A General Obligation Model and Continuity-Enhanced Policy Enforcement Engine for Usage Control

Basel Katt
University of Innsbruck
Innsbruck, Austria
basel.katt@uibk.ac.at

Xinwen Zhang
Samsung Information Systems
America, San Jose, CA, USA
xinwen.z@samsung.com

Ruth Breu
University of Innsbruck
Innsbruck, Austria
ruth.breu@uibk.ac.at

Michael Hafner
University of Innsbruck
Innsbruck, Austria
m.hafner@uibk.ac.at

Jean-Pierre Seifert
Samsung Information Systems
America, San Jose, CA, USA
j.seifert@samsung.com

ABSTRACT

The usage control model (UCON) has been proposed to augment traditional access control models by integrating authorizations, obligations, and conditions and providing the properties of decision continuity and attribute mutability. Several recent work have applied UCON to support security requirements in different computing environments such as resource sharing in collaborative computing systems and data control in remote platforms. In this paper we identify two individual but interrelated problems of the original UCON model and recent implementations: oversimplifying the concept of usage session of the model, and the lack of comprehensive ongoing enforcement mechanism of implementations. We extend the core UCON model with continuous usage sessions thus extensively augment the expressiveness of obligations in UCON, and then propose a general, continuity-enhanced and configurable usage control enforcement engine. Finally we explain how our approach can satisfy flexible security requirements with an implemented prototype for a healthcare information system.

Categories and Subject Descriptors

H.1 [**Models and Principles**]: General; D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architecture*

General Terms

Security, Design

1. INTRODUCTION

Access control mechanisms aim to restrict access to sensitive resources (objects) from users and processes (subjects). Traditional enforcement of access control rules applies instantly on access requests, and after an access is granted there is no control on the resource. The growing involvement of information technologies in all sectors of human life like eHealth, eGovernance, eCommerce, and others, results in a dramatic increase of data flow between different actors in distributed environments, where the processing of data is carried out on clients as well as on servers. Consequently, new requirements have risen going beyond traditional access control mechanisms. They demand the control of resources after an access has been granted.

To overcome the shortcomings of traditional access control mechanisms, a usage control (UCON) concept has been introduced with the unique properties of decision continuity and attribute mutability. These features satisfy the security requirements of many contemporary information systems. Different approaches have been proposed to describe the UCON model formally. UCON_{ABC} family model consists of the following core models: Authorization (A), oBligation (B) and Condition (C), with sub-models representing whether an access decision is made before *(pre-)* or during *(on-)* a *usage session*. This leads to the following UCON sub-models: authorization core models *preA and onA*, obligation core models *preB and onB*, and condition core models *preC and onC* [14, 15, 22].

In our work we investigate usage and access control requirements in healthcare systems. Based on some practical studies in the eHealth domain [10, 25] and studies that deal with authorization [3] and privacy [27] issues in eHealth systems, we summarize some example security requirements in this domain:

- Only authorized actors with recognized certificates are allowed to retrieve patient records. Access rights are based on the roles of the actors. For example, pharmacists are allowed only to access the part of the healthcare record containing prescriptions.
- 4-eyes principle: the presence of a patient should be checked during the access session to the record [26].
- Retention time: a patient record should be saved in the doctor's machine for a maximum one month.
- Patient consent: after the end of a treating session, the retrieved document should be stored in the local machine of the doctor in case the patient approves it; otherwise it should be deleted from the system.

SACMAT'08, June 11–12, 2008, Estes Park, Colorado, USA.

- In case the patient is not present before the normal termination of a treating session, the document must be deleted and an abnormal session notification should be reported to the service provider.

These requirements show clearly the importance of usage control: the whole usage of a patient's record should be controlled and multiple authorization checks are needed during this usage. However, from the last two requirements we conclude that some decisions and actions must be taken and executed after each usage session. Such actions are not supported in the current UCON model as it only consists of *(pre-)* and *(on-)* core models. To meet these new requirements the current UCON model and its policy specification have to be extended. First, the family of UCON_{ABC} core models are extended by adding post obligations. We argue that only obligations should be checked after the usage session while (post-) authorizations and conditions are not needed. Second, the sequence of system states that are used to define the context of a resource usage control are extended. Ongoing obligation trigger actions are illustrated with the ongoing-check state.

Our extension is based on the principle that a security model should support as many enforceable security policies as possible. By augmenting UCON core models with post-obligations, we can enforce recently proposed obligation policies in literature [17, 12, 4, 6, 5, 7]. We claim that these policies can be supported by UCON in multi-usage-session environments. For example, the obligation requiring that a user has to pay an amount of money after getting a service is regarded as a post-obligation in UCON. We argue that these obligations are not enforceable with original core UCON models which focus on a single usage session. We consider the scenario that the user has further service access requests after the current usage session, which is the natural and typical usage pattern in real world. Thus, the obligation can be enforced, e.g., by providing different quality of service to the user if the obligation is not satisfied. In the above eHealth usage scenarios, typically the client (e.g., a doctor) needs to access patients' records frequently, (e.g., everyday); therefore the obligations can be enforced. On the other side, we believe that authorizations and conditions are single-session oriented naturally, which means that pre- and ongoing authorizations and conditions are expressive enough to specify most (if not all) usage control policies.

As aforementioned, UCON has the novel features of decision continuity and attribute mutability. At the same time it can support many traditional access control policies in one model. However, not much work has been done to fully utilize UCON properties in a practical framework. The fundamental problem is that UCON enforcement mechanism lacks a general ongoing and post decision checking and enforcement engine. This drawback keeps UCON far away from practical and real systems. To cope with this problem, our second novel contribution in this work, is to introduce a comprehensive usage control enforcement engine. The corresponding enforcement engine is configurable using a proposed enforcement-focused usage control policy specification. A proof-of-concept prototype of the engine is implemented and tested for an eHealth application scenario. This implementation is based on XACML that has established itself as a de-facto mature standard for access control. Doing so the well standardized and implemented XACML access control policy and enforcement architecture are utilized.

