A Context-centric Security Middleware for Service Provisioning in Pervasive Computing

Antonio Corradi, Rebecca Montanari, Daniela Tibaldi, Alessandra Toninelli

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

Abstract

Pervasive user mobility, wireless connectivity and the widespread diffusion of portable devices raise new challenges for ubiquitous service provisioning. An emerging architecture solution in the wireless Internet is based on mobile proxies (implemented as mobile agent-based middleware components) over the fixed network that follow the movements and act on behalf of the limited wireless clients. It is crucial that mobile proxies have full visibility of their context, i.e., the set of available and relevant resources, depending on access control rules, client location, user preferences, privacy requirements, terminal characteristics, and current state of hosting environments. The paper presents the design and implementation of a context-centric security middleware, called UbiCOSM, for MA-based service provisioning in pervasive computing. UbiCOSM dynamically determines the contexts of mobile proxies, and effectively rules the access to them, by taking into account different types of metadata (user profiles and authorization policies), expressed at a high level of abstraction and cleanly separated from the service logic. The paper also shows the functioning of UbiCOSM in the design and the development of a mobile context-centric airport business assistant.

1. Introduction

The widespread availability of wireless network connectivity in the environments where users live and work together with the increasing diffusion of portable devices creates novel opportunities for users to access services anywhere, at any time and from various access devices. In particular, the recent proliferation of heterogeneous portable devices and of different technologies for wireless connectivity in home/office environments suggests not only to extend to mobile users/devices the access to traditional Internet services, designed and implemented for the fixed network infrastructure, but also to develop new classes of services that can provide results that depend on the relative position of clients and on the consequent resource visibility (*context-aware services*).

However, the design and deployment of ubiquitous services impose new challenges to the retrieval and operation on distributed resources, undermining several assumptions of traditional service provisioning solutions. Whereas traditional service provisioning relies on a static characterization of context operating conditions where changes in the set of both service clients (users/devices) and accessible resources are relatively small, rare, or predictable, user/device mobility causes frequent changes in physical user location and in consequently available resources. As users roam across different network localities, they have different resource visibility, depending on their location and other context-dependent information, such as device characteristics, local security policies and resource state.

The complexity of the above scenario calls for novel middleware solutions for facilitating context-aware service development and for supporting service delivery to wireless devices in mobility environments. We claim that this novel middleware should exhibit the non-traditional property of context-awareness and should enable the dynamic installation/migration of client-side middleware/service components and their seamless discarding when no longer needed. An emerging guideline is to have active middleware components that are deployed at service provision time to act as user/device proxies over the fixed network and to carry on the needed service configuration, tailoring and management. The relevance of implementing middleware components as mobile proxies that follow client movements and execute in their same network locality starts to be recognized [4], [5]. We claim

the suitability of designing and implementing mobile proxies for the wireless Internet in terms of Mobile Agents (MAs). MA-based proxies can carry on service requests autonomously even in case of temporary device disconnection, can migrate among different network localities by maintaining the session state, and can exploit the full context visibility typical of the MA programming paradigm to support context-based service configuration and tailoring [6].

However, the deployment of MA-based proxies also raises novel and challenging security concerns [7], [8], [9]. On the one hand, the possible injection of malicious proxies can compromise the security of the wireless Internet fixed nodes similarly to the case of viruses and worms. On the other hand, malicious nodes may try to disclose the private information carried by the hosted proxies and to tamper with the MA code and state.

The paper focuses on the novel access control issues stemming from the adoption of MA-based proxies for the support of context-dependent service provisioning over the wireless Internet. Several practical techniques have been already proposed to control and confine the interactions between MAs and hosting execution environments. Type-safe languages permit to determine whether incoming MAs respect safety properties, such as address space confinement. Sandboxing techniques have been used to rigidly limit resource visibility and access scope of MAs while executing and have evolved to propose more advanced access control solutions [9]. However, these solutions are not flexible enough for the addressed highly dynamic pervasive environments: they typically evaluate permissions depending on the identity/role of the client requesting access to resources. The new ubiquitous scenario makes service providers also deliver services to unknown entities and, more important, whose identity may be un-informative or not sufficiently trustworthy. In fact, it is almost impossible to know in advance the identities/roles of all subjects that are likely to request access to their managed resources/services. Instead, service providers can more easily define the conditions for making resources available and for allowing/denying users resource visibility and access according to their context operating conditions.

