Chapter 1
UCON+: Comprehensive Model, Architecture and Implementation for Usage Control and Continuous Authorization

Ali Hariri, Amjad Ibrahim, Bithin Alangot, Subhajit Bandopadhyay, Antonio La Marra, Alessandro Rosetti, Hussein Joumaa, and Theo Dimitrakos

Abstract In highly dynamic and distributed computing environments (e.g., Cloud, Internet of Things (IoT), mobile, edge), robust access and usage control of assets is crucial. Since assets can be replicated in various locations on heterogeneous platforms and dynamic networks with unknown or partially authenticated users, the need for a uniform control mechanism is essential. The theory of Usage Control (UCON) is an example of such a mechanism to regulate access and usage of resources based on expressive polices and a loosely-coupled enforcement technology. However, in complex socio-technical systems, concerns about scalability, performance, modularity often arise, and existing UCON models and frameworks cannot meet such requirements. To tackle these concerns, we introduce UCON+, an improvement over existing UCON models, which adds continuous monitoring before granting and after revoking authorizations as well as policy administration and delegation. This chapter aggregates our recent contributions on the conceptual, architectural, and implementation level of UCON+, and provides a comprehensive reference to describe the current state-of-the-art and the novelties of UCON+.

Ali Hariri · Amjad Ibrahim · Bithin Alangot · Subhajit Bandopadhyay · Hussein Joumaa · Theo Dimitrakos
German Research Center, Huawei Technologies Düsseldorf GmbH, Munich, Germany
e-mail: firstname.lastname@huawei.com

Antonio La Marra · Alessandro Rosetti
Security Forge, Pisa, Italy e-mail: firstname.lastname@security-forge.com

Ali Hariri · Hussein Joumaa
Department of Information Engineering and Computer Science, University of Trento, Trento, Italy

Subhajit Bandopadhyay
School of Mathematics, Computer Science and Engineering, City, University of London, London, United Kingdom

Theo Dimitrakos
School of Computing, University of Kent, Canterbury, United Kingdom

1.1 Introduction

Access control has been used to protect data privacy and security. Many access control models exist to address different security requirements. They evolved from coarse-grained models such as Mandatory Access Control (MAC) [13], Discretionary Access Control (DAC) [12] and Role Based Access Control (RBAC) [14] to more flexible and fine-grained models such as Attribute Based Access Control (ABAC) [2]. ABAC provides higher level of granularity by managing access rights using policies that are based on attributes of subjects, resources and the environment. It also incorporates obligations and advice, which are respectively mandatory and optional actions to be completed upon evaluation of policies.

Cyber-physical systems and consumer Internet of Things (IoT) devices, together with an ever-expanding network of sensors and actuators, help interconnect people, appliances as well as Information and Communication Technology (ICT) resources pervading homes, vehicles, healthcare systems and many other aspects of human life. The continuity of interaction combined with the heterogeneity of devices, information, connectivity and automation creates a dynamic environment. Monitoring and controlling permissions, access and usage of information and resources in such environments is essential for maintaining quality, security and safety. However, existing Identity and Access Management (IAM) technologies and models (e.g., ABAC) fail to offer an effective and cost-efficient solution because they assume that access rights do not change during access, so they do not react to situation changes.

For this reason, we introduce a comprehensive model and a novel technology for continuous authorization. Our model builds on the Usage Control (UCON) model [10, 6], which extends ABAC with mutable attributes and continuous control. We designate this technology as UCON+ because it enhances UCON with the following novelties: (1) Extend continuous monitoring to cover interactions before granting and after revoking authorizations; (2) Enable involving auxiliary evaluators in the decision making process (e.g., trust level evaluation, threat intelligence); (3) Support policy administration and delegation as well as defining trust authorities and roots of trust; and (4) Leverage a lightweight policy language for ABAC and extend it with UCON novelties without modifying the syntax and semantics of the language. We also describe Usage Control System Plus (UCS+), a modular, scalable and lightweight implementation of UCON+. UCS+ is an robust dynamic authorization technology for embedded systems as well as cloud and Zero-Trust Architectures (ZTAs), benefiting them with:

- Policy-based and code-less behavior at the “brain” of such systems;
- Continuous monitoring of context, environment and data/resource access;
- Intelligent (algorithmic) combination of policies with manageable resolution of conflicts;

- Proactive decisions and interactions to improve user experience, optimize usage of apps and resources, mitigate security and safety risks;
- Automated response to change including revocation of access or restriction of privileges; and
- Ability to advice or prompt for input and involve humans in real-time decision making.