We develop our work by following the recently proposed PEI security engineering framework [23]. In a security system, PEI distinguishes the problems of *what* the security requirements and *how* these requirements can be satisfied or enforced with different model layers. Particularly, in the *policy model* layer, we augment UCON core models with post-obligation considerations and extend the state transition of UCON (cf. Section 4); in the *enforcement model layer*, we distinguish two main functional modules to handle attributes and actions in a UCON system (cf. Section 5); finally, in the *implementation model* layer, we consider different implementation variants and develop our strategy based on standardized XACML policy specification and enforcement engine (cf. Section 6).

Outline: Background and related work are present in Section 2. We discuss our model extensions in Section 4. Regarding the enforcement model, we introduce a general continuity-enhanced usage control enforcement framework in Section 5. In Section 6 we propose the implementation model and discuss our proof-of-concept prototype and show how it leverages the capabilities of our enforcement model. We conclude this paper and present our ongoing work in Section 7.

2. RELATED WORK

2.1 UCON_{ABC} Model

Park et al. [15, 14, 16] systematically treat the usage control concept and develop a comprehensive model. The introduced $UCON_{ABC}$ model integrates Authorization (A), oBligation (B) and Condition (C) components in usage control decisions. Because of its session-based nature it has two remarkable features that distinguish it from the traditional access control models: *decision continuity* and *attribute mutability*. Given that, the model is divided into three core models: (1) Authorizations, where the access decision is made based on the requesting subject's and target object's attributes. (2) Obligations, in which the fulfillment of some actions by specific subjects on specific objects (the obligation's subjects and objects may differ from the authorization's pair) should be checked. (3) Conditions, where environmental information (also called environment attributes) are checked. To introduce the two main features in the model, it is further classified according to decision continuity and attribute mutability. With respect to the continuity factor, it can be distinguished between decisions made before an access has been started *(pre-)* or the decisions that are taken during the access session *(on-)*. While considering the mutability factor, update actions are introduced before, during, or after an access session. Our policy model is based on the proposed UCON model by Park et al. with the extensions in core model to include decisions that are made after an access session, i.e. *post-* sub models and augment the expressiveness of obligation to include system based obligations. Furthermore, we investigate obligations in more details and introduce general obligation model.

Zhang et al. [29] have defined a formal model and policy specification for $UCON_{ABC}$ based on an extension of the Temporal Logic of Actions (TLA). Figure 1 shows the system's state transition of a single usage [29], which distinguishes actions done by the system (*tryaccess* and *endaccess*) and actions done by a subject (the others).

The following states have been defined in original UCON

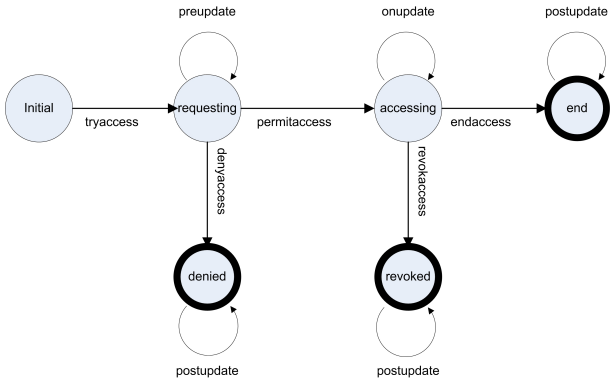

Figure 1: Traditional UCON state transition diagram.

in a usage process: initial, requesting, accessing, denied, revoked and end. Initial state means that access request is not generated; requesting state indicates that the access has been generated and is waiting for the system's usage decision; denied state refers to the state where the system has denied access; the accessing state means that the system has permitted access and the subject is accessing the object immediately after; the termination of the access done either when the system revokes the access after it has been granted in requesting state or it is ended normally by the user.

We identify one problem of this state transition, shown in Figure 1, related to ongoing decision checks. UCON supports decision continuity, which means that during a usage session multiple ongoing checks can occur. However this state transition diagram does not show the ongoing checks and the ongoing transitions. Hence, the state transition is extended by adding one state and showing the ongoing trigger actions (cf. Section 4). The extended state transition is used in designing our enforcement framework.

2.2 Obligations and Enforcement Framework

Pretschner et al. [17, 18] deal with distributed usage control policies, i.e. the loss of control over a data item after giving it away. Their work distinguishes "observation based enforcement" vs. "enforcement by direct control over the service provider action" by using compensating actions. Their work maps to the post obligations in our model. However, they do not consider usage sessions and related aspects. The policy proposed is at abstract level and no practical approach to leverage the enforcement engine is discussed.

Hilty et al. [12, 11] deal with usage control requirements with respect to obligations and conditions. Two kinds of obligations are defined: usage restrictions and action requirements. They express mandatory actions that must be executed either unconditionally or after a specified usage has been performed. From the policy model point of view, conditions and usage restrictions can be mapped to UCON core models of obligations, authorizations, and conditions together. Furthermore, action requirements can be covered in our extended UCON model by adding post obligations. They argue that their policy language can be mapped to DRM (Digital Right Management) policies. However they do not provide a configurable enforcement engine that utilizes their policy language apart from DRM systems.

Irwin et al. [13] define formal model of obligation, and distinguish secure and insecure system states based on the concept of *accountability*. The paper does not mention how to check obligation fulfillment, rather it deals with the formal specification and analysis of obligations and explains the complexity of checking accountability properties. The formal model is similar to our obligation model. We assume that the obligation's subject is the only entity responsible for the fulfillment of an obligation without any further obligation interferences. Additionally, we propose a comprehensive enforcement model for the obligation-extended UCON, and test it on a prototypical enforcement engine.

Gama et al. [8, 9] worked with an obligation enforcement engine. In the first paper they developed *Heimdall* as an enforcement platform for obligations using an obligation-enabled policy language xSPL, an extension of SPL [19]. In the second the authors showed the feasibility of their engine by integrating Heimdall with practical grid platform for resource usage management. In this paper reevaluation mechanism was added by extending xSPL. Comparing to our approach, additionally to supporting obligations in the context of a usage session and realted aspects of compensating actions and re-evaluation , we propose a new concept of obligation PDP. We believe that some kind of untrusted obligations need a fulfillment check upon receiving the obligation event. For example, the obligation that a user provides her valid email address requires that the enforcement system checks the validity of the address upon receiving it. Furthermore, tracing and recording a sequence of history events related to a specific usage of a resource is better achieved by state machine based engine.

