Ubiquitous Service Regulation Based on Dynamic Rules

José Viterbo F.
Department of Informatics
PUC-Rio
R. Mq. de S. Vicente, 225
22453-900, Brasil
viterbo@inf.puc-rio.br

Markus Endler
Department of Informatics
PUC-Rio
R. Mq. de S. Vicente, 225
22453-900, Brasil
endler@inf.puc-rio.br

Jean-Pierre Briot
LIP6
Université Paris 6
8 rue du Capitaine Scott
75015 Paris, France
briot@poleia.lip6.fr

Abstract

Ubiquitous computing systems may be seen as typical open systems where due to the mobility of users and their devices, heterogeneous and previously unknown entities may come to interact spontaneously. In this highly dynamic and heterogeneous scenario, applications must be capable of accessing the appropriate instances of the required services in each visited network or region. Some services, however, are available to be used only by users and applications that fulfill some conditions. In ubiquitous systems the interaction between client and server applications happens in a physical space, involving human and artificial agents that act under social and administrative rules. We propose the integration of context-awareness with a social regulation approach to control the interaction of mobile applications in such environments. This has the advantage that access policies and rights can be defined and monitored independently of the applications for the ubiquitous system.

1 Introduction

In the vision of ubiquitous computing, computer systems will seamlessly be incorporated into our everyday lives, providing services and information anytime and anywhere [19]. Compared to traditional distributed systems, ubiquitous computing systems feature increased dynamism and heterogeneity [14]. The underlying ubiquitous computing infrastructures are more complex and bring into the foreground issues such as user mobility, device disconnections, join and leave of devices, heterogeneous networks, as well as the need to integrate the physical environment with the computing infrastructure [6].

A fundamental characteristic of a software infrastructure for ubiquitous applications is context-awareness, i.e., the capability of providing services based not only on user inputs, but also on implicit contextual information probed (and deduced) from a wide range of distributed and heterogeneous sensors. While the standard context-aware approach to design and implement ubiquitous systems usually focuses only on two dimensions, the topological space (e.g., the modeling of rooms, locations) and on the availability of resources (e.g., battery charge or network bandwidth), our approach extends the support for ubiquitous computing so as to take into account the social context, by representating and managing social norms [17].

As a sort of open system, a ubiquitous system may contain dynamically interacting entities engaged in complex coordination protocols. In such kind of systems, entities, human or artificial ones, interact with each other, either to cooperate or to compete. Due to the mobility of users and their devices, previously unknown entities of different sorts, using specific devices may come to interact spontaneously in different environments. Some ubiquitous applications may have as primary purpose to execute a given task using the support of available resources or services in the user's vicinity or in the currently visited network domain [1]. Nevertheless, some services should be made available only to a restricted set of users, under specific conditions. For example, services provided by active spaces such as smart houses, offices or classrooms, are meant to be used only by the users that play a specific role inside those spaces, such as the owner of a house, an employee in a office, etc. Since there is no prior knowledge of exactly which entities will enter an environment and interact with the locally available devices and present users, a regulation enforcement mechanism needs to be applied to ensure that some basic interaction norms will be obeyed [9], such as, for example, that in a smart classroom, only a user in the instructor role is able to display his mobile device's screen through the classroom projector.

In this regulatory mechanism the access conditions may be defined by rules that establish access policies, which are a means of dynamically constraining and regulating a system's behaviour without changing code or requiring the co-

operation of the components being governed. Policy-based approaches have many benefits, including reusability, efficiency, extensibility, context-sensitivity, verifiability, support for both simple and sophisticated components, protection from poorly designed, buggy, or malicious components, and reasoning about component behavior [16].

This work proposes a system that applies regulatory mechanisms to coordinate the interaction among client/server applications in a ubiquitous computing scenario. In our approach, the policies are described by rules correlating some context variables of an application, and the regulation service auto-detects the permissions for the clients to access a service independently from any synchronous query. The next section presents a motivating scenario. Section 3 describes the basic concepts. In Section 4, we present our approach In Section 5, we describe a case study based on the proposed scenario. Finally, Section 7 brings the conclusions about the proposed system.