The paper presents the design and implementation of a security middleware, called UbiCOSM (**U**biquitous **CO**ntext-based **S**ecurity **M**iddleware), for context-centric access control in the wireless Internet. UbiCOSM dynamically determines the contexts of MA-based proxies, and effectively rules the access to resources, by taking into account different types of metadata (user/device/resource profiles and authorization policies), expressed at a high level of abstraction and cleanly separated from the service logic. UbiCOSM provides an integrated environment for both the specification of metadata and for their runtime enforcement. The proposed access control middleware integrates with the CARMEN framework which offers the support facilities for user/terminal mobility and for the proxy-based discovery/binding of wireless devices to the needed resources in their contexts [6].

The paper finally presents the case study of a context-centric airport business assistant service prototype, built on top of UbiCOSM and deployed over a group of coordinated IEEE 802.11 localities, to evaluate the usability and effectiveness of the proposed middleware.

2. UbiCOSM Security Framework

UbiCOSM is an access control middleware for securing agent-to-agent and agent-to-environment interactions in service provisioning scenarios with context awareness requirements. In particular, UbiCOSM focuses on three main peculiar aspects: context-centric access control, active context view provisioning to middleware components and support for disclosure of the security properties of UbiCOSM agents and resources to interested entities. UbiCOSM access control decisions depend on dynamic context attributes, such as resource state and availability, in addition to more traditional attributes, e.g., the identity of the MA code implementer, or the identity/role of the principal on behalf of whom MAs are executing.

Another distinctive feature of UbiCOSM is to provide MAs entering a new locality with a controlled visibility of the directly accessible physical/logical resources and of the other MAs locally executing (*active context views*). Active context views contain resources that both MAs are willing to access and the UbiCOSM access control function have qualified as accessible. The provision of active context views to MAs has many benefits. MAs can exploit the visibility of available resources to adjust their behaviour accordingly and to reduce the risk of undesired task failure during execution. Active context views can also help MAs to decide whether it is more profitable to stay in a locality than to move from it and to explore a new computing environment.

UbiCOSM addresses also the security issues that arise in dynamic environments where different actors sharing little or no prior knowledge about each other have to interact in order to achieve some kind of result. When invoking a service or choosing an entity to interact with, it is crucial to know its security attributes and policies. The visibility of security properties allows to reason about the security effects of enabling a proxy to invoke a service or to interact with another proxy. Security property disclosure

is also important to facilitate service/interaction agreement negotiation between the interacting parties. UbiCOSM enables users and service providers to specify their security characteristics in terms of both what they are capable to do and what they require from other subjects to do. UbiCOSM facilities are then in charge of translating these specifications into a system-compliant form and of storing them.

2.1 Security Model

To support context-centric access control and interaction based on disclosure of security properties, UbiCOSM allows system administrators and final users (on behalf of whom MAs act over the network) to specify their functional and security characteristics at a high level of abstraction in terms of metadata. Metadata are declarative rules that describe the attributes of users, devices and service components and the desired access control requirements.

A primary advantage of exploiting metadata is the possibility to separate security logic from security control. Metadata govern access control decisions, but are decoupled from the implementation of the system components in charge of enforcing access control accordingly to the metadata specifications. This favors the rapid prototyping of secure MA-based services, their runtime configuration and maintenance. By changing metadata to accommodate evolving security requirements, the behaviour of a running agent system can be dynamically and rapidly varied without any intervention on agent and system code. In addition, developers do not need to manually insert calls to security checking code inside each resource that a host may want to protect from illegitimate agent usage. Metadata can also facilitate automated security reasoning: all basic elements involved in access control decisions can be easily extracted from declarative notations, analyzed and checked for conflicts. The relevance of metadata adoption to decouple management logic from mechanism implementation details has been recently recognized in network, systems and service management [10]. UbiCOSM supports two kinds of metadata: *profiles* and *policies*. Profiles generally represent the synthetic description of a subject and have been wide adopted in the context of mobile applications (e.g. W3C Composite Capability/Preference Profiles). Profiles facilitate interoperability: when an entity (a *user*, a *device* or a *resource/service)* joins the system, it presents its profile to provide other entities it may wish to interact with an explicit description of its characteristics. This allows an entity to predict (part of) the exhibited behaviour of another possibly unknown entity.