Various other obligation concepts have been introduced in literature. In the Ponder policy language [7], an obligation is more like a duty that a subject has to perform. This obligation is not directly required by an access control purpose. Bettini et al. [6, 5] introduce the concept of *provisions* that should be done before an access is granted, and *obligations* that must be done after an access has been granted. These can be regarded as pre-obligations and post-obligations in UCON, respectively. However, they also do not propose an enforcement engine to support their policies.

With regard to UCON model, Zhang et al. [28] use XACML as a policy language and extend the XACML policy enforcement architecture with some extra components like "Usage Monitor" to monitor the updates of subject, object, and environment attributes. This framework lacks obligations handling, does not consider updates for denied and revoked states, and finally, does not distinguish between constraints applied in different states.

3. MOTIVATING EXAMPLE

In order to illustrate some of our framework's functionality, we take an application scenario from the medical domain (cf. the main requirements illustrated in Section 1). In this scenario, a patient's electronic health record (EHR) is stored and managed by a distributed Hospital Information System (HIS). Different actors are trying to get access to this record (or part of it), such as insurance companies, general practitioners, pharmacists, research institutes and others. Let us take the general practitioner (GP) as an example. In this scenario the usage requirement of our specific healthcare system states that patient attendance should be checked at the beginning and during a treatment (according to the 4-eyes principle). At the end of the treatment session, the patient

admission should be given before the GP saves the patient's record permanently; i.e. the patient has accepted to trust this GP for future treatments, or the record must be deleted. This is because of the fact that the record should only be stored by trusted and personal GP of the patient. In more details the work flow is as follows:

1. First, the doctor should have installed a document reader with UCON enforcement mechanisms.
2. While the patient is visiting the doctor for treatment, the doctor requests the document from the HIS (or its web portal).
3. At the HIS's side (service provider), the doctor is authenticated and authorized to access to the document, for example, based on his role in the system.
4. Upon authorization, the required document is released with the corresponding UCON policy in one encrypted package.
5. At the doctor's system, the document and the UCON policy are verified (for integrity), decrypted, and stored temporarily.
6. The UCON policy includes the usage requirements mentioned before. When the doctor tries to use the document, the policy is enforced by the enforcement mechanism that we are proposing in this work. This occurs in such a way that: (1) The "4-eyes principle" is checked before and during the usage session; (2) The document must be stored in the doctor's machine under the consent of the patient at the end of the treating session, otherwise the document must be deleted.

Note that we only describe here one instance of the work flow scheme. Variants exist in many implementation aspects such as object and policy distribution, integrity verification, and so on. For example, a UCON policy can be defined based on the type of healthcare records (e.g., according to different diseases), such that individual records and policies are distributed separately, and a doctor only needs to download a single copy of the same type of records.

Privacy is another example requirement applicable to usage control policies: each patient presents her/his preferences in privacy policy. Such preferences can include the retention time, the location, or the purpose of the usage [27].

4. OBLIGATION MODEL AND UCON EXTENSIONS

As aforementioned, the main drawback of the original UCON model is the inability to handle actions after the usage of the resources in a specific session [1]. In this section we deal with this problem to enable the model to handle this requirement by introducing post-obligations in UCON. Instead of simply introducing a *postB* core model, we consider how to specify obligations (including pre- and ongoing obligations in UCON) in a general and formalized way. We then extend the original UCON state transition scheme by exploring more detailed obligation trigger actions.

[1]The original UCON core models can enforce update actions after a single usage session– post-updates, which are *system actions*–performed by systems [29]. This is different from post-obligations that we discuss in this paper, which include actions that must be performed by subjects and were regarded as non-enforceable in original UCON model.

4.1 Obligation Model

An obligation in UCON is a predicate that utilizes some functions to check whether a certain activity has been fulfilled or not. For example, before downloading a white paper of a company, a user is required to give her/his valid email address and accept the privacy terms and conditions of the company. The *obligation subject* in this case is the same subject that tries to access the object. In general, an obligation subject can be any other subject, even the security system itself, e.g., the reference monitor and supporting components of a system. For example, watching an online video requires that streaming video data is stored in a temporary folder, and is required to be deleted after being watched. This obligation should be performed by the system (e.g., the media player or DRM agent that enforces usage policies) on the video data (*obligation object*). The current UCON model includes two obligation core models, namely *preB* and *onB*, where the first is the one applied before the access and the second during the access. In this paper, we generalize the requirement of obligations, which include actions that must be performed by a subject or the system and the fulfillment can be checked after the access.

Taking the mentioned healthcare scenario into consideration, the client software should delete the record in the revoked state or store the record in the end state when the patient gives her/his permission, where the deleting and storing of the record are considered obligations that should be done and the "fulfillment" should be checked by the system after the end of the session.

In general the purpose of post obligations is twofold: First, it can be used to execute obligation actions that are related to the current usage despite it has no affect on the decision making of the current usage. However these actions are required to be executed and a notification of the fulfillment of these action can be send to the service provider of the data owner (e.g., storing the patient's record after normal ending of a treatment session). Secondly, it can affect future usage sessions, which can be done, for example, by sending a request to the usage control policy repository to change the UCON policy related to this subject when trying to access this object or any other objects in the future.

We consider an obligation from four points of view: (1) *Who* must perform the action; (2) *What* the obligation must be applied to; (3) *When* it should be performed; and (4) For *how long* it should be carried out.

- *Who* means which entity is obligated to fulfill the action. Here we distinguish system-performed obligations and subject-performed obligations. This distinction is important because of the fact that typically a security enforcement system is trusted while we do not have any influence on other subjects. Doing so we are augmenting the expressiveness of obligations in original UCON, where updates are considered as the only actions that must be done by the system.
- *What* refers to the obligation object. Here we can distinguish objects that the system controls and objects that are out of the control of the system. *Controllable objects* are those that are within a target system's

domain, while *non-controllable objects* are outside the system's domain. For example, deleting a file that is controlled by the system differs from deleting a file that is stored in a remote file server. In the latter case we should commit additional actions to ensure that the file server is up, the file exists, the network connection is working, and the file is really deleted after the obligation action.