2 Scenarios

As a typical scenario to exemplify our approach, we consider two universities in two different countries, for instance, PUC-Rio, in Brazil, and LIP6, in France. We assume that these organizations have a positioning/location service for detecting the symbolic location of a user (i.e. her mobile device), such as the one developed in the middleware MoCA [12]. At these two universities there would be several atomic spaces, which are the smallest symbolic areas distinguishable by the location service such as classrooms, laboratories, seminar rooms, offices, corridors, etc. Although each of these spaces corresponds to a different geographical region, some of them have similar uses. People visiting these spaces are assumed to have a well defined role in the corresponding organizations, e.g. a lecturer, a student or an administrative staff member, a temporary visitor from another institution, or even an unknown visitor.

In the proposed scenario, which involves an *active learning environment*, we further assume that some classrooms and seminar rooms are equipped with several multimedia and portable devices, as well as several applications that enable a rich and interactive learning experience among instructors and students, such as the Active Classrooms at PUC-Rio. Each person in this scenario carries a mobile device with a wireless LAN interface and capable of executing applications that access different services, some of which may be subject to location-based and administrative regulation. For instance, when a lecturer enters an active classroom at his university, he can use a specific client application running on his mobile device to access a *slide presentation service* running on a server executing on a (static) host within that same place. In this case, the authorization to use the service would depend on detecting the user's presence in the classroom and then checking if she has the role of a lecturer in the corresponding institution. Conversely, such authorization would not be granted to a student or visitor.

As another application, we consider a *chat application* suitable for experimental classes, where a server executing on the mobile device of the lecturer makes it possible for the students to interact with each other (and with the lecturer) through a chat client for a "silent" exchange of ideas. But the chat should be available only when they are within the same classroom, and as soon as a user leaves the room the service should become unavailable for her. A user identified as a visiting student from another university would be able to use the service, but not a user identified as an unknown visitor. In this case, the service access authorization for the users would result first from detecting that the user running the service is in a given room, then that the users are in the same place and that they play the role of students or visiting students in that institution.

3 Basic Concepts

We believe that a regulation mechanism shall be useful for controlling the interactions among entities in a ubiquitous system. While users carrying mobile devices walk through different spaces of an organization, applications executing on their devices will have to interact with different services, each subject to specific restrictions, such as the ones presented in the scenario of Section 2. In our approach we use the MoCA architecture [11] for providing the context information regarding the computing environment and the device's location. We created an ontology for describing context information involving not only aspects of the topology and the resources but also social aspects, such as organizational features, roles and personal preferences. We chose the SWRL ontology [4] for representing the rules that set the conditions for having access to services. The main reason was that it uses description logics, allows the definition of variables and has some powerful pre-defined expressions (built-ins) adequate to describe context conditions.

3.1 The MoCA Architecture

MoCA architecture provides support for developing and executing distributed context-aware applications, particularly those that comprise mobile devices interconnected through wireless infrastructured LANs (802.11b/g). The services provided by MoCA support the collection, distribution and processing of context information acquired directly from the mobile devices or inferred through context services. Among then, the Context Information Service (CIS) is responsible for managing the information about the availability of resource of the devices involved, such