Profiles are composed of *Capabilities* and *Preferences*. Capabilities include all the information needed to qualify an entity in terms of what it is capable of, both from a functional and from a security-related perspective. For instance, the Capabilities profile part of a service describe the operations it can perform, as well as the security mechanisms it is able to enact (security capabilities). Note that, in case of a user profile, the described capabilities actually refer to the mobile proxy acting on the user behalf.

Preferences are used to express the desired context visibility of an entity, i.e., the kind of resources it is looking for, their expected properties and also the security features it requires from the entities it wishes to interact with. For example, User Preferences specify which kind of resources the user is willing to use as well as which other users she wishes to see, and her requirements in terms of security solutions.

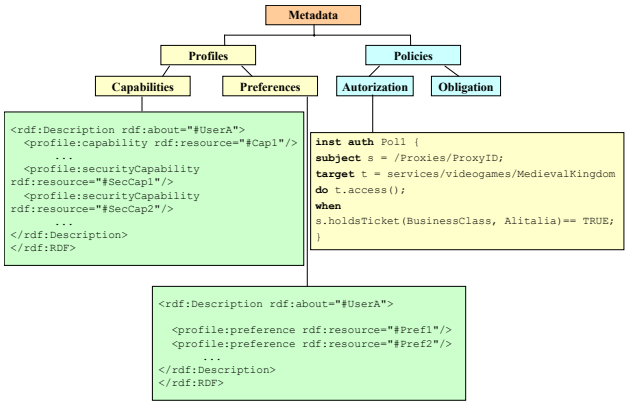

Figure 1. UbiCOSM metadata taxonomy.

UbiCOSM adopts RDF-compliant formats for profile representation to deal with the heterogeneity of data representation over different architectures, i.e., the Ontology Web Language (OWL) for the interoperable description of user/device/resource profiles [10]. The adoption of a semantic language, such as OWL, provides a significant expressivity enhancement in profile specification. Semantic languages allow machines to learn and reason about components' properties, instead of simply performing syntactic comparisons between words and expressions. Moreover, it provides more flexibility to the system, since it enables different levels of compatibility between components. For instance, if a service profile declares a P1 property stating "XML-based Encryption" as a security capability and a user specifies the P2 property "Encryption" as a security requirement, semantic reasoning enables to recognize that P1 can fulfil P2 because the concept of P1 is included in the concept of P2.

```
<?xml version='1.0'?>
<!--OWL Language-->
<rdf:RDF>

<cesa:CESAUser rdf:ID="Bob">
</cesa:CESAUser>

<action:ServiceRequestAction rdf:ID="Cap1">
       <action:target rdf:resource="&cesa;CESAEntertainmentService"/>
</action:ServiceRequestAction>

<action:MobilityAction rdf:ID="Cap2">
</action:MobilityAction>
       ...
<airport:Ticket rdf:ID="Cap3">
       <ticket:Company rdf:resource="&ticket;Alitalia"/>
       <ticket:Class rdf:resource="&ticket;BusinessClass"/>
</airport:Ticket>
       ...
<preference:LocalityPreference rdf:ID="Pref1">
   <preference:locality rdf:resource="&airport;Terminal"/>
   <preference:sameLocalityAs rdf:resource="#Bob"/>
</preference:LocalityPreference>

<preference:ResourcePreference rdf:ID="Pref2">
       <preference:target rdf:resource="&entertainement;Videogame"/>
</preference:ResourcePreference>
       ...

<rdf:Description rdf:about="#Bob">
       <profile:capability rdf:resource="#Cap1"/>
       ...
       <profile:securityCapability rdf:resource="#SecCap2"/>
       ...
       <profile:preference rdf:resource="#Pref1"/>
       ...
</rdf:Description>
</rdf:RDF>
```

Figure 2. UbiCOSM user profile

Policies are high-level directives regulating resource access control and defining choices in security management operations. UbiCOSM distinguishes between *authorization* and *obligation policies*. Authorization policies rule access control by defining what a subject can or cannot do on specific target resources if certain context conditions are met. These conditions may depend on the runtime resource/system state and on the subject identity/role. Obligation policies allow to automate security management tasks by specifying when the system must perform a specific management action on a set of target objects. For instance, MA system administrators can exploit obligation policies to block the execution of an agent when its CPU consumption is higher than a tolerated threshold. In the following, we focus only on authorization policies, essential for UbiCOSM context-centric access control model, whereas for details about obligation policies please refer to [11].