In conjunction with the aforementioned subject/system distinction, we can conclude the following obligation classifications: *system obligations on controllable objects*, *system obligations on non-controllable objects* and *subject obligations*. We argue that the first obligation type does not need fulfillment check, as it is done by a trusted system on controllable objects. However the second and third types need fulfillment check, either because the object is not controllable or the subject is not trusted. Updates can be classified as system obligations on controllable objects, in case the updated objects are controllable. Hence, updates are considered in the original UCON model as actions rather than a predicate because they do not need further check. Nevertheless, it has been mentioned that when a target object is not available, e.g., because of network problem or storage problem, the update process should be monitored without capturing these aspects in the model [29]. In our approach we are dealing with this issue systematically by this classification. Therefore we are considering the updates as a special type of system obligations, either on controllable or non-controllable objects.

- *When* means whether the obligation should be fulfilled: before a usage control session is finished (e.g., before the access or during the access) or after the session is ended. It is important to distinguish between obligations after the usage session and obligation before the session is finished. This is crucial because after the usage session is finished, the result of the obligation can not affect any more the usage decisions, while it does before the access is granted or during the usage. *Note that the post obligation can affect the usage decisions of future sessions.*

- *Durability* means that the obligation should be fulfilled within specific period of time. For example, a user must pay his fee within one month, and whenever he pays he can get access to offered services. In this case the fulfillment should be checked continuously to start the service. In some cases there is a need for instant fulfillment check. For example, when a user must give his valid email address before requesting the white paper of a company, the system should check the validity of the email instantly after the request is generated and before the user is given the access. Similar aspects are also mentioned by Hilty et al. [11].

Based on these considerations, we define an obligation as a tuple of $OBL = (OBS, OBO, OBA, WHEN, DURATION)$, where OBS is the obligation subject or system, OBO is the obligation object (e.g., healthcare record), OBA is the action that has to be performed (e.g., delete), $WHEN = pre|on|post|$, and $DURATION$ is the time point or period to check the fulfillment of the obligation. The concept of fulfillment check and compensating actions associated with each obligation will be discussed in the implementation model.

4.2 State Transition of UCON

Figure 1 shows the original UCON state transition schema, in which pre-decision components (preA, preB, preC or any combination of them) are evaluated at the *requesting* state. Once the access request is granted, the system comes to the *accessing* state, in which the subject is accessing the object. From the *accessing* state the system moves either into *end* state when the users ends the session, or *revoked* state when policy rules are violated during the session. We argue that the original UCON does not show the actions that trigger an ongoing decision check during the usage session. The state in which the system is checking the policy rules during the session, analogy to the *requesting* state, is not mentioned as well. That means, the accessing state and the state where the ongoing policy rules are checked are merged in one state called *accessing*, and the trigger action that triggers the re-evaluation of attributes is hidden.

To express these subtle but critical state transitions in UCON, we divide the *accessing* state in original UCON into two states, namely *accessing* and *ongoingCheck*, respectively. A trigger action transits the system state from *accessing* to *ongingCheck*.

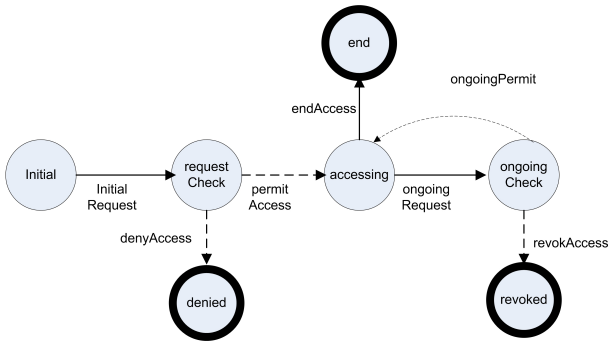

Figure 2: Expanded UCON state transitions.

Figure 2 shows our extended UCON state transition scheme. Comparing to the state transitions shown in Figure 1, we omit in our new state transition diagram the update actions/transitions: We consider updates as system obligations (as discussed before). That means, the needed updates are stated in the obligation rules of the corresponding state. When the object of the update obligation is controllable, there is no need for fulfillment check. This case maps to the original update actions in [14, 29].

Secondly, one new state, *ongoingCheck* state, and two transitions, *ongoingRequest* and *ongoingPermit*, are added in our new scheme. When the subject is exercising the access to the resource, the system is in the *accessing* state. Any changes or updates of subject, object, or environment attributes trigger *ongoingRequest* transition and the system moves to the *ongoingCheck* state. In this state the decision (onA, onB, onC or any combination of them) is made and the new set of attributes should be evaluated. Accordingly, either the system revokes the usage, i.e., *revokAccess* transition, or it continuously grants the access to the subject, i.e., the *ongoingPermit* transition, and the system goes back to

the *accessing* state. Note that any updates occur in the *onglingCheck* state will have no direct effect on the current evaluation. The new updates will be checked in *accessing* or *revoked* states later.

5. ENFORCEMENT MODEL

In PEI framework, the enforcement models focuses on the system architecture and functional modules to illustrate how the policy model can be achieved, not being restricted to a particular system. In this section we present a comprehensive usage control enforcement model with the corresponding meta-model. The proposed architecture of a configurable enforcement engine considers all UCON core models and the extensions proposed in our work.

Conceptually, an enforcement model includes decision and enforcement functionalities [1]. However, the session nature and decision continuity feature of usage control policy model require additional components. Furthermore, obligation handling is not considered in the existing access control enforcement [1] as it is not part of the traditional access control models. Given that, we are introducing a usage control specific enforcement model.

5.1 Usage Control Enforcement Model

Figure 3 shows our enforcement model consisting of three main components: *enforcement point (EP)*, *decision point (DP)* and *session management point (SMP)*.