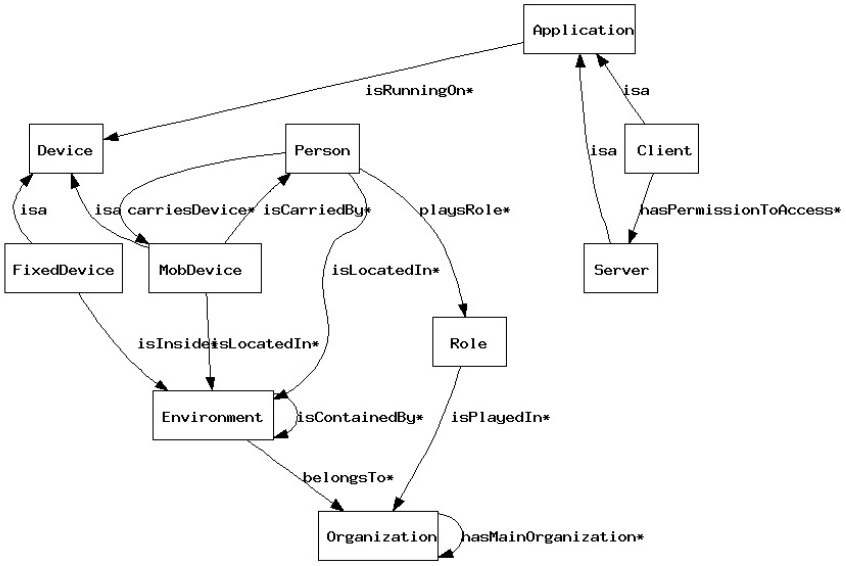

Figure 1. MoCA generic ontology

as free memory or battery charge. The Location Inference Service (LIS) is responsible for inferring the location of a mobile device from the information about RF signal patterns collected from reference points [8]. Besides context management, CIS and LIS also implement context monitoring, allowing clients to register their interest in specific context states (involving one or several context variables) modeled as logical expressions, and to be asynchronously notified whenever the corresponding context-expression is satisfied [12]. This functionality reduces the cost to build client applications, since they are relieved from managing the context information delivery.

3.2 The MoCA Ontology

The use of ontologies to represent context information in a ubiquitous system has not only the advantadge of enabling the reuse and sharing of common knowledge among several applications [13], but also of allowing the use of logic reasoning mechanisms to deduce high-level contextual information [18]. We based our meta-ontology on the DynaCROM [3] generic ontology. This ontology considers three basic concepts: Environment, Organization and Role, that represent separate contextual scopes. We have extended this ontology to comprise three new concepts: Device, Person and Application.

In our approach, while the Environment concept has a topological semantics and Device represents each computational device with its resources (e.g. battery level, memory available), Organization, Role, Person and Application concepts represent the social aspects. The Environment describes physical spaces, places such as buildings or rooms, and subclasses of Environment may describe specific kinds of spaces that are common to different organizations, such as a classroom, for example. Device describes the characteristcs of the computational devices. Its mandatory subclasses are Mobile Device, which may comprise subclasses such as PDA, Smartphone, etc, and Fixed Device, that may describe a stationary host. The Organization describes some social structure or institution, like a university, that may have as subclass a department, for example. Organization describes institutions and their particular aspects. Role describes some social or professional function attached to a given individual while Person describes the personal characteristcs and preferences of a individual. Finally, Application comprises the aspects related to the software agents.

In each case, for describing a particular domain, the generic ontology has to be instantiated, with the definition of specific individuals for thet domain, and may extended with the definition of appropriate suclasses. For example, we can think of University and High School as subclasses of Organization, and PUC-Rio as an individual that is an instance of the subclass University. Or Country and City as subclasses of Environment, and Brazil or France as instances of the subclass Country. Organization and Environment concepts are orthogonal and may be intertwined, i.e, arbitrary relationships between subclasses and instances of the basic classes may be defined freely. For example, we may think of a classroom 511 (Environment instance) at PUC-Rio (Organization instance), in Brazil (Environment instance), and a classroom 534 (Environment instance) at LIP6 (Organization instance), in France (Environment instance). A Role may also have subclasses that indicate some special kind of social role valid across organizations and en-

vironments. Person and Application concepts represent the human and software agents that interact in a given enviror-ment. For example, Marie (Person instance), who is a student (Role instance) from LIP6, may be visiting PUC-Rio and attending a meeting at classroom 511. She carries an HP smartphone (Device instance), running a browser (Application instance).

3.3 SWRL

For representing the rules of access to services, we choose the Semantic Web Rule Language (SWRL) because, as it uses description logics, it allows a straightforward description of information derived from classes and properties of an ontology. Besides, it allows the definition of variables and has some pre-defined expressions (built-ins) that are particularly useful for describing context conditions [4]. SWRL allows users to write rules to reason about OWL individuals and to infer new knowledge about those individuals. It extends the set of OWL axioms to include Horn-like rules, that then can be combined with an OWL knowledge base. The proposed rules are of the form of an implication between an antecedent (body) and consequent (head). The intended meaning can be read as: whenever the conditions specified in the antecedent holds, then the conditions specified in the consequent must also hold. Both the antecedent (body) and consequent (head) consist of zero or more atoms. Multiple atoms are treated as a conjunction. Atoms in these rules can be of the form C(x), P(x,y), sameAs(x,y) or differentFrom(x,y), where C is an OWL description, P is an OWL property, and x,y are either variables, OWL individuals or OWL data values. SWRL does not support negated atoms or disjunction. Using an informal description logic notation, in the form *"antecedent"* $\Rightarrow$ *"consequent"*, an example of a rule asserting that the composition of playsRole and isPlayedIn properties implies the worksIn property would be written as:

$$\textbf{Person(?x)} \wedge \textbf{playsRole(?x,?y)} \wedge \textbf{isPlayedIn(?y,?z)} \Rightarrow \textbf{worksIn(?x,?z)}$$

In the above rule, variables were indicated using the standard convention of prefixing them with a question mark (e.g., ?x). The execution of this rule would cause the addition of the worksIn binary property to all OWL individuals related by the property playsRole to another individual which is related by the property isPlayedIn to another individual. In practice, an XML syntax is used to represent these rules, which then can be stored in OWL text files. Using this representation, the same example rule would be represented as in Code1.

