerging proposals of context-based policy models, mainly in the field of access control. Quality of context has also been the subject of recent research efforts, which have investigated representation models, calculation techniques and distributed architectures for applying QoC to context provisioning and management/context-aware systems. To the best of our knowledge, however, none of existing solutions considers the issue of modeling, evaluating and assessing the security impact of QoC when used to regulate access to resources. In this section we review some significant research efforts in the fields of context-based security and QoC support for context management systems, respectively.

The security model presented in [9] extends role-based access control by creating a new type of role called *environment role*. Environmental roles, which capture relevant conditions in the current situation, are used to restrict user privileges in accessing resources. Acting as intermediaries between users and permissions they are similar to Proteus protection contexts. The implicit assump-

tion, however, is that environmental roles activation is always reliable and the model does not deal with possibly incorrect or imprecise context information. In addition, no integrated support for environmental role/policy representation at a high level of abstraction and reasoning is provided. The context-sensitive access control Cerberus framework implicitly deals with QoC by supporting authentication with a variable "confidence level": different strengths of authentication are associated with confidence values representing how confident the authentication system is about the identity of the principal [10]. Although the concept of confidence is clearly related to QoC, mainly trustworthiness and correctness, Cerberus does not provide any explicit modeling support for QoC and only deals with quality for authentication mechanisms. Several other context-aware access control solutions for distributed/pervasive environments have recently emerged, such as [11] and [12], but they do not support QoC management not take QoC security impact into account when controlling access to resources.

The term "quality of context" was firstly introduced in work by Buchholz, which provided the original concept and a set of base QoC attributes [4]. Various research works have defined since then QoC abstraction and representation models, mostly referring to a similar set of quality attributes, such as the ones described in [5]. Proteus support the representation of all significant attributes, but also allows the application developer to personalize the set of needed attributes. Some approaches provide ontology models to represent quality parameters, either modeled with logic predicates [10], or with DL-based ontologies [13, 14]. With respect to these ontologies, the Proteus QoC ontology has the advantage of being built on RDF reification model, thus simplifying representation and reasoning since based on the very structure of adopted semantic languages.

As far as QoC values calculation is concerned, very few existing systems actually provide applicable methods to numerically determine quality attribute values: [7], for example, represents a promising direction as it describes functions to concretely calculate accuracy and correctness values.

7 Conclusions and Future Work

Context-aware access control solutions, which exploit context-awareness to control access to resources based on context visibility and changes, should take into account the security impact deriving from the quality of context information used to regulate access decisions. In this paper we presented Proteus, a QoC-aware access control framework for sharing resources in pervasive environments. Proteus exploits QoC and semantic technologies to discard context data with insufficient quality, and to select applicable policies based not only on current context, but also on its quality. This helps minimizing the risk of granting access to resources based on incorrect or ambiguous context information, while reducing the overhead due to policy management.

First evaluations on Proteus prototype middleware show encouraging results. We are currently working on an optimized context data storage model to make

context assertion removal faster and more efficient. We are also testing performances with the new support for incremental reasoning provided with the Pellet 2.0. Finally, we are planning to extend the current CM implementation with additional context source modules to support interaction with different context provisioning systems.

References

1. Toninelli, A., et al.: A semantic context-aware access control framework for secure collaborations in pervasive computing environments. In: ISWC, pp. 473–486 (2006)
2. Dey, A.K.: Understanding and using context. Personal and Ubiquitous Computing 5(1), 4–7 (2001)
3. Sandhu, R.S., et al.: Role-based access control models. IEEE Computer 29(2), 38–47 (1996)
4. Buchholz, T., Kupper, A., Schiffer, M.: Quality of context: What it is and why we need it. In: HPOVUA 2003 (2003)
5. van Sinderen, M., et al.: Supporting context-aware mobile applications: an infrastructure approach. Communications Magazine 44(9), 96–104 (2006)
6. Lassila, O., Khushraj, D.: Contextualizing applications via semantic middleware. In: MOBIQUITOUS 2005, pp. 183–191. IEEE Computer Society, Washington (2005)
7. Kim, Y., Lee, K.: A quality measurement method of context information in ubiquitous environments. In: ICHIT 2006, vol. 2, pp. 576–581 (November 2006)
8. Riva, O.: Contory: A middleware for the provisioning of context information on smart phones. In: Middleware, pp. 219–239 (2006)
9. Covington, M.J., et al.: Securing context-aware applications using environment roles. In: SACMAT 2001, pp. 10–20. ACM, New York (2001)
10. Al-Muhtadi, J., et al.: Cerberus: a context-aware security scheme for smart spaces. In: PerCom 2003, pp. 489–496 (March 2003)
11. Dersingh, A., Liscano, R., Jost, A.: Utilizing semantic knowledge for access control in pervasive and ubiquitous systems. In: WIMOB 2008, pp. 435–441 (October 2008)
12. Lachmund, S., et al.: Context-aware access control; making access control decisions based on context information. In: Mobiquitous 2006, pp. 1–8 (July 2006)
13. Tang, S., Yang, J., Wu, Z.: A context quality model for ubiquitous applications. In: IFIP NPC Workshops, pp. 282–287 (September 2007)
14. Bu, Y., et al.: Managing quality of context in pervasive computing. In: QSIC 2006, pp. 193–200 (October 2006)