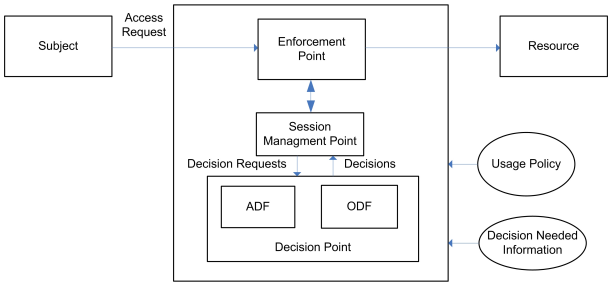

Figure 3: Usage control enforcement model.

The DP is responsible for making required decisions during a usage control session. Introducing obligations into the model requires a new decision-making component, i.e., a component that decides whether an obligation is fulfilled or not. Hereby, the DP in our enforcement model consists of the following two sub-modules:

- *Attribute decision function (ADF)* handles the attribute-based access decision during a usage session. Attributes can be either subject, object or environment attributes. The *decision needed information* provides the DP with required information about the subject, object, and environment.

- *Obligation decision function (ODF)* makes the decision whether a specific obligation has been fulfilled. According to our classification of obligations mentioned in Section 4, we have three types of obligations: *system obligations on controllable objects*, *system obligation on non-controllable objects*, and *subject obligations*. It is argued that the first type of obligations does not need fulfillment check. for the other types, the DP checks the fulfillment of an obligation by transforming it into an ordered sequence of system actions, which should be defined for all obligations. We consider these aspects in designing the ODF component in our implementation model (cf. Section 6.2).

The SMP is the element that manages individual usage sessions. This includes requesting required decision(s) from suitable modules in each state during the usage session.

When a usage request is received, the EP forwards it to the SMP. The SMP sends corresponding pre-decision requests to the DP. In case a negative decision is received, the SMP sends a denied response to the EP and requests post-obligation actions from the ODF (including post updates), otherwise it sends a permit response to the EP. In the *accessing* state, the SMP monitors related subject, object, and environment attributes, as well as any further actions requested by the subject (e.g., an *end* request from the subject to end the usage session). Upon receiving such trigger actions, the SMP requests ongoing decision checks. According to the decision received, the SMP either revokes the ongoing access by sending a *revoke* response to the EP, or keeps permitting the access. If the action is an *endaccess* action from the subject, the SMP sends an *end* response to the EP to stop the access. In revoking or ending a session, the SMP requests the post-obligation fulfillment check with the corresponding obligation rules.

As aforementioned, updates are considered as special type of system obligations. Hence, the pre-, ongoing, and post-updates of UCON policy are stated in the corresponding obligation rules in our enforcement model.

5.2 Enforcement Meta-model

As enforcement model provides the system architecture in an abstract level, a general configuration mechanism is desired for system designer or administrators to customize the development and deployment of UCON in a real system. For this purpose we propose a meta-model for UCON enforcement which is able to configure the enforcement engine with rules needed for each state in usage sessions. In other words, a meta-model aims to include information related to all kinds of rules/constraints defined in the *Decision Point* component of an enforcement model (cf. Figure 3) in the context of usage session (cf. Figure 2). As shown in Figure 4, the core elements of the meta-model are the ADF_Rules and ODF_Rules. ADF_Rules are the rules representing ADF function of the enforcement model, i.e. the authorization and condition predicates of a UCON policy. The ODF_Rules on the other hand are obligation rules representing ODF functions of the decision point. For better understanding an instance of this meta-model is presented in Section 6 for configuring the developed engine as an XML schema.

As shown in the state diagram of Figure 2, each state contains some authorization related decisions that must be taken. In the *requestCheck* state, the preA and preC decision constraints (defined in ADF_Rules) and preB decision rules (defined in ODF_Rules) of the policy are evaluated. The same evaluations are made in the *ongoingCheck* state but for the onA, onC and onB decision components of the policy. On the other hand, it can be argued that there is no need for attribute related authorization checks in the *end*, *revoked* and *denied* states, thereby they contain only postB decision components.

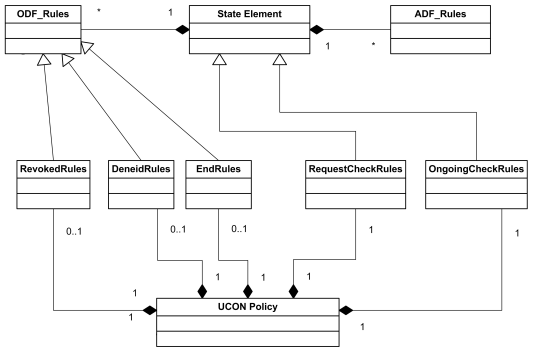

Figure 4: UCON enforcement meta-model.

As Figure 4 shows, *StateElement* type contains elements of both "ADF_Rules" and "ODF_Rules" types. The rules of the *requestCheck* and *ongoingCheck* states are of type "StateElment". However, the rules of the rest of the states are of the ODF_Rules type. Finally we organize the different rules in each state in one UCON policy element that includes the following rules: *OngoingCheckRules*, *RequestCheckRules*, *RevokedStateRules*, *DeniedRules* and *EndRules* for the *ongoingCheck*, *requestCheck*, *denied*, *revoked*, and *end* states, respectively.

6. IMPLEMENTATION MODEL: ENFORCEMENT ENGINE AND PROTOTYPE

To show the feasibility and functionality of our usage control enforcement model, we design a concrete usage control enforcement engine. We then develop and test a proof-of-concept prototype. Furthermore, in order to utilize and facilitate the existing standards and frameworks in the area of access control, XACML is extended and used in this prototype.

Two main aspects should be considered when implementing a usage control enforcement engine in remote clients: The first issue is the *trust* problem and second is the *configurable usage control enforcement mechanism*. Given that the usage enforcement should be done in a remote client platform, we have no control over the remote system. That requires trust establishment between the service provider and the remote client before any data is released. Trusted computing technologies provide the technical underpinning for trust establishment between remote machines [24, 20, 21]. While the trust issue goes beyond the scope of this paper, we assume that the remote client is trusted before the usage control policy enforcement takes place. A general configurable usage control enforcement model has been presented in section 5 and a prototypical implementation of that model is discussed in this section.