As a particularly useful feature, SWRL includes a set of built-in predicates that allow to describe the relation of variables with concrete domains, such as integers and strings. Built-ins may occur in the body of a rule and have a predefined logical meaning (a fixed interpretation, such as comparison operators $<$, $>$,$=$,$\neq$, etc) and can be considered as dynamically evaluated predicates.

$$\textbf{Room(?x)} \wedge \textbf{hasTemperature(?x,?y)} \wedge \textbf{lessThan(?y,15)} \Rightarrow \textbf{ColdRoom(?x)}$$

As an example, the execution of the above rule would cause the addition of the ColdRoom unary property to all OWL individuals with a given individual as temperature which has value less than 15 degrees.

Code 1 :*Rule description*

```
<ruleml:imp>
 <ruleml:rlab ruleml:href="#example1"/>
 <ruleml:body>
  <swrlx:binaryPropertyAtom swrlx:property="playsRole">
   <ruleml:var>x</ruleml:var>
   <ruleml:var>y</ruleml:var>
  </swrlx:binaryPropertyAtom>
  <swrlx:binaryPropertyAtom swrlx:property="isPlayedIn">
   <ruleml:var>y</ruleml:var>
   <ruleml:var>z</ruleml:var>
  </swrlx:binaryPropertyAtom>
 </ruleml:body>
 <ruleml:head>
  <swrlx:binaryPropertyAtom swrlx:property="worksIn">
   <ruleml:var>x</ruleml:var>
   <ruleml:var>z</ruleml:var>
  </swrlx:binaryPropertyAtom>
 </ruleml:head>
</ruleml:imp>
```

4 Proposed Approach

This work proposes a regulation mechanism to be used to control the interactions among entities in ubiquitous systems, where client applications running on mobile devices interact with different entities, each responsible for a given set of services constrained to a given environment. For that sake, it is necessary to formalize the description of rules and that provides a flexible and easy-to-operate mechanism to support the regulation of interacting applications. That mechanism should also permit that rules be created, deleted and modified at runtime [2]. By making explicit the service access rules in a ubiquitous environment and keeping them separate from the implementation of the service itself, we obtain several advantages. First, we make simpler both the rules and the service implementations. Second, we facilitate

the rule management independent of services. Third, when visiting different domains, that is, environments where the type of information probed from from sensors may be different, or in an evolutionary scenario [10], where new types of context are added, or the underlying context model is changed, rules may be more easily adapted to the new situation.

In general, regulation can take two forms. In the first approach, applications use the information about current norms (norms may dynamically change depending on the location, role, etc.) as an additional criterium to trigger their own events. In the second approach, applications are considered black boxes and an external mediation mechanism enforces that all the interactions with the services follow the norms. We chose the second approach, associating access rules to each service available in a ubiquitous system and providing a regulatory service that monitors the context information of each entity in that system, granting access to services only for those clients that meet the necessary conditions.

We propose then the implementation of a specific service to regulate the interaction between client and server applications by collecting context data from sensors, reasoning about the service access rules and enforcing that the interacting entities stick to the rule's outcomes. This is done by the Domain Regulation Service (DRS), which is implemented by a server that can receive the subscription of each application server available at a given organization domain and each application client that is interested in accessing the services in this domain. The application servers must inform the conditions under which they will provide access to their services, i.e. when a service becomes available in the domain, it must send to the DRS the rules that describe the context conditions that the clients must satisfy to gain access to the service.

Application clients subscribe to DRS informing on which device they are running, and from this moment on the device's context data will be monitored by the regulation service based on the data ontology available in the organization and the data provided by sensors, particularly by the MoCA core services. Whenever a given application client comes to satisfy the necessary condition to have access to a specific application server, the DRS issues a temporary token and sends it to both applications. Then the client must include the token in each message that it sends to the server. When the condition is not satisfied by the client anymore, the DRS notifies both application server and client again and the token becomes invalid.