Each system entity (user, device or resource) can specify its own policies. It is possible to create individual-specific policies, in order to protect a component against malicious code or to determine its behaviour without any static control. Other policies may apply to the system as a whole, e.g. policies concerning performance issues, such as load balancing and availability.

UbiCOSM exploits the Ponder language for expressing both types of authorization policies [11]. In Figure 1 `Pol1` is an example of system access control policy that allows an entering MA (the `subject` clause) to access the Videogame service (the `target` clause) depending on the current context conditions (the `when` clause). In particular, the entering MA can access the service if the user it is acting on behalf of holds a "Business Class" ticket for an Alitalia flight. Note that a subject can operate on target objects, by only invoking methods visible on the target interface and that the `when` clause allows to limit the applicability of authorization policies on the basis of context conditions.

UbiCOSM exploits all the described metadata information to take access control decisions, to determine the active context view to return to incoming MAs, and to perform a security property matching procedure between two system components. In particular, a user/software component can exploit the UbiCOSM metadata support to verify if the declared capabilities of one entity fulfil her requirements.

3. UbiCOSM Middleware

UbiCOSM has been built on top of the Java-based CARMEN system that supports the accessibility of mobile users/terminals to both traditional Web and to new context-dependent services [6]. CARMEN is centered on the distributed deployment of active middleware proxies over the fixed network to support service provisioning to portable devices. CARMEN provides any portable device with a companion middleware proxy (*shadow* proxy) that autonomously acts on its behalf, possibly negotiates service tailoring to fit user/device characteristics and follows user/device movements among network localities. CARMEN implements shadow proxies by exploiting the MA programming paradigm. In particular, CARMEN provides proxies with execution environments, called places, that typically model nodes. Places can be grouped into domains that correspond to network localities, e.g., either Ethernet-based LANs or IEEE 802.11b-based wireless LANs. With a finer degree of detail, a shadow proxy is implemented by one CARMEN agent running on a place in the domain where the portable device is currently located. CARMEN domains can facilitate policy evaluation and enforcement for context-centric access control. In fact, the domain abstraction allows to define a well-specified management boundary: each domain holds references to the entities currently members of the domain (both MAs and resources) and to the applicable metadata. In particular, profiles and individual policies are maintained in the CARMEN directory service with global visibility and are accessible via the local domain directory

component [6]; system policies are stored locally at the domains where they have to be enforced to increase management decentralization and local policy access.

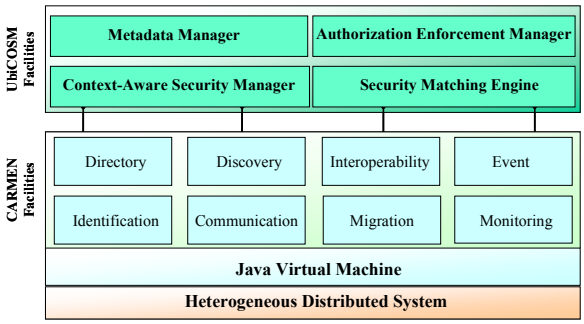

Figure 3. UbiCOSM middleware facilities

At first proxy instantiation, UbiCOSM creates a secure trust relationship between the device and the proxy according to the protocol steps described in [12]. At run-time, proxy interactions with the hosting environments are protected by means of the UbiCOSM middleware facilities shown in Figure 3. The **Metadata Manager** enables users to specify the needed metadata. The **Authorization Enforcement Manager** is responsible for enacting authorization policies. The **Context-aware Security Manager** returns to proxies active context views on the basis of specified metadata and the **Security Matching Engine** performs a matching between desired and expressed security properties of system components. UbiCOSM facilities exploit CARMEN lower-level functions for proxy identification, resource discovery, directory, context monitoring, and event registration/dispatching [6].

