# A Universal Model for Policy-Based Access Control-enabled Ubiquitous Computing

**Yixin Jing*, Jinhyung Kim*, and Dongwon Jeong****

**Abstract:** The initial research of Task Computing in the ubiquitous computing (UbiComp) environment revealed the need for access control of services. Context-awareness of service requests in ubiquitous computing necessitates a well-designed model to enable effective and adaptive invocation. However, nowadays little work is being undertaken on service access control under the UbiComp environment, which makes the exposed service suffer from the problem of ill-use. One of the research focuses is how to handle the access to the resources over the network. Policy-Based Access Control is an access control method. It adopts a security policy to evaluate requests for resources but has a light-weight combination of the resources. Motivated by the problem above, we propose a universal model and an algorithm to enhance service access control in UbiComp. We detail the architecture of the model and present the access control implementation.

**Keywords:** Access control, Ubiquitous computing, Task computing, Context-awareness

## 1. Introduction

The beginning of the merging of web techniques and the ubiquitous environment could be traced back to 2003. Many efforts have gone into this field aiming to deal with the new demand that has arisen along with ubiquitous and pervasive computing. Application in such an "anytime, everywhere" environment is beyond the traditional program presence pattern and has become embedded in daily life without drawing attention. A most important topic in the ubiquitous environment is how to capture and process contextual or situational information in dynamic service requesting. Service is becoming a standard interface over the internet. A service is a function that is well-defined, self-contained, and does not depend on the context or state of other services. The technology of the Web Services [1] is the most likely connection technology of service-oriented architectures. Web services essentially use XML to create a robust connection. To receive a service from a service provider, a service consumer has to send a request message to invoke the exposed service interface. In this case, the service provider returns a response message to the service consumer. The request and subsequent response connections are defined in some way that is understandable to both sides. A service provider can also be a service consumer.

In the ubiquitous computing environment, services are supposed to be the most popular approach establishing the connection among people, devices, and applications. Services published on a Web server can therefore be discovered by the interested service consumer. Through a discovery mechanism such as UPnP [2], the service consumer can call for a service by transferring the corresponding parameters. However, a service means a hardware/software resource in a given subnet. To consume resources the service consumer needs to meet certain pre-conditions. The process to define or limit the access right of the service consumer to the service is thus called 'service access control'. Services are changing as they may be merged, composed, split or moved to other usage scenarios. So the access control to a service needs to be sufficiently adaptive. As introduced in [3, 4], the traditional subject-based access control solutions are inadequate to rule access to resources in cases of frequent user/device mobility and context changes. Instead, the rule based on the properties of a subject, which include the subject's natural properties and contextual properties, is more suitable to meet the dynamic nature of ubiquitous computing [5, 6]. In this paper, we propose a new model to enhance the existing service access control methods. The proposed model is based on a service policy-based access control method rather than the traditional subject-based access control.

The outline of the paper is as follows. In Section 2, we introduce the related work. Section 3 describes the universal model in detail. Section 4 provides an implementation of the access control based on the proposed model. A conclusion of the paper and the outlook for future work is presented in Section 5.

## 2. Related Works and Motivation

Much effort has been applied to the issues relating to

Manuscript received February 2, 2006; accepted March 3, 2006.
**Corresponding Author:** Dongwon Jeong
* Dept. of Computer Science and Engineering, Korea University, Seoul, Korea ({jing, koolmania}@software.korea.ac.kr)
** Dept. of Informatics and Statistics, Kunsan National University, Kunsan, Korea (djeong@kunsan.ac.kr)

services in ubiquitous computing, such as "The Task Computing Environment" (TCE) [7, 8, 9], Ubiquitous Context-based Security Middleware (UbiCOSM) [6], KAoS [10, 11] and Ontology-enabled Services-Oriented Architecture for UbiComp (OASA) [12].

Nevertheless, these service access control models do not address, or scarcely address, the problem of enabling users to obtain the desired services. The only issue related to assisting users to obtain a desired service is addressed in the access control model of Task Computing. The Task Computing access control mechanism exploits the delegation supported by the Rei [13] to help a user reach the required service. The weakness here is that support alone for delegation is not enough. Even the user successfully gets the delegation from somewhere, and he/she still encounters the problem when a service demands other contextual information which the user cannot provide. The access control model of UbiCOSM utilized the context description to authorize rights to users. However, a merely context-centered policy limits the ability of the mobile user to input the service parameters, which results in a policy enforcement process that is totally dependent on contextual information but does not consider the users' preferences.