Figure 2 shows the architecture for the proposed service. The **Ontology Manager** module is responsible for loading the **Ontology Data Base** from a data storage and interpreting it, creating a runtime data representation of the instances and properties described in the ontology (i.e. those data

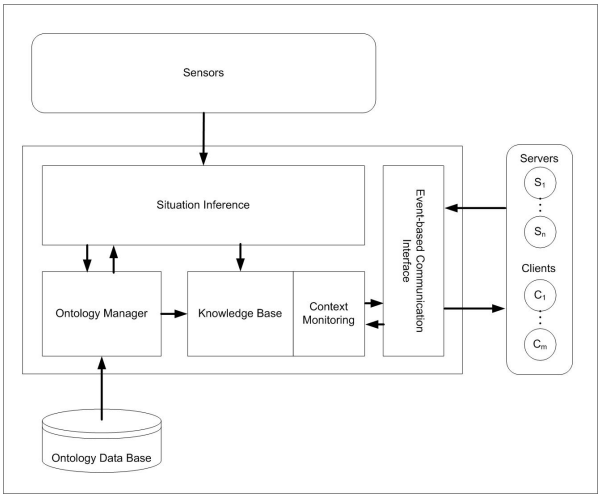

Figure 2. Architecture for the regulation service

that are not dynamic), which consists in the **Knowledge Base**. Context information collected from sensors is also interpreted according to the ontology by the **Situation Inference** module and the respective changes in the properties are added to the Knowledge Base. Server aplications will susbcribe the service informing an access rule, and client applications will subscribe to be notified about an appropriate service. The **Context Monitoring** module will check when a client's context description satisfies a given rule, and then application server and client are notified by the **Event-based Communication Interface** module.

We discuss, in the next subsections, some important conceptual aspects that have to be considered in the description and implementation of the regulation service.

4.0.1 Access Rules

The service access rules are comprised of a set of conditions which a client application context description must fulfill in order to be allowed to interact with the service. As those rules are defined using SWRL, they may combine variables with classes, properties or instances defined in the ontology, which results in a great expressive power. For example, assuming a presentation server running on a static host somewhere but that controls a datashow device in a given classroom at PUC (Room 511), then, an application (class), in order to be allowed to connect to that server (class), should be executing (property) on mobile device (class) that is located (property) inside room 511 (instance), and operated by (property) a person (class) who should play the (property) role of a lecturer (instance). This condition may be defined as the antecedent of the following the rule:

Client(?c) ∧ Server(?s) ∧ isRunningOn(?c,?d) ∧ isLocatedIn(?d,"Room511") ∧ isCarriedBy(?d,?p) ∧ playsRole(?p,"Lecturer") ⇒ hasPermissionToAccess(?c,?s)

The consequent of the rule asserts a new property "hasPermissionToAccess" relating the instances of Client and Server — which are two subclasses of Application, as shown in Figure 1. This property is added dynamically to the knowledge base when the **Situation Inference** and the **Context Monitoring** modules identify that a client satifies the proper conditions.

4.1 Dynamic Properties and Event-triggered Reasoning

The reasoning process to verify if a given permission must be granted to a client involves not only the monitoring of context variables that are obtained from some ontology that describes an institution, such as the properties "isCarriedBy" and "playsRole", but also those that are collected from different distributed sensors and vary dynamically, such as the property "isLocatedIn". We name these properties that are not pre-defined in the ontology data base as "dynamic properties", because they are effective only after a sensor produces the proper context values related to them, and are valid only while not replaced by some new context information. As such, they are first defined and change dynamically. Therefore, reasoning operations are event-triggered, because they need to occur only when there is some context data change, that is, when some sensor notifies a different value for the corresponding context type.

Considering the example rule presented before, a new reasoning operation happens only at the event of a change of the property "isLocatedIn" for a monitored device. Since each application client executes on a unique mobile device and the MoCA services also provide context information about devices (location, CPU usage, available memory, etc), we have a device-centric inference process.