Metadata Manager (MM). MM provides various tools for metadata editing, updating, removing, and browsing. In particular, with regards to policies, MM integrates tools for syntactic analysis of policy specifications and for transforming both system authorization and obligation policies into Java objects, which act as policy information containers to be interpreted at runtime [13]. MM also provides a semantic tool for parsing OWL profiles and translating them into ontological concepts which are represented as Java objects.

Moreover, MM is in charge of distributing specified metadata (both in the high-level and low-level formats) to the CARMEN directory component responsible for storage, i.e., the one in the domain of first user registration (user home domain). At its first instantiation, any shadow proxy exploits the CARMEN directory to retrieve its profile and individual policies, which become part of its carried state.

Context-Aware Security Manager (CASM). CASM is responsible for establishing the active context view of any entering MA. When a new shadow proxy enters a domain, CASM calculates and returns to the proxy a valid active context view on the basis of the proxy responsible user profile's preferences, of the active system access control policies and of the individual policies imposed by the other proxies currently executing in the domain. The active context view sent to the proxy is a copy of the active view maintained by CASM for any proxy in the locality, until the proxy exits from the domain.

CASM is also in charge of maintaining active context views up-to-date when relevant variations in context information occur, such as changes in the MAs executing in a locality, in resource availability, or in individual policies specifications. CASM allows service providers to choose among differentiated view update strategies, ranging from an *eager* strategy, which requires to update active context views at the occurrence of any relevant context change, to a *lazy* strategy, consisting in updating the active context view only on user-demand. The choice of the most appropriate strategy to adopt depends on several factors, such as service requirements, trade-off between tailoring/configuration optimal choices and context update overhead. Similar considerations guided the implementation of protection domains in the JDK 1.2 security architecture [9].

Authorization Enforcement Manager (AEM). AEM mediates proxy-resource interactions by granting/denying proxies the access to resources, possibly depending on runtime conditions. Shadow proxies cannot directly access the resources included in their active context view, but have to interface with AEM at any resource access request. Active context views provide proxies with only the identifiers of accessible resources along with the permitted actions; no direct handles to the resources listed in the active view are returned to proxies.

When a proxy requests to access a resource, AEM intercepts the request and evaluates whether to deny/accept it. The type of access control checks needed depends on the update strategy adopted by CASM. In the case CASM adopts an eager update strategy, AEM grants/denies the proxy request by simply checking whether the requested resource and action are included in the already updated active context view maintained by CASM. On the contrary, in the case of lazy update strategy, AEM has to calculate the set of permissions that currently applies to the requesting proxy.

Security Matching Engine (SME). SME is in charge of performing a security-related matching on behalf of the requesting user and to verify if one component's security properties fulfil those requirements, according to the desired level of accuracy. First of all, SME exploits CARMEN directory facilities to retrieve stored metadata about both the requester entity and the security profile of the selected component. Let us note that the requester entity (user, service or device) has previously selected the component from all entities currently included in the active context view on the basis of its functional properties and capabilities. Finally, SME executes a matching algorithm to verify if a compatibility exists between required and declared security properties. Depending on the options expressed by the requester, SME returns successfully if all the requirements or at least one or a specific one have been satisfied.

SME is able to perform both a *syntactic* and a *semantic* matching. Syntactic matching verifies whether security capabilities are expressed exactly in the same way as required properties. This means that the security properties of the requester and the security capabilities of the possibly matching component must be described using the same syntax and vocabulary. This matching technique is particularly suitable for environments where a common syntax exists and interacting parties share an agreement about the meaning of the adopted vocabulary.

Semantic matching enhances this technique by adding support to ontology definition and reasoning. In order to determine whether the security capabilities of an entity are compatible with the expressed requirements, SME retrieves the ontological concepts created by MM while parsing profiles and then reasons about them. It must be noted that reasoning about ontologies could be a very consuming task in terms of computational power and memory usage: therefore, this matching modality may not fit resource-constrained devices.