This chapter consolidates our recent advancements on UCON+ with our previous works presented in [5, 1, 4].

1.2 The UCON+ Model

The UCON model was introduced as a generalization that goes beyond ABAC by adding mutability of attributes and continuity of control. UCON was originally introduced by Park and Sandhu [10] focusing on the convergence of access control and Digital Rights Management (DRM), and extending them with context-based conditions. Unlike preceding models, UCON considers attributes to be mutable, and specifies that policies must be re-evaluated when attribute values change during usage of resources. Lazouski et. al [6, 3] introduced a particularly relevant flavor of UCON that preserves a full ABAC baseline model, complemented with an authorization context, continuous monitoring and policy re-evaluation as well as an implicit temporal state. The temporal state is captured by classifying policy rules as *pre*, *ongoing* and *post*, such that *pre* rules are evaluated before granting authorization, *ongoing* rules are continuously evaluated while auth orization is in progress, and *post* rules evaluated upon the end or revocation of authorization. The novelties of UCON enable it to monitor attribute values and re-evaluate policies upon changes in order to guarantee that access rights still hold whilst usage is in progress, or to revoke access if the security policy is no longer satisfied. This makes UCON an excellent baseline for dynamic environments and cyber-physical systems where contextual changes are frequent and authorizations are long-lived.

Continuous control in UCON is limited to the duration of active authorizations only (i.e., only when access has started and is in progress). For this reason, we introduce UCON+, an improvement of the UCON model that extends continuous control to cover interactions before granting and after revoking authorization. This allows UCON+ to cover use-cases that require continuous control and monitoring before and after authorization (e.g., allow subject to improve trust level before granting authorization, ensure that a smart vehicle stops safely upon revoking authorization). In addition, UCON+ enhances UCON with the ability to involve auxiliary decision factors such as trust level or threat intelligence. UCON+ also extends UCON with the Administration and Delegation Profile (ADP) [11] by incorporating administrative policies that define trust authorities and allows their delegation. In this section, we describe the UCON+ model as well as the policy language used to express UCON+ policies.

1.2.1 Model Components

The UCON+ model consists of five main components adopted and adapted from UCON [10, 7]. The five components are: Attributes, Phases, Rules, Decisions and Instructions. These components together compose authorizations as shown in Figure 1.1 and described as follows.

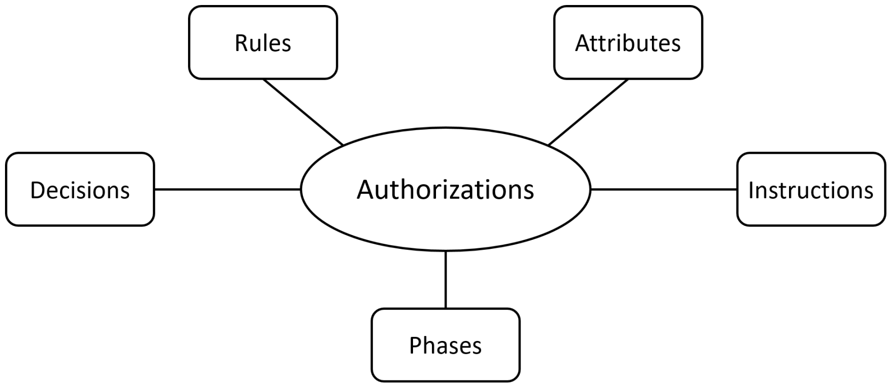

Fig. 1.1 UCON+ Model Components.

`Attributes` are properties of the subject (e.g., name), resource (e.g., file type) or environment (e.g., CPU load). Unlike UCON, we consider attributes of all categories (i.e., subject, object and environment) to be mutable, so their values may change while an authorization is in progress. An attribute value may change as a consequence of the authorization itself (e.g., updating metadata of a file upon granting authorization to access it) or due to contextual changes (e.g, subject location changes due to mobility). Since attributes are mutable, the security policy must be re-evaluated whenever an attribute value changes throughout the lifetime of an authorization.