To allow a more efficient monitoring of the entities in a system by the **Context Monitoring** module, the inference service internally defines some intermediary properties, which are equivalent to inferred values for some identifiable subparts of a proposed rule. For example, for a client application being monitored under the rule predented above, we define two intermediary properties that apply to the device being monitored. The first is "isLocatedIn" (property001) and the second is the combination of the properties "isCarriedBy" and "playsRole" (property002). For example, if a client application is running on a given device, it will be monitored to check if the device has "property001=Room511 and property002=Lecturer". This expression needs to be reevaluated only if any of the constituent properties change.

4.2 Ephemeral Properties and Latency of a Rule

We define the "hasPermissionToAccess" property relating instances of Client and Server as an "ephemeral property", because it is valid only if and while the condition is true. If the condition becomes false, the property has to be deleted from the knowledge base. As such, the **Situation Inference** and the **Context Monitoring** modules that were responsible for setting this rule when a client came to satify the conditions, then start to monitor when the conditions become false, for removing the property from the knowledge base when it happens.

As some context information used to infere the access permission comes from sensors that may be unstable, the permission to access a service could also become unstable and bounce in some circunstances. To prevent this from happening we added a latency to be associated with each rule. In practice, we extended the SWRL ontology used to describe rules, defining the latency as a property for the "rule" concept. The application servers that present their access rules, must define this latency time. When the DRS will identify a permission as not valid anymore only if the access conditions leave to be satisfied for a period longer than the latency.

5 Case Study

This case study highlights the great expressive power of our approach. It describes a specific situation to show the regulation of services using permisson rules based on complex expressions that comprise dynamic context data provided by the MoCA services. We will consider the following permission rule:

Server(?s) ∧ isRunningOn(?s,?d1) ∧ isLocatedIn(?d1,?r) ∧ isCarriedBy(?d1,?p1) ∧ playsRole(?p1,"Lecturer") ∧ Client(?c) ∧ isRunningOn(?c,?d2) ∧ isLocatedIn(?d2,?r) ∧ hasFreeMemory(?d2,?x) ∧ greaterThan(?x, 200) ∧ isCarriedBy(?d2,?p) ∧ playsRole(?p,"Student") ⇒ hasPermission(?c,?s)

In this example we assume a service which must be running on the mobile device of a lecturer. The application is a collaborative chat server that allows the teacher and students in the same classroom to exchange messages and files, including .JPG images. The students have to be in the same classroom with their mobile devices for the client applications that they are executing be able to connect to the server. Besides, the students's mobile device must have the amount of free memory greater than 200kB to be allowed to connect.

In practice, when a server application S with the aforementioned characteristics, subcribes the DRS, supposing

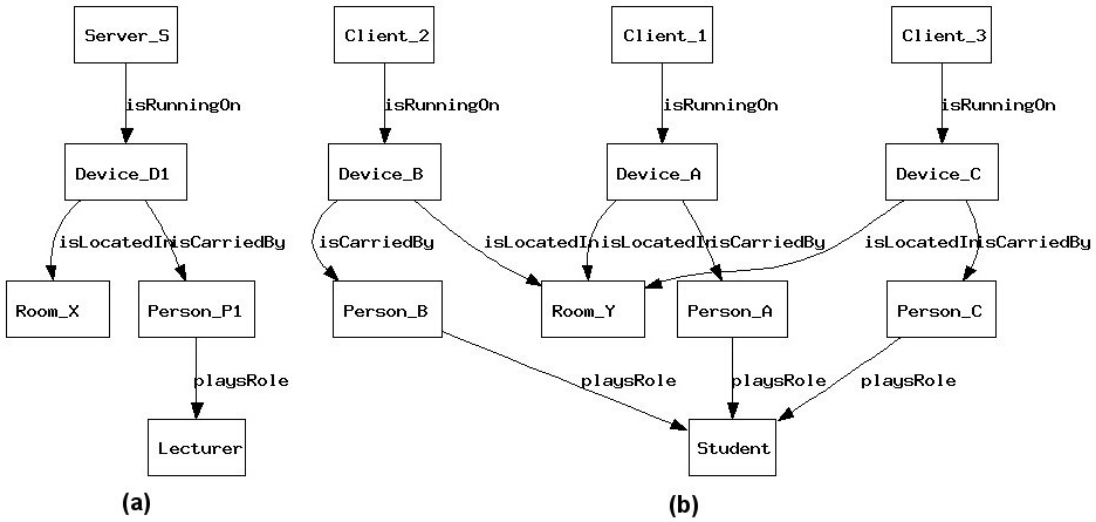

Figure 3. (a) Context information of Server S. (b) Context information of Clients 1, 2 e 3.

that this application is executing on Device *D1* which is located in the *Room X* and is operated by *Person P1* that is a *Lecturer*, DRS starts to monitor the respective application clients checking for those running on devices that have "property001=RoomX and property002>200 and property003=Student". Figure3a exemplifies the situation of this application server. While this expression is reevaluated whenever any of the constituent properties of the client's device change, if the location of the server changes, the expression being monitored also changes. For example, in Figure3b, supposing that *Client 1* is evaluated as having "property001=RoomY and property002=300 and property003=Student", if *Person A* enters *Room X* the expression will be reevaluated as true and both client and server will be notified. On the other hand, if instead *Person P1* moves to *Room Y*, the new expression being checked now is "property001=RoomY and property002>200 and property003=Student", and again the expression will be evaluated as true for *Client 1*, but also for *Client 2* and *Client 3*.

6 Related Work

The use of policies which constrain the behavior of system components, are becoming an increasingly popular approach to dynamic adjustability of applications in academia and industry [16]. Multiple approaches for policy specification have been proposed that range from formal policy languages that can be processed and interpreted easily and directly by a computer, to rule-based policy notation using an if-then-else format, and to the representation of policies as entries in a table consisting of multiple attributes [7].

KAoS [16] consists in a set of platform-independent services that allow the definition of policies to ensure the adequate control over distributed systems. It is one of the first systems to use OWL , i.e., description logic, to describe and specify policies and context conditions. Therefore, KAoS is able to classify and reason about both domain and policy specification basing on ontological subsumption, and to detect policy conflicts statically, i.e., at policy definition time. However, a pure OWL approach encounters some difficulties with regard to the definition of some kinds of policies — particularly those requiring the definition of variables. Relying purely on OWL, it is not possible to define policies that refer to statically unknown values [15].

Rei [5] is a flexible and expressive policy language that is based on deontic concepts, i.e. logic programs, and which can be used to describe several kinds of policies. It allows the description of individual policies as well as group and role based policies. Differently from KAoS, Rei's rule-based approach enables the definition of policies that refer to dynamically determined values, i.e. context variables, thus providing greater expressiveness and flexibility to policy specification [15]. On the other hand, the choice of expressing Rei rules similarly to declarative logic programs prevents it from exploiting the full potential of the OWL language.

In comparison to other rule-based regulation systems, our approach has the advantages fully exploiting the capabilities of the OWL language to describe rules. Besides that, it also allows the definition of rules of access that refer not to static values (or instances) but to dynamically determined values, as discussed in the cases study presented in Section 5. Furthermore, our regulation service is based in the monitoring of the context values to check them against the rules proposed by the services available in a domain.

Our approach is proactive, as such, DRS auto-detects the permissions for the clients, independently from any synchronous query. That feature is particularly adequate for ubiquitous spaces, where applications must interact seamlessly.

7 Conclusion

This work proposes a system that applies regulatory mechanisms to coordinate the interaction among applications in a ubiquitous computing scenario. In our approach, explicit rules are used to describe which applications have access to a service. These rules are comprised by a set of conditions which a client application context description must fulfill in order to be allowed to interact with the service. This service regulates the interaction between client and server applications collecting context data from sensors, reasoning about the permission rules and and enforcing that the interacting applications stick to the rule.

In comparison to other rule-based regulation systems, our approach has the advantages of fully exploiting the capabilities of the OWL language to describe rules, while also enabling the definition of rules of access that refer to dynamically determined values, instead of pre-defined and fixed values. In our approach the rules of access do not need to be previously described for each service. Instead, our regulation service allows that when services become available in a given domain, they present their own access rules. This allows that services running on mobile devices that travel through different domains may use this regulation service.

As its main feature, the proposed system is proactive, that is, by monitoring the context information available, our service auto-detects when any client has permission to access a service independently from any synchronous query, and informs both the client and the server. This characteristic is what makes our system the most adequate for ubiquitous spaces, where applications must interact seamlessly with each other, in a transparent way for the users. As the monitoring associated with the reasoning process may be computationally intensive, our system still tries to achieve a more efficient operation by executing the reasoning operation only when there are changes in context data collected from sensors. As future work we will implement different services with their appropriate set of rules to collect performance results and evaluate the efficiency of the proposed approach with different sets of rules and a growing number of entities involved.

References

[1] D. Chakraborty, A. Joshi, and Y. Y. amd T. Finin. Toward distributed service discovery in pervasive computing environments. *IEEE Transactions on Mobile Computing*, 2006.

[2] C. Felicíssimo, R. Choren, J.-P. Briot, and C. Lucena. Supporting regulatory dynamics in open MAS. In *Proceedings of AAMAS06 - Workshop on Coordination, Organization, Institutions and Norms in Agent Systems (COIN-06)*, 2006.

[3] C. Felicíssimo and C. Lucena. An approach to regulate open multi-agent systems based on a generic normative ontology. In *Proceedings of the 1st Workshop on Software Engineering for Agent-oriented Systems (SEAS 2005)*, 2005.

[4] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A semantic web rule language combining owl and ruleml. W3C member submission, 2004.

[5] L. Kagal, T. Finin, and A. Joshi. A policy language for a pervasive computing environment. In *Proceedings of the IEEE 4th International Workshop on Policies for Distributed Systems and Networks (POLICY 2003)*, pages 63–74, June 2003.

[6] T. Kindberg and A. Fox. System software for ubiquitous computing. *Pervasive Computing Magazine*, 2002.

[7] R. Montanari, A. Toninelli, and J. Bradshaw. Context-based security management for multi-agent systems. In *Proceedings of the IEEE 2nd Symposium on Multi-Agent Security and Survivability*, pages 75–84, August 2005.

[8] F. N. Nascimento, V. Sacramento, G. Baptista, H. K. Rubinsztejn, and M. Endler. Development and evaluation of a positining service based in ieee 802.11 (in Portuguese). In *Proceedings of the XXIV Brazilian Symposium on Computer Networks (SBRC 2006)*, 2006.

[9] R. Paes, G. Carvalho, H. Almeida, C. Lucena, and P. Alencar. A conceptual architecture for law-governed open multi-agent systems. In *Proceedings of Argentine Symposium on Software Engineering (ASSE2004)*, 2004.

[10] R. Rocha and M. Endler. Context management in heterogeneous, evolving ubiquitous environments. *IEEE Distributed Systems Online*, 7(4), April 2006.

[11] H. K. Rubinsztejn, M. Endler, V. Sacramento, K. Gonçalves, and F. N. Nascimento. Support for context-aware collaboration. *First International Workshop on Mobility Aware Technologies and Applications (MATA 2004)*, 5(10):34–47, 2004.

[12] V. Sacramento, M. Endler, H. K. Rubinsztejn, L. S. Lima, K. Gonçalves, F. N. Nascimento, and G. A. Bueno. MoCA: A middleware for developing collaborative applications for mobile users. *IEEE Distributed Systems Online*, 5(10), 2004.

[13] A. Shehzad, H. Q. Ngo, K. A. Pham, and S. Y. Lee. Formal modeling in context aware systems. In *Proceedings of the First International Workshop on Modeling and Retrieval of Context*, September 2004.

[14] J. Soldatos, I. Pandis, K. Stamatis, L. Polymenakos, and J. L. Crowley. Agent based middleware infrastructure for autonomous context-aware ubiquitous computing services. *Journal of Computer Communications*, 2006.

[15] A. Toninelli, L. Kagal, J. M. Bradshaw, and R. Montanari. Rule-based and ontology-based policies: Toward a hybrid approach to control agents in pervasive environments. In *Proceedings of the Semantic Web and Policy Workshop (SWPW)*, November 2005.

[16] A. Uszok, J. M. Bradshaw, M. Johnson, R. Jeffers, A. Tate, J. Dalton, and S. Aitken. Kaos policy management for semantic web services. *IEEE Intelligent Systems*, pages 32–41, July/August 2004.

[17] J. Viterbo, C. Felicíssimo, J.-P. Briot, M. Endler, and C. Lucena. Applying regulation to ubiquitous computing environments. In *Proceedings of the 2nd Workshop on Software Engineering for Agent-oriented Systems (SEAS 06)*, pages 107–118, Outubro 2006.

[18] X. Wang, D. Zhang, T. Gu, and H. Pung. Ontology based context modeling and reasoning using OWL. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, pages 18–22, March 2004.

[19] M. Weiser. The computer for the twenty-first century. *Scientific American*, 265(3):94–104, September 1991.