4. Case Study

We have tested UbiCOSM in the design and implementation of a Context-centric Entertainment Service Assistant (CESA) that allows mobile users to find available entertainment services at each airport terminal. CESA exploits the visibility of user location to retrieve only those services which are situated at the airport terminal where the user is currently located. Moreover, CESA ensures that only passengers with required context-related capabilities, such as flying with a company, or being a business or economy class passenger, can access the services. In addition, CESA

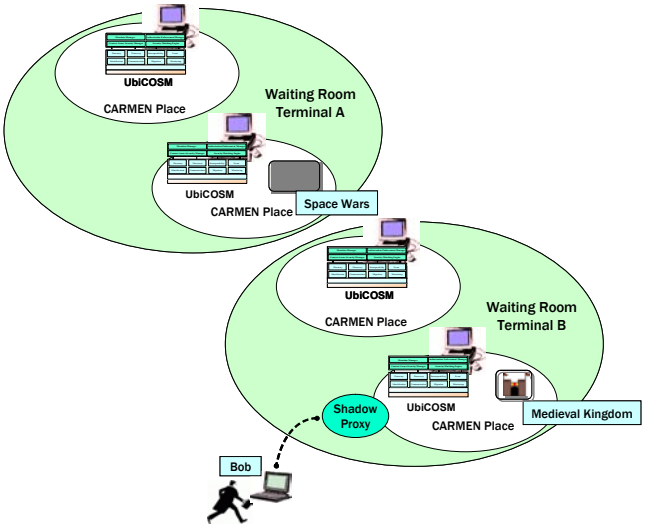

Figure 4. CESA scenario.

enables users to select the services that fulfill user security requirements.

Our simulated airport testbed setting for CESA consists of a wireless local network composed by several 802.11 network localities, with each locality modeled as a CARMEN domain (see Figure 4). Each domain provides execution environments (*places*) for shadow proxies on each physical node, offers UbiCOSM middleware facilities on each place and hosts info service components providing information about the terminal locally available resources.

CESA users interact with the UbiCOSM infrastructure via device-specific clients running on their wireless access devices (Toshiba e740 Pocket PCs with Wi-Fi connectivity).

The clients allow users to subscribe to CESA, by filling in a form with user profile and security policy information and to authenticate themselves to the service before starting any CESA session. After successful authentication, the user interacts with the MM service component in order to specify her profile, e.g., her preferences and security requirements. CESA instantiates a shadow proxy in the domain where the user is currently attached, and the proxy loads the user profile in its state part. At service provision time, the clients are only in charge of forwarding user requests (and of visualizing the received service results) to (from) their responsible proxies.

Let us consider the case of a flying company, Alitalia, that decides to offer to its "business class" passengers waiting for their flight departure some entertainment services, such as video on demand, music download and

videogames. Service are available at different terminals, for instance videogames at Terminal A and video on demand at Terminal B.

Now let us consider a videogame named "Medieval Kingdom" which is offered by the flying company to business class passengers boarding at Terminal B. The videogame is represented as a UbiCOSM service and is associated with a profile and a set of policies. Figure 4 and Figure 1 show an extract of the videogame profile and policies set, respectively. The authorization policy Pol1 states that "Medieval Kingdom" game is accessible only by Alitalia "business class" passengers. Thus, other travellers currently waiting to board at Terminal B are not allowed to access the game.

In addition, two different configuration settings are available for the videogame execution. It is possible to install some plug-ins on- board on the client side or not to install them. The game profile specifies the available configuration settings as a security property.

Suppose now that a traveler, Bob, holding a business class ticket for a Alitalia flight enters Terminal B. Figure 2 shows an excerpt of the traveler's profile. As specified in the profile Preferences part, the user is willing to have resource visibility within the terminal area he is currently located and, in particular, he is looking for a videogame service. Moreover, he holds a "business class" ticket of Alitalia flying company.

When the proxy, acting on Bob behalf, enters the network locality, CASM retrieves and interprets Bob profile and the policies to apply to his proxy. CASM evaluates these policies by coordinating with the CARMEN monitoring facilities. On the basis of profile/policy metadata and of context conditions currently holding in the system, CASM generates Bob active context view. The view lists the services that are situated in the same terminal area where the user is currently located, namely at Terminal B, and that satisfy Alitalia security requirements. For example, if an "economy class" passenger who has subscribed to CESA enters Terminal B, she cannot see in his Active Context View the "Medieval Kingdom" videogame, because she does not hold a "business class" ticket. In fact, CESA ensures that videogame services are available to only business class passengers.