`Phases` refer to the temporal state introduced by UCON and describe the stages of an authorization starting from initial request to enforcement to revocation. Like UCON, UCON+ defines three phases: (1) the *pre* phase indicates that and authorization has been requested but has not been granted yet; (2) the *ongoing* phase denotes that the authorization has been granted and is in progress; and (3) the *post* phase refers to the last stage of the the authorization in which post-usage actions are enforced. In each of these three phases, attribute values and the authorization context are continuously checked in order to promptly react to changes. This enables continuous control in the *pre* and *post* phases in UCON+, which was not enacted in UCON.

`Rules` are functional predicates over attributes that must be evaluated to determine whether an authorization can be granted based on the values of attributes. Rules are classified by phases such that each class is evaluated only when the authorization enters the corresponding phase. For instance, *ongoing* rules must be evaluated only when the authorization has been granted and is in progress. Rules must be re-evaluated whenever an attribute value changes and the authorization decision must be updated accordingly.

`Decisions` are the evaluation outcome of rules. UCON+ extends the classical two-valued decisions (i.e., *Permit*, *Deny*) with two additional values as follows: *Indeterminate* indicates a decision cannot made due to an error or a missing attribute; and *NotApplicable* denotes that the system could not find any rule that matches the authorization request. The decision of an authorization may alter between these values as the context changes throughout the different phases of the authorization as mentioned above.

`Instructions` are mandatory or optional actions defined in security policies and enforced upon evaluation. Instructions include subject obligations, attribute updates and system actions. For instance, a security policy may specify that the metadata of a file must be updated (attribute update). Another example is a security policy specifying that the subject must complete Multi-Factor Authentication (MFA) (subject obligation), and an email must be sent to the sysadmin when the authorization is granted (system action).

1.2.2 Authorization Session

In traditional access control (e.g., RBAC, ABAC), authorizations are momentary such that only one authorization decision is made per request. UCON introduced the concept of continuous authorization that spans over a period of time, during which the security policy is re-evaluated and the authorization decision is updated accordingly. We designate the context of a single continuous authorization as the *Authorization Session*. An authorization session refers to all events, policy evaluations and contextual information that belong to the lifetime of a single continuous authorization. An authorization session in UCON+ is initiated by and associated with a single *authorization request*. As aforementioned, UCON+ enacts continuous control after the revocation or end of authorization. Thus, an authorization session may last beyond the revocation of authorization to ensure that post-usage actions are completed. Therefore, a UCON+ authorization session starts with a request and ends upon the enforcement of all *post* rules. Attributes that are relevant to a particular authorization session (i.e., required for policy evaluation) are continuously monitored throughout the session, and a re-evaluation of the security policy is triggered whenever an attribute value changes.

1.2.3 Policy Language

eXtensible Access Control Markup Language (XACML) is the OASIS standard policy language to express ABAC policies [9]. XACML cannot be used to express UCON policies as it does not include phases, which are a main component of UCON and UCON+ as mentioned in Section 1.2.1. To enable the novelties of UCON, Colombo et al. introduced U-XACML [3, 15], an extension of XACML that incorporates UCON concepts. However, U-XACML is incompatible with XACML as it includes some modifications of the standard. For example, U-XACML rules consist of multiple conditions to be evaluated during *pre*, *ongoing* and *post* phases. This change is not compatible with the XACML, which allows only one condition per rule. Thus, U-XACML requires specific modifications to evaluators making its adoption harder. Moreover, XACML and U-XACML are verbose and complex languages, which undermines their readability and efficiency.

To overcome the above drawbacks, UCON+ uses Abbreviated Language For Authorization (ALFA) [8], a pseudocode domain-specific policy language that is in the standardization process by OASIS. ALFA maps directly to XACML without adding any new syntax or semantics, and is used to express ABAC policies. Instead of modifying the ALFA standard, we express UCON+ phases as attributes specified during a usage request, unlike U-XACML which defines phases as conditions within a single rule. ALFA is much less verbose than XACML, which makes it more human readable and shorter in size allowing faster parsing and evaluation. ALFA adheres to the same hierarchy as XACML where decision predicates are expressed in rules that are nested under policies, which in turn are nested under policy sets and/or namespaces. Policies and rules resolve to one of the decisions described in Section 1.2.1 (i.e., *Permit*, *Deny*, *Not Application*, *Indeterminate*) based on their evaluation by the ALFA policy engine. Like XACML, ALFA relies on combining algorithms to resolve conflicts between sibling rules or policies. ALFA also allows the use of functions, such as regular expression, string concatenation and others, which helps to further refine applicability of policies and rules. Listing 1.1 gives an example ALFA policy that limits the action of opening the front door of the house to its owner. The policy adheres to a non-inclusive hierarchy expressed in ALFA syntax as a nested structure of graph parentheses. Thus, each ALFA element is enclosed between parentheses and nested under the parent element.