6.1 Prototype Architecture

In our eHealth scenario, when a patient visits a doctor, the doctor requests the patient's EHR from the HIS. After authenticating and authorizing the doctor based on his role, the HIS releases the record and a UCON policy in one encrypted package. The enforcement component, which is integrated into the document reader, checks the integrity of the package and extracts the usage control policy and the patient's record.

Figure 5 shows the architecture of our usage control enforcement engine. The implementation is based on XACML enforcement engine. The elements with white background represent the original XACML enforcement engine elements, while those with the grey background represent our extensions. In general the engine consists of the following components:

- *Policy enforcement point (PEP)* acts as single entry point to protected resources and performs access control. It receives usage requests from an access requester (the subject), i.e., the doctor in our example, and makes a *usage decision request (UDQ)* and consequently receives *usage decision response(s) (UDS)* from the *finite state machine (FSM)* element during the whole usage session. The PEP enforces the authorization decisions it receives by either allowing the access or denying it. It is important to note that we integrate the PEP in our prototype in the document reader. Without loss of generality, the document reader in our prototype is a simple notepad-like application. The design of PEP depends on the application and the target architecture. For example, it can be implemented as a security gateway in SOA system [3] or as a bundle in OSGi architecture [2].

- *Finite state machine (FSM)* represents the session management point in our enforcement model. It is the dynamic part of the whole engine and capture the continuity behavior of the access control system. Furthermore, it orchestrates the functions of other elements of the architecture and ensures the transitions from one state to another according to the extended UCON states illustrated in Figures 2. It can be configured using an instance of the enforcement meta-model showed in Figure 4. Initially the FSM is in the *initial* state. Upon receiving a UDQ request from the PEP, the *initial-request* transition is triggered and the FSM moves to the *requestCheck* state. In this state the FSM requests authorization and obligation decisions from the PDP_L1 and the PDP_L2, respectively. According to the corresponding rules specified in the *RequestCheckRules* of the meta-model, the decision points (PDP_L1 and PDP_l2) make their decisions. Deny responses received from PDP_L1 or PDP_L2 trigger *denyAccess* transition and the FSM moves to the *denied* state. On the other hand, permit responses from both decision points trigger the *permit* transition and the FSM moves to the *accessing* state. In the *denied* state FSM checks the obligation (postB) rules specified in the *DeniedRules* of the meta-model and sends a deny response to the PEP to deny the requested access. Finally it goes back to the initial state waiting for new session requests. This sequential session nature of the engine make post obligations proposed in this contribution enforceable, as their result can affect future usage sessions. Similarly the rest of the states are managed by the FSM.

- *Policy decision point level 1 (PDP_L1)* refers to the *ADF* function in our enforcement model and is represented as an XACML PDP. It is the component that evaluates attribute related constraints (authorizations and conditions) and renders decisions to the FSM.

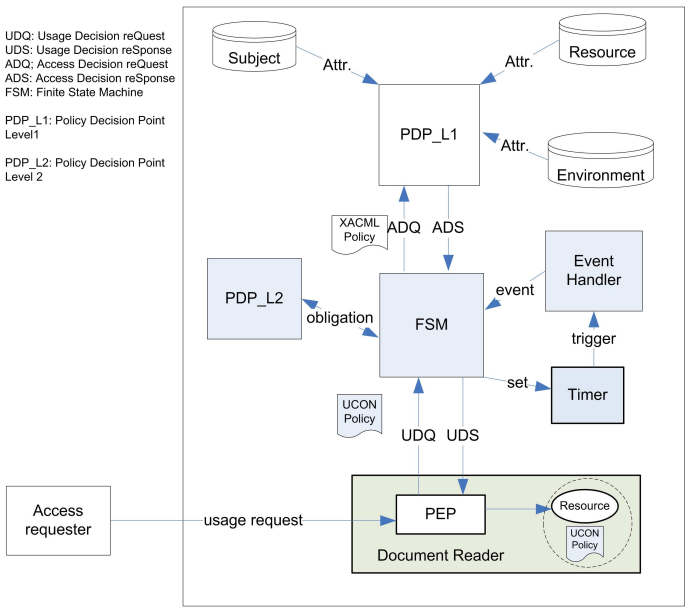

Figure 5: Implementation architecture with finite state machine.

- *Policy decision point level 2 (PDP_L2)* refers to the *ODF* function of our enforcement model. It receives an obligation request from the FSM and checks whether the obligation has been fulfilled. As already mentioned, we classify obligations according to two factors: The executor of the obligation (system vs. subject) and the object controllability. The system obligations on controllable objects do not need fulfillment check, while other obligations must be checked. The check can be done by transforming the obligation into a sequence of system actions. We describe the details of this obligation check mechanism in next subsection.

- *Event handler* handles the events that trigger transitions from one state to another. The transitions of a UCON state diagram can be classified into two classes: *internal transitions* indicated by dashed arrows in Figure 2 and *external transitions* indicated by solid-line arrows. Internal transitions are transitions that are triggered by internal events, while the external ones are triggered by external events. The internal events are the events that result from the decision responses received form the decision points. *denyAccess*, *permitAccess*, *ongoingPermit* and *revokeAccess* are examples of internal events because they are raised according to deny/permit decisions. All other events are considered as external events, like the usage request from the subject, the timer trigger, and attribute updates. The event handler entity is responsible to listen to the external events and send trigger actions to the FSM.

- *Timer* handles temporal conditions. A timer can be set by the FSM or PDP_L2 and it triggers an event through the event handler. The Timer can be used for supporting obligations with deadlines and obligation reevaluation.

This architecture represents the implementation model in our PEI framework and provides full configurable continuity-enhanced usage control enforcement engine. The FSM leverages the continuity feature and orchestrates the functions of other components during a usage session. While PDP_L2 facilitates the obligation handling, the possibility to track multi-sequential sessions makes post-obligations enforceable. This is due to the fact that post-obligations can have impact to future usage sessions, e.g., by declining the next usage request from the same subject or offering different quality of services.