Note that the user profile preference is not bounded to a specific terminal, but simply refers to "the terminal where the user is currently located". CASM automatically translates this specification into a precise location information, by retrieving user current location via coordination with the underlying CARMEN monitoring facility. For example, if a videogame ("Space Wars") is situated in the waiting area at Terminal A, the game is not visible to the user while the user resides at Terminal B. If the user moves to Terminal A, CASM updates the active context view by adding "Space Wars" and removing "Medieval Kingdom" from the list of available games, provided that both services are accessible to the user.

```
<?xml version='1.0'?>
<!--OWL Language-->
        ...
<rdf:RDF>

<cesa:CESAEntertainmentService rdf:ID="Medieval Kingdom">
    <cesa:typeOfService rdf:resource="&entertainment;Videogame"/>
    <cesa:serviceOfferLocation rdf:resource="&airport;TerminalB"/>
</cesa:CESAEntertainmentService>

<owl:ObjectProperty rdf:ID="PlugInConfigProperty">
    <rdfs:subPropertyOf rdf:resource="&service;ServiceConfigProperty"/>
    <rdfs:range rdf:resource="&service;PlugIn"/>
</owl:ObjectProperty>
        ...

<preference:DataPreference rdf:ID="Pref1">
    <preference:dataFormat rdf:resource="&data;Binary"/>
</preference:DataPreference>
        ...

<rdf:Description rdf:about="#MedievalKingdom">
    <profile:capability rdf:resource="#Cap1"/>
        ...
    <profile:securityCapability rdf:resource="#PlugInConfigProperty"/>
        ...
    <profile:preference rdf:resource="#Pref1"/>
        ...
</rdf:Description>
</rdf:RDF>
```

Figure 4. Entertainment service profile.

Suppose now that Bob decides to play a videogame on his laptop while waiting for his flight to board. He does not want to have external plug-ins installed on his laptop, to avoid the risk of security breach. In the testbed, Bob selects the videogame service "Medieval Kingdom" from the list of available resources in his active context view. Then, he asks the UbiCOSM middleware to verify if the selected service actually fulfils his security requirements, i.e., if "Medieval Kingdom" game does not install any plug-in module on the client side. The user responsible proxy exploits the MM semantic tool to translate the profile-compliant query into an ontological information, and forwards the ontological request to SME, which adopts as default matching modality the semantic one. Depending on application requirements, UbiCOSM users can also command SME to exploit only a syntactic matching. Then, SME retrieves from the UbiCOSM repository all the Java objects created by MM to represent the ontological concepts related to "Medieval Kingdom" and Passenger profiles. Finally, it performs the matching algorithm between the videogame service profile and the security requirements expressed by the user. In particular, SME parses the "Medieval Kingdom" profile, takes each single security capability and semantically compares it with the

user requirement. This means that SME exploits its knowledge base and its reasoning engine to determine if the capability can fulfil the requirement. In our case, SME finds out that "Medieval Kingdom" can execute either by installing some plug-ins on the client side or by not installing them. On the other side, Bob has required a service which does not install any plug-in on his laptop. It is worth stating that these two features do not match syntactically. Nevertheless, they are compatible from a semantic point of view: if "Medieval Kingdom" supports both execution settings, with and without plug-ins, it certainly supports each of them and specifically the second one. So, SME returns successfully reporting the security property exhibited by "Medieval Kingdom".

UbiCOSM introduces different forms of overhead depending on the performance of the different UbiCOSM middleware functions involved, from the access control checks to semantic matching actions.

During the testing of the CESA prototype, we have carried out several measurements to give an evaluation of the overhead induced by UbiCOSM services. In particular, CASM and SME introduce the main performance penalties. Thus we here specifically focus on the evaluation of their overhead.

In particular, with regard to CASM we have compared the overhead introduced by the lazy and the eager update strategies. We have verified that the adoption of an eager strategy introduces a performance penalty of one magnitude more than the adoption of a lazy one. Let us note that the penalty is due to the fact that in the case of the lazy strategy CASM has to calculate the current active context view at any user resource access request time.