The "target" clause at line 2 determines the applicability of the policy "door" based on the values of the `resource` and `action` attributes, which must be `FRONT_DOOR` and `OPEN` respectively. Line 4 specifies the combining algorithm *firstApplicable* which takes the result of the first applicable rule, in this case the result of *permitIfOwner* defined at line 5. There is only one rule *permitIfOwner* and it is applicable if its *target* matches the value `employee`. If applicable, the decision of this rule evaluates to *permit* and includes an optional instructions (i.e., advice) as expressed in lines 9-14. The instruction assigns the value `open`, `door` and `email` to attributes `command`, `resource` and `channel` respectively. UCON+ also supports administration and delegation ALFA policies, which allow resource owners and administrators to determine who is allowed to issue policies about their resources. This is further described in the following section.

Listing 1.1 An example ALFA policy

```
policy door {
    target clause Attributes.resource == FRONT_DOOR &&
                    Attributes.action == OPEN
    apply firstApplicable
    rule permitIfOwner {
        target clause Attributes.role == "employee"
        permit
        on permit {
            advice notify {
                command = open
                resource = door
                channel = email
            }
        }
    }
}
```

1.2.4 Administration and Delegation

Policy administration controls the types of policies that individuals can create and modify for specific groups of resources. Different classifications such as action type, topic, or some other property are also possible under which policy issuance, revocation or any change can take place. Further operations like delegation are also possible within policy administration, where delegation of authority, or the delegation of the right to issue/revoke policies are possible. For instance, system administrators or resource owners may want to restrict and control who can write policies about their resources. As another example, a resource owner may want to delegate their authority to someone else based on some conditional statements, which could be absence from office or conditions based on time, location etc. The ADP [11] supports such use cases and allows administrators to write policies about other policies forming trees that start with top level policies designated as root of trust.

The truth table mentioned in Table 1.1 illustrates the outcomes given the *Admissible* and *NotAdmissible* evaluation effects combined with the Usage policy evaluation effects. Indeed, in some cases, multiple policies may be Admissible and have a Permit/Deny result. For this reason, a policy reduction algorithm is used, which we discuss in detail in Section 1.4.1. Editing a full-fledged policy with ADP incorporates the steps as mentioned in Listing 1.2.

In our previous work [1], we introduced the “`PolicyIssuer`” keyword to ALFA, which identifies the author of the policy, and is used to create an

Table 1.1 Evaluation Outcomes

Usage Policy	**Administrative Policy**	**Result**
Permit	Admissible	Permit
Deny	Admissible	Deny
Permit	NotAdmissible	Find other policy
Deny	NotAdmissible	Find other policy

Listing 1.2 Evaluation and Reduction process

```
<given a root policy defined by authority / organization>
user writes custom policy Pa
synthetic request is crafted from Pa
request is evaluated against Pa
    select administrative policy Padmin for Pa
        for each Pi in Padmin
            create administrative request Ra using R and Pi
            evaluate Ra against Pi
            if deny
                <show modifications>
            else
                <allow policy>
```

administrative request. This enables the use of ALFA to express and issue administrative policies and to support ADP in UCON+.

The policy evaluation flow can be outlined in two steps: reduction and combination. *Reduction* determines whether a usage policy was written by an authorized personnel and whether the evaluation outcome can be considered or not. This is achieved by creating an evaluation tree that has a root-of-trust policy as its root node and the results of applicable policies as its edges, and then finding the branches that can reach the root-of-trust. In the *combination* step, the Policy Decision Point (PDP) combines all valid results using combining algorithms as specified in the policies. The ADP defines a maximum delegation depth to limit the number of administrative policies to be evaluated. Accordingly, the path is discarded if the number of nodes in a path in the reduction graph exceeds the maximum delegation depth.

For both administrative and usage policies, we use the *denyUnlessPermit* combining algorithm due to its deterministic and restrictive nature. This algorithm is restrictive because its default result is always *deny*, unless there is an explicitly applicable permit rule. This also eliminates any indeterministic results like *indeterminate* when an attribute value is missing or a condition is false. Therefore, when an administrative policy evaluates into *indeterminate*, the corresponding branch is discarded from the entire delegation tree.

1.3 Architecture

This section presents UCS+, an architecture that realizes the UCON+ model presented in the previous section. UCS+ is mainly based on the architecture put forward by the XACML[9] standard and by the UCON framework introduced by Lazouski et. al [6] with several enhancements that we added. We start this section by eliciting the fundamental requirements to be filled by this architecture, and then we describe the components that build up this architecture. Lastly, we illustrate how these components interact to achieve the different required workflows.

1.3.1 Requirements and Components Classification

The main objective of UCS+ is to enable controlling the usage of resources in a uniform way across the broad spectrum of the digital medium, e.g., cloud, edge, terminal, or IoT. The control functionality must accommodate to the mutability of contexts by providing functions to sense the environmental changes (e.g., by monitoring mutable attribute values). In turn, UCS+ must enable methods to grant and revoke decisions of access or usage. To achieve this ambitious goal in a modular, and extensible way, UCS+ comprises three categories of components that are: *fixed*, *programmable*, and *configurable*. Firstly, fixed components are responsible for a specific task regardless of the environment or the application. Secondly, programmable components provide basic behavior and the skeleton of specific functionality. This behavior can be extended to realize any domain-specific requirements. Thirdly, configurable components have defined functionalities that do not change; however, they need to be adapted to cope with specific system properties.

Regardless of their class, all the components communicate using a publish-subscribe model to manage the potential of increasing interactions efficiently. As such, each component registers to events and messages relevant to it. This enables a distributed infrastructure where a component can be spawned in several instances. Thus, UCS+ contains a Message Bus (MSG BUS) to handle our communication paradigm. In the following, we decompose the classes mentioned above into concrete components as illustrated by the 1.2.

1.3.2 Components Description

Context Handler (CH) is a *fixed* core component that is responsible for receiving and routing access requests and managing the authorization workflow of such requests. The mentioned MSG BUS is a sub-component that enables CH to achieve this management of the workflow.

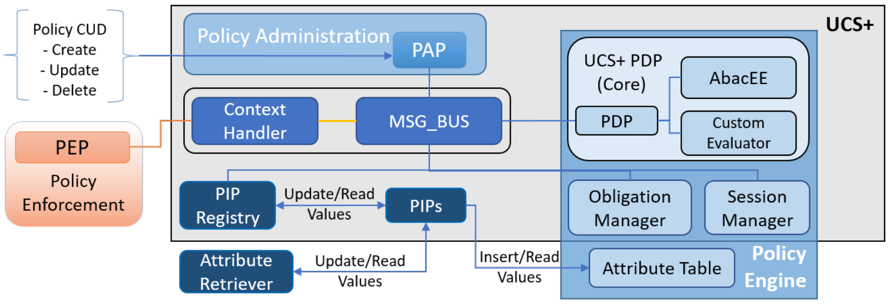

Fig. 1.2 The Architecture of UCS+ illustrating the different component.

Policy Administration Point (PAP) is a *configurable* component that acts as a data store of policies, policy sets and namespaces in the system. It offers Application Programming Interface (API) to retrieve, add or update them. This component can be tuned according to different scenarios to persist policies in, e.g., SQL, NO-SQL, or in-memory databases.

Session Manager (SM) is a *configurable* component that is responsible for the continuity of control. Essentially, it is a data structure that stores information about all active sessions. Depending on their authorization workflow, sessions transition among different phases, i.e., *pre*, *ongoing*, and *post*. As such, for a specific session, the first entry in SM is the event following the (Permit) response to the request. Semantically, this means that access has been granted, but the actual usage of the resource still did not start. Later, when the usage of the resource begins, SM changes the status for this session, and UCS+ starts tracking any changes of the mutable attributes. Furthermore, SM keeps a record of the relevant policy for a session, the analyzed request, and a pointer to the relevant attributes. In cases of access revocation, SM updates the session status to Revoked. The session is kept in such a state until all possible obligations are handled, and the access is stopped. Afterwards, the session record is removed from the SM.

Policy Information Point (PIP) is a *programmable* component that defines where to retrieve the values of specific attributes and how to