In a word, a service-rich environment as well as access control introduces many challenges to users wishing to use the service. A service execution depends on two factors: one concerns the user's personal status, such as the user's identification or preferences; the other concerns the contextual information of the users. As for the former factor, the identification and position could be acquired through the handset of the user; his/her preference could be collected when he/she configures the service input parameters. To collect the latter factor, which is not acknowledged by the non-expert user, an easy service access mechanism is needed to offer help. To resolve this problem, we propose a universal model within which the hardware and the service are connected to provide more precise service access control by offering richer contextual information.

## 3. The Universal Model

The universal model for ubiquitous computing is designed for the capability of managing hardware/software, computational/non-computational resources in the ubiquitous environment, and consequently realizing service access control. The structure of the universal model is composed of different interrelated classes as introduced below.

### 3.1 Service Class

Service class is the key portion of the universal model where the other classes are designed to facilitate service creation, modification, and management. (See Def. 1):

**Definition 1** Service: A service consists of a service name, service description, and sets of input/output parameters:
*service = <name, description, {input_para}, {output_ para}>*,
where each service instance could be identified by service.name.

In Service, four properties are provided. Service name is used to identify the service instance. Therefore in the same name space, service name is distinct. Service description is a human readable plain text to help people understand the functionality of the service instance. Input parameter/ output parameter collection depicts the parameter inputting and outputting of the service. Input/output parameter might be the value of the primitive data type, an object, or the collection of the both.

### 3.2 Entity Class

Entity class (see Def. 2) intends to define the sources in ubiquitous environment and represent them along with exposed services. An instance of Entity class might be a computational device (for example an automatically washing machine or a server running Web Service). In all cases, the computational device can independently execute an automatic process without cooperation with other devices. This atomic process, defined as Action (see Def. 3) of the Entity.

**Definition 2** Entity: An entity consists of an entity name and a set of Actions:
*entity = <name, {action}>*,
where each action instance could be identified by action.name.

**Definition 3** Action: An action is composed of an action. name and a set of parameters inherited from Service:
*action = <name, {input_para}, {output_para} >*,
where
input_para/output_para =<para_name, para_type>;
and in either {input_para} or {output_para}, each para_name must be distinct.

An Action is a set of state transitions which can last for anything from a few seconds to several hours depending on the desired goal. The Action exploits the resources of the device where the process is being executed without interacting with other surrounding entities. Action could not be divided into smaller granularity. The Action is the sub-class of Service, therefore it might have input and output parameters which respectively decide how action would be launched and determine the effect of the action. Each parameter has its data type.

Fig. 1 illustrates an example of the Action. The action "washing clothes" has some necessary parameters, such as the "minutes" spent on washing and drying. In this case, the type of parameter is an integer. Another possible input parameter is an integer which decides how many times this action should be executed.

Action instance:
*name*: *washing_clothes*
*input_para:*
*washing_minutes, integer*
*drying_minutes, integer*
*execution_times, integer*
*output_para:*
*null*

**Fig.1.** Example of action instance

## 3.3 Task Class

With the support of Action, the more complicated task could be made up by compositing those action instances. We abstract this type of task as Task class, which derives from Service (see Def. 4).

The Task class is a lightweight composition of action instances. The object property Task.mode clarifies the mode by which the action instances are composed. The collection {action_inst} depicts the action instances involved in a task. Each involved action instance and the name of its related entity comprise each element of the collection.

**Definition 4** Task: A task is composed of a name, and a sequence of actions:
*task = <name, {action_inst}, mode>*,
where
action_inst = <entity.name, action>;
mode = sequent/concurrent

## 3.4 Request Class and Policy Class

A highlight of pervasive computing is that a computing module is seamlessly embedded in various kinds of surrounding facilities. These entities can be aware of a change in the environment and offer adaptive service. These services are triggered by different things, such as information on the location change of a user, a time schedule, or a sudden occurrence. We wrap up these trigger events in terms of Request class (see Def. 5).

**Definition 5** Request: A request is composed of a requested service.name and a set of request_para:
*request = <name, service.name, {input_para}, {context_para}>*,
where
context_para =< para_name, para_type >;
and each context_para.para_name must be distinct.

A request has its own name to enable itself to be identified. The service name denotes which service instance this request is going to access. Input parameter collection carries the parameter values that are necessary to execute a service. The context parameter collection takes the contextual information when this request is composed.

To prevent service abuse, a specially designed filter is needed to evaluate requests to the service, and consequently realize the access control to a service. Class Policy is used to accomplish this evaluation (see Def. 6). A policy approach is widely used in access control for network security [14, 15]. The properties of policy include a distinct name as an identifier and a service name denoting with which service instance this policy is associated. The collection of clauses defines the principle to evaluate a request. A clause is a boolean combination of expressions for a context parameter of the request. If every context parameter satisfies the corresponding clause, the request is qualified to access the service. A context awareness access control can capture contextual information along with the request and examine them one by one with the conditions defined in the policy. (See Fig. 2). From Fig. 2, we can tell that request930@korea.ac.kr could not access service01 because the time value was not in the range of that which is enforced in the policy, although the temperature value did fulfill the requirement.

**Definition 6** Policy: A policy has the following structure,
*policy=<name, service.name, {clause}>*,
1) *clause = expression {O expression}*, where O is a logical operator (such as AND, OR);
2) *expression=<para_name□val>*, where *para_ name* denotes the name of the parameter; *val* is the value of the parameter; and □ is any relational comparison operator (such as ≥, >).

| Request instance: | Policy instance: |
|---|---|
| *name: request930@korea.ac.kr* | *name: policy01* |
| *service.name: service01* | *service.name: service01* |
| *input_para: Null* | *clauses:* |
| *context_para:* | *clause 1:* |
| *time=14:00* | *time≥16:00* AND *time≤18:00* |
| *temperature=26* | *clause 2:* |
| | *temperature ≥25* |
| a) Request instance | b) Policy instance |

**Fig. 2.** The example of a request-policy pair

The overview of the universal model is outlined as in Fig. 3.

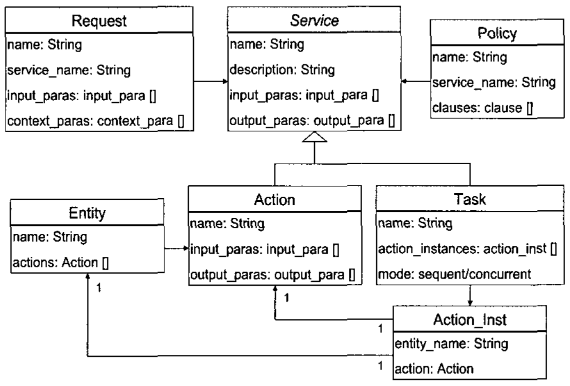


**Fig. 3.** Overview of the universal model

## 4. Implementation and Discussion

This section describes the system architecture and request evaluation algorithms for implementation, and discussion of the proposed model.

### 4.1 Architecture and Key Algorithms for Implementation

The system architecture is composed of two sides. The first concerns the Service Management Environment as shown in Fig. 4. The Entity Management module registers the entity information with the local repository. The registered content includes the entity name, type, manufactory and, most importantly, the action that the entity provides. According to the registered action information, the Service Management can translate the action into the standard Web Service interface and publish it in the Service Engine waiting to be discovered. In the same way, the Task Management module could integrate the action to form a more complicated behavior. The integrated task can also be mapped to the Web Service by the Service Management module. Therefore, the conceptual service is connected to the concrete entity resource, which can not only complete the service execution but also provide more contextual information through the sensor entity as long as the corresponding services are invoked.

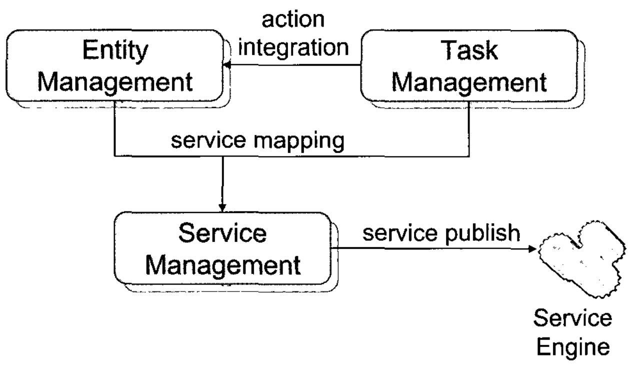


**Fig. 4.** Service Management Environment Architecture

The other side of the system is the access control environment framework, which allows the universal model-based service request and service policy to be specified and enforced through an application. The primary system design is depicted in Fig. 5. An access control environment is a middle ware residing between service request submission and service execution. The main module of the environment consists of a Policy Administration Point (PAP), a Policy Decision Point (PDP), and a Response Handler (RH). The interaction between the access control environment and the other module outside occurs through three interfaces: the PDP-Service Request Generator (SRG), RH-SRG, and RH-Service Engine (SE).

SRG has two functionalities: the first serves to connect some sensors to listen and collect the real-time information; the other is to receive a service invoking signal from

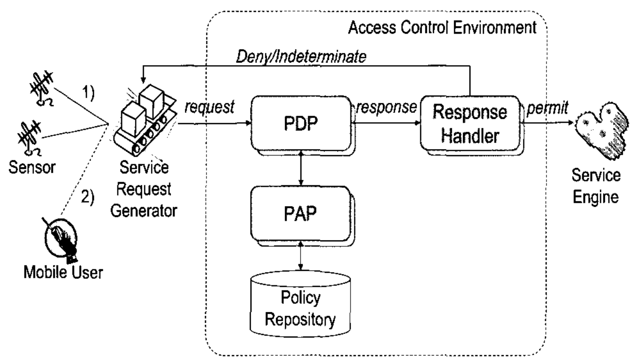


**Fig. 5.** Access Control Environment Architecture

the mobile user and transfer it to a system understandable format. In the ubiquitous computing environment, a mobile user can discover his/her available service via UPnP. The user can compose a simple service request and send it to SRG in order to acquire the service. SRG is responsible for expressing this simple request in a PDP understandable format and may collect more contextual information, which cannot be provided by the mobile user, through the first functionality.

```
Procedure serviceAccess(req){
  get service_name from req;
  get service_policy of service_name;
  evaluateResult =evaluateReq(reqID, service_policy_name);
  if (evaluateResult is "violate") return;
    else if (evaluateResult is "qualified) {
      executeService(service_name);
      return;
    }
   else if (evaluateResult is "insufficient"){
      new_req =requestMoreCondition(insufficient_clauseList);
      serviceAccess(new_req);
    }
}
                              a)
```

```
Procedure string evaluateReq(req, service_policy_name){
  req_condition[]=getConditionfromReq(reqID);
  pol_condition[]=getClauses(service_policy_name);

  for(i=0,i<req_condition[].length,i++){
    for(j=0,j<pol_condition[].length,j++){
      if(req_condition[i].para_name
        = =pol_condition[j].para_name){
        result =
          check(req_condition[i].req_para, pol_condition[j]);
        if (result = = false) return "violate"
      }
    }
  }

  for (each element q in pol_condition[] but not in req_condition[]){
    insufficient_clause_list.add(q);
  }
  if (insufficient_clause_list.length>0) return "insufficient";
    else return "qualified";
}
                              b)
```

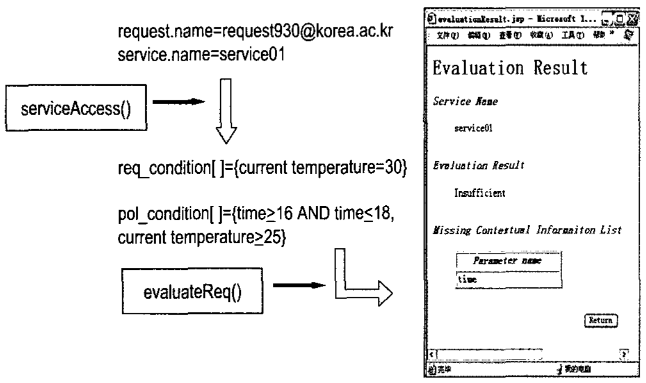
**Fig. 6.** Algorithm for Service Request Evaluation

With the support of PAP, PDP can look up the policy associated with the given target service in the request. Afterwards, PDP is responsible for evaluating the request against the policy, which results in a response message. Response Handler is in charge of analyzing the response message to determine the next step. The response message might carry one of the three results of the request evaluation, which are "qualified", "violate" or "insufficient". If the request satisfies all of the clauses in the policy, Response Handler can notify the Service Engine to invoke the corresponding service. Otherwise, the service cannot be executed. In the event that any clause of the policy is violated, "violate" is returned to SRG and consequently the service request process is terminated. If the contextual parameters in the request fulfill the clauses of the policy but at least one clause of the policy cannot find the corresponding parameter in the request, "insufficient" along with the missing parameter are returned to SRG.