6.2 Design of Obligation PDP

Base on our proposed policy model, we conclude that obligations are either obligations that need fulfillment check (we call these obligations *non-trusted obligations*) or obligations that do not need, which is called *trusted obligations*. *Trusted obligations* are the ones that must be fulfilled by our trusted system on controllable objects. We argue that this distinction is very crucial in designing the obligation PDP (the PDP_L2 in Figure 5) as these two types should be handled differently. Trusted obligations are simply system actions that returns *permit* response always. On the other hand, we can further distinguish between instance obligations that need instant fulfillment check, and obligations that must be fulfilled within a specific period of time (deadline) (cf. Section 4). An *instant non-trusted obligation* is treated using PDP_L2 by transforming the obligation into an ordered sequence of system actions that include a condition at the end. Accordingly, the obligation decision response is created. For example, the obligation *obl1(s1,o1,provide,t0,0)*: that a user *s1* must *provide* his **valid** email address *o1* instantly can be transformed into the following actions (please note that the obligation fulfillment check includes insuring that the email address is valid): (1) check that the address is in the form of xxx@xxx.xxx; (2) send a testing email to that address; (3) wait for 2 minutes; (4) check the inbox of the sending email account for new error email; (4) if an error notification email is received which indicates that the user email is not valid, send back *deny* response; (5) otherwise send

permit response. Similarly, obligations with deadlines can be treated with the help of other components like the timer and some special environmental attributes and variables.

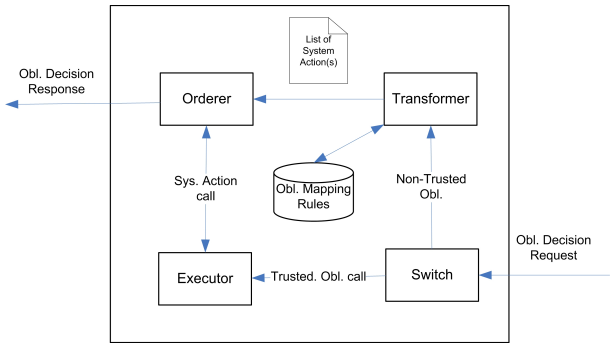

Figure 6: Check obligation fulfillment with obligation PDP.

Figure 6 shows the proposed design of the obligation PDP. When a request comes to the decision point, a switch separates the *trusted* and the *non-trusted* obligations. The trusted obligations are then sent directly to the *executor* for execution. The non-trusted obligations are transformed by the *transformer* into a set of ordered system actions using the specific rules stored in the *obligation mapping rules* repository. The list of system actions is sent to the *orderer* which ensures that the actions are going to be executed by the *executor* in order and finally sends the response back according to the replies it receives from the executor. The obligation mapping rules repository is extendable and configurable that maps between the actions that the executor can commit and the obligations that are needed for each policy. It is important to notice that the permission check, according to our enforcement model (cf. Section 5), whether a subject is allowed to fulfill an obligation or not, is done by the *ADF* component, i.e. PDP_L1 in Figure 5.

Compensating actions that must be committed in the *denied* or *revoked* states are considered system obligations (like attribute updates) . Hence, the obligation PDP is responsible for checking the fulfillment of untrusted obligations as well as for enforcing them.