With regard to the SME overhead, it is worth stating that semantic matching represents the main overhead impact factor in terms of both memory and computational consumption. This overhead, however, is counterbalanced by the increased interoperability between system components and by the possibility to support security attribute disclosure and reasoning.

5. Conclusions and Ongoing Work

Effective service provisioning over the wireless Internet requires the full visibility and the flexible management of context information. Requirements for context visibility start to be recognized in the security area also for traditional fixed networks, where interesting novel proposals are emerging to enhance protection techniques with context awareness [19], [20]. By focusing on mobile environments, UbiCOSM proposes and implements a novel security model for context-centric access control that exploits different types of metadata to express articulated security strategies, separately from the application logic. This separation of concerns increases flexibility, dynamicity and reusability of middleware/service components. In addition, the choice of the most proper evaluation strategy allows administrators to tune the access control overhead, which is acceptable for most wireless Internet services, usually having no hard real-time constraints.

First experiences in implementing services on top of UbiCOSM are encouraging further research to extend the middleware prototype. We are working on how to deal with possible runtime conflicts among policies to be enforced, and we are investigating and prototyping solutions based on policy prioritization. Finally, we are integrating UbiCOSM with mechanisms for inter-cell mobility prediction based on IEEE 802.11 signal strength variations, in order to anticipate the migration of (copies of) shadow proxies towards the new locality and the determination of the new active views in advance.

References

1. A.K. Dey, G.D. Abowd, "Towards a Better Understanding of Context and Context-Awareness", *Proc. of CHI*, The Hague, The Netherlands, April 2000.
2. T. Rodden, K. Cheverst, K. Davies, A. Dix, "Exploiting Context in HCI Design for Mobile Systems", *Proc. of Workshop on Human Computer Interaction with Mobile Devices,* Scotland, May 1998.
3. P. Bellavista, A. Corradi, R. Montanari, C. Stefanelli, "Dynamic Binding in Mobile Applications: a Middleware Approach", *IEEE Internet Computing*, Special Issue on "Mobile Applications", Vol. 7, No. 2, March/April 2003.
4. P. Bellavista, A. Corradi, C. Stefanelli, "The Ubiquitous Provisioning of Internet Services to Portable Devices", *IEEE Pervasive Computing*, Vol. 1, No. 3, July-September 2002.
5. IKV++ Technologies AG, enago Open Service Platform, http://www.ikv.de
6. P. Bellavista, A. Corradi, R. Montanari, C. Stefanelli, "Context-Aware Middleware for Resource Management in the Wireless Internet", *IEEE Transactions on Software Engineering*, Vol. 29, No. 12, December 2003.D.
7. R. Montanari, C. Stefanelli, N. Dulay, "Flexible Security Policies for Mobile Agents Systems", *Microprocessors and Microsystems*, Elsevier Science, Amsterdam, Olanda, Vol. 25, No. 2, 2001.
8. N. Mitrovic, U. Arronategui Arribalzaga, "Mobile Agent security using Proxy-agents and Trusted domains", *Proc. of SEMAS 2002*, Bologna, Italy, July 2002.
9. L. Gong, "Inside Java 2 Platform Security", Addison Wesley, 1999.
10. L. McGuinness, F. van Harmelen, OWL Web Ontology Language Overview, W3C Recommendation 10 February 2004.

11. N. Damianou, N. Dulay, E. Lupu, M. Sloman, “The Ponder Policy Specification Language”, *Proc. of Policy 2001*, Springer-Verlag, LNCS 1995, pp. 18-39, Bristol, January 2001.
12. M. Winslett, T. Yu, K. E. Seamons, A. Hess, J. Jacobson, R. Jarvis, B. Smith, L. Yu, “Negotiating Trust on the Web”, *IEEE Internet Computing*, Vol. 6, No. 6, November/December 2002.
13. A. Corradi, N. Dulay, R. Montanari, C. Stefanelli, “Policy-driven Management of Mobile Agent Systems”, *Proc. of Policy 2001*, Springer-Verlag, LNCS 1995, Bristol, January 2001.