The aforementioned access control environment requires an algorithm to evaluate the request upon the policy. For this reason, we suggest a primitive algorithm to implement the PDP. The algorithm is divided into two procedures: serviceAccess (Fig. 6 a) and evaluateReq (Fig. 6 b). The former is responsible for receiving the incoming request and calls upon the latter to check the request against task policy. If the request does not violate the policy but lacks certain contextual information, evaluateReq can return the unsatisfied conditions.

## 4.2 Evaluation and Discussion

To evaluate the key algorithm, we implement a JSP application to deal with the access authority of the incoming requests. Fig. 7 illustrates the processing steps with an example. In Fig. 7, time, the contextual parameter of the request in Fig. 2, is removed. As a result, the request evaluation informs of this missing contextual parameter.


request.name=request930@korea.ac.kr
service.name=service01
serviceAccess()
req_condition[ ]={current temperature=30}
pol_condition[ ]={time≥16 AND time≤18, current temperature≥25}
evaluateReq()
evaluationResult.jsp - Microsoft I...
Return

Evaluation Result

*Service Name*

service01

*Evaluation Result*

Insufficient

*Missing Contextual Informaiton List*

| Parameter name |
|---|
| time |

Fig. 7. Service Request Evaluation Process

In the above example, we did not deny the service request for the missing parameter but returned the parameter name. Consequently, the request can try to acquire the missing parameter according to the evaluation result. Compared to other access control model, we can summarize as shown in Table 1.

**Table 1.** Qualitative comparison

| Comparative Items | Access Control of TCE | UbiCOSM | KAoS | OASA | Our model |
|---|---|---|---|---|---|
| Policy | Rei language | RDF | KAoS | Rei or KAoS | Policy clause array |
| Entity Resource Management | Yes | No | No | Yes | Yes |
| Service Composition | Support | Not support | Not support | Support | Support |
| Return missing contextual parameter | No | No | No | No | Yes |

As shown in Table 1, the policy representation strategy varies in terms of the language. As for the entity resource, access control of the TCE, OASA as well as our universal model provide the management to a different degree. The same three approaches also provide the service composition. From the perspective of the missing parameter of the service request, only the suggested universal model can return the missing parameter, while the other approach will deny the request without any further information.

## 5. Conclusion

In this paper, we proposed a universal model to achieve adaptive service access control in the ubiquitous environment. The universal model is orienting the service. In the model the hardware and software are recognized as the entity, and each entity has its exposed action published as a service. Action could consist of a more complicated task according to a given mode. Task is present as a service too. To evaluate the request to a service, the system architecture along with the suggested algorithm is exploited to manage a policy-based access control. The definition of the universal model is mapped to the algorithm to implement a context-awareness request/ response.

As the access control in the ubiquitous computing environment remains at a beginning stage, a lot of the area is far from mature. In a future study, a more sophisticated system will be developed, while further research into security and efficiency is also envisaged.

## References

[1] W3C, Web Services Activity, http://www.w3.org/2002/ws/.

[2] UPnP Forum, http://www.upnp.org/.

[3] L. Kagal, T. Finin, A. Joshi, "A Policy Based Approach to Security for the Semantic Web", Proceedings. of 2nd International Semantic Web Conference (ISWC2003), Sanibel Island, Florida, USA, Oct., 2003.

[4] A. Corradi, N. Dulay, R. Montanari, C. Stefanelli, "Policy-driven Management of Mobile Agent Systems",

Proceedings Of Policy 2001, Springer- Verlag, LNCS 1995, Bristol, Jan., 2001.

[5] Ryusuke Masoka, Mohinder Chopra, Yannis Labrow, etc., "Policy-based Access Control for Task Computing Using Rei", Proceedings of the Policy Management for the Web Workshop, pp. 37~43, May, 2005.

[6] Antonio Corradi, Rebecca Montanari, Daniela Tibaldi, "Context-based Access Control for Ubiquitous Service Provisioning", Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC), 2004.

[7] R. Masuoka, B. Parsia and Y. Labrou. "Task computing-the semantic web meets pervasive computing", Proceedings of 2nd International Semantic Web Conference 2003 (ISWC), Florida, USA, pp. 866~881, Oct., 2003.

[8] R. Masuoka, Y. Labrou, B. Parsia and E. Sirin, "Ontology-enabled pervasive computing applications", IEEE Intelligent Systems, vol.18, no.5, pp. 68~72, 2003.

[9] Z. Song, Y. Labrou, R. Masuoka, "Dynamic service discovery and management in task computing", Proceedings of 1st Annual International Conference on Mobile and Ubiquitous Systems: Networking and Sevice (MobiQuitous), Boston, Massachusetts, USA, pp. 310~318, Aug., 2004.

[10] A. Uszok, J. M.Bradshaw, "KAoS policies for web services", W3C Workshop on Constraints and Capabilities for Web Services, Redwood Shores, CA, USA, Oct., 2004.

[11] A. Uszok, J. M. Bradshaw, R. Jeffers, and M. Johnson, "Policy and contract management for semantic web services", Proceedings of AAAI Spring Symposium on Semantic Web Services., California, USA, Mar., 2004.

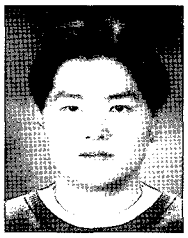
[12] Q. Ni, M. Sloman, "An ontology-enabled service oriented architecture for pervasive computing", Proceedings of International Conference on Information Technology: Coding and Computing (ITCC), Las Vegas, Nevada, USA, pp.797~798, Apr., 2005.

[13] L. Kagal, "Rei : a policy language for the Me-Centric project", HP Labs Technical Report, 2002.

[14] Rafae Bhatti, Elisa Bertino and Arif Ghafoor, "A Trust-based Context-Aware Access Control Model for Web-Services", Proceedings of the IEEE International Conference on Web Services (ICWS), 2004.

[15] Rafae Bhatti, Elisa Bertino and Arif Ghafoor. "XML-Based Specification for Web Services Document Security", IEEE Computer, vol.37, no.4, pp.41~49, 2004.

**Yixin Jing**
He received his BS and MS degrees in Computer Science from Wuhan Univ. (China) in 2001 and 2004 respectively. He has been a Ph.D. student in Korea University since 2005. His research interests include Metadata Registry, Modeling Technology, Software Engineering, Semantic Web, Data Management, Access Control, and Message Management in Ubiquitous Computing.



**Jinhyung Kim**
He received his MS degree in Computer Science from Korea University in 2005. He has been a h.D. student in Korea University since 2006. His research interests include XML Technology, Relational and Object- Oriented DBMS, Semantic Web, Metadata Management, and Ubiquitous Computing.

**Dongwon Jeong**
He received a BS degree in Computer Science from Kunsan National University, Korea, in 1997, an MS in Computer Science from Chungbuk National University, Korea, in 1999, and a Ph.D. in Computer Science from Korea University, Korea, in 2004. He worked as a full-time instructor from 1999 to 2000 in the Advanced Institute of Information Technology, Korea. From 2001 to 2005, he worked as a Senior Research Engineer in the Lime Media Technology Laboratory. He was a Research Assistant Professor in the Research Institute of Information and Communication Technology, Korea University, from 2004 to 2005. From 2002 to the present he has been a committee member of the Telecommunications Technology Association, Korea. 2005 He was a Visiting Research Scholar (Post-Doc.) at the School of Information Sciences & Technology, Pennsylvania State University, PA, USA. He was on the Program Committee of SERA2005 (International Conference on Software Engineering Research, Management & Applications). Now, he is a Professor in the Department of Informatics and Statistics, Kunsan National University, Korea, and is on the Program Committee of EUC2006 and UIC2006. He is one of the Publicity Co-Chairs of SERA2006. His research interests include metadata registry-based integration and application, mobile agent security, XML-to-RDB conversion and hybrid security, ubiquitous computing and semantic maintenance, and ontology conversion.