6.3 Policy Specifications

Figure 4 shows the meta-model for UCON policy that can configure the enforcement engine. Based on this meta-model and XACML policy specification, we have developed a UCON policy specification. In this specification, ADF_Rules are substituted by the XACML *policy set*, while the ODF_Rules are represented by *StateAction* elements in XACML. The following shows our developed UCON policy schema based on the XACML schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema >
 <complexType name="StateActionType">
  <sequence minOccurs="0">
   <element ref="ucon:StateAction"/>
  </sequence>
 </complexType>
 <element name="RevokedPolicy" type="ucon:StateActionType"/>
 <element name="DeniedPolicy" type="ucon:StateActionType"/>
 <element name="EndPolicy" type="ucon:StateActionType"/>
 <complexType name="StatePolicy">
  <sequence>
   <element ref="ucon:StateAction" />
   <element ref="xacml:PolicySet"/>
  </sequence>
 </complexType>
 <element name="RequestcheckPolicy" type="ucon:StatePolicy"/>
 <element name="OngoingcheckPolicy" type="ucon:StatePolicy"/>
 <complexType name="UCONPolicyType">
  <sequence>
   <element ref="ucon:RequestcheckPolicy"/>
   <element ref="ucon:OngoingcheckPolicy"/>
   <element ref="ucon:DeniedPolicy"/>
   <element ref="ucon:EndPolicy"/>
   <element ref="ucon:RevokedPolicy"/>
  </sequence>
  <attribute name="UCONPolicyId" type="anyURI" use="required"/>
 </complexType>
 <element name="UCONPolicy" type="ucon:UCONPolicyType"/>
 <element name="StateAction" type="ucon:StateActions"/>
 <complexType name="StateActions">
  <sequence>
   <element ref="xacml:Obligations"/>
  </sequence>
 </complexType>
</schema>
```

This schema specifies all elements of a UCON policy. Specifically, the *UCONPolicy* represents the root element of the policy, which consists of different rules for each state. These rules have the same name of the state with the suffix *policy*, e.g., the rules of *requestCheck* state is *RequestCheckPolicy*. *RequestCheckPolicy* and *OngoingCheckPolicy* are of *StatePolicy* type, which contains XACML's *PolicySet* and *StateAction* elements. *DeniedPolicy*, *EndPolicy*, and *RevokedPolicy*, on the other hand, contain only *StateAction* elements, which represent the obligations in the ODF_Rules.

Based on this schema, a UCON policy can be created and used to configure our developed enforcement engine, to cope with the requirements discussed in Section 3. Due to space limit, we omit the full XACML policy specification here.

7. CONCLUSION AND FUTURE WORK

Usage Control meets new security requirements that go beyond the capabilities of traditional access control systems. Such requirements are identified in eHealth in our project. To leverage the capabilities of usage control system in real applications, we have developed a UCON-based solution for a secure healthcare information system by following PEI framework. First, an extended version of UCON_{ABC} model is proposed as a formal policy model. Secondly, a continuity-enhanced usage control enforcement model with related specification of configuring meta-model is introduced. This model consists of two sub-modules(*ADF* and *ODF*). The first is responsible for authorizing a subject to do some actions and the second checks the obligation that the subject must fulfill. Hence, the relationship between usage/access control and obligation is clearly defined. without subordinating any aspect to the other, which is the case in most related work in both areas. Finally, a proof-of-concept prototype is implemented and a UCON policy schema is developed and tested in the context of our healthcare use case. Performance evaluation and different use cases will be considered in future work.

Considering the implementation model, we integrate the PEP in the document reader application. This approach is not flexible in real world. On other hand, the more we approach the hardware layer the more secure and trusted the system is. Hence, designing a general PEP in kernel space and bridging the gap between kernel and user space semantics is another future research direction. Trust management

protocols can be investigated as well to facilitate trust establishment between remote systems. Finally, the feasibility of using our enforcement architecture as history-based policy engine by adding additional components for storing related session's information is an open issue.

8. REFERENCES

[1] [ACF]ITU-T Rec X.812 (1995) | ISO/IEC 10181-3:1996. Security frameworks for open systems: Access control framework. Technical report, 1996.

[2] B. Agreiter, M. Alam, R. Breu, M. Hafner, A. Pretschner, J.-P. Seifert, and X. Zhang. A technical architecture for enforcing usage control requirements in service-oriented architectures. In *Proc. ACM workshop on Secure web services*, 2007.

[3] M. Alam, M. Hafner, M. Memon, and P. Hung. Modeling and enforcing advanced access control policies in healthcare systems with sectet. *Mothis*, 2007.

[4] C. Bettini, S. Jajodia, X. SeanWang, and D. Wijesekera. Provisions and obligations in policy rule management. *J. Network and System Mgmt.*, 2003.

[5] C. Bettini, S. Jajodia, X. Sean Wang, and D. Wijesekera. Obligation monitoring in policy management. *IEEE 3rd Intern. Workshop on Policies for Distributed Systems and Networks*, 2002.

[6] C. Bettini, S. Jajodia, X. Sean Wang, and D. Wijesekera. Provisions and obligations in policy management and security applications. In *Proc. of the 28th VLDB Conference ,Hong Kong, China*, 2002.

[7] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. *Lecture Notes in Computer Science*, 2001.

[8] P. Gama and P. Ferreira. Obligation policies: An enforcement platform. In *Sixth IEEE International Workshop on Policies for Distributed Systems and Networks*, 2005.

[9] P. Gama, C. Ribeiro, and P. Ferreira. A scalable history-based policy engine. In *Seventh IEEE International Workshop on Policies for Distributed Systems and Networks*, 2006.

[10] M. Hafner, R. Mair, R. Breu, B. Agreiter, S. Unterthiner, and T. Schabetsberger. Health@net. Die verteilte elektronische gesundheitsakte- eine fallstudie in modell-getriebenem security engineering. *IT-Sicherheitskongress des BSI*, 2007.

[11] M. Hilty, D. Basin, and A. Pretschner. On obligations. In *Proc. of European Symposium on Research in Computer Security*, 2005.

[12] M. Hilty, A. Pretschner, D. Basin, C. Schaefer, and T. Walter. A policy language for distributed usage control. In *Proc. of the 12th European Symposium on Research in Computer Security*, 2007.

[13] Keith Irwin, Ting Yu, and William H. Winsborogh. On the modeling and analysis of obligations. In *Proc. of ACM Conference on Computer and Communications Security*, 2006.

[14] J. Park and R. Sandhu. The ucon_abc usage control model. *ACM Transactions of Information and System Security*, 7(1):128–174, 2004.

[15] J. Park and R. Sandhu. Towards usage control models: Beyond traditional access control. In *Proc. of ACM symposium on Access control models and technologies*, 2002.

[16] J. Park, X. Zhang, and R. S. Sandhu. Attribute mutability in usage control. In *Proc. of the Annual IFIP WG 11.3 Working Conference on Data and Applications Security,*, 2004.

[17] A. Pretschner, M. Hilty, and D. Basin. Distributed usage control. *Communication of the ACM , 49(9):39-44*, 2006.

[18] A. Pretschner, M. Hilty, F. Casati, and F. Massacci. Usage control in service-oriented architecture. In *Proc. of the 4th Intl. Conf. on Trust, Privacy & Security in Digital Business*, 2007.

[19] C. Ribeiro, A. Zuquete, P. Ferreira, and P. Guede. Spl: An access control language for security policies with complex constraints. In *Proc. of the Network and Distributed System Security Symposium*, 2001.

[20] R. Sailer, T. Jaeger, X. Zhang, and L. van Doorn. Attestation based policy enforcement for remote access. *ACM Conference on Computer and Communications Security*, 2004.

[21] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of tcg based integrity measurement systems. In *Proc. of the 13th conference on USENIX Security*, 2004.

[22] R. Sandhu and J. Park. Usage control: A vision for the next generation access control. *Inter. Workshop on Mathematical Methods,Models and Architectures for Computer Networks Security*, 2003.

[23] R. Sandhu, K. Ranganathan, and X. Zhang. Secure information sharing enabled by trusted computing and pei models. In *Proc. of ACM Symposium on Information, computer and communications security*, 2006.

[24] R. Sandhu and X. Zhang. Peer-to-peer access control architecture using trusted computing technology. In *Proc. of ACM symposium on Access control models and technologies*, 2005.

[25] S. Unterthiner, M. Hafner, R.Breu, and T. Schabetsberger. Endpoint security in elga architekturen. *eHealth-Medical Informatics meets eHealth. Vienna*, 2007.

[26] G. Vogt. Multiple authoriztion- a model and architecture for increased, practical security. In *Proc. of IFIP/IEEE Symposium on Integrated Network Management*, 2003.

[27] G. Yee, L. Korba, and R. Song. Ensuring privacy for e-health services. In *Proc. of The First International Conference on Availability, Reliability and Security*, 2006.

[28] X. Zhang, M. Nakae, M. J. Convington, and R. Sandhu. A usage-based authorization framework for collaborative computing systems. In *Proc. of ACM Symposium on Access Control Models and Technologies*, 2006.

[29] X. Zhang, F. Parisi-Presicce, R. Sandhu, and J. Park. Formal model and policy specification of usage control. *ACM Transactions on Information and System Security*, 8(4):351–387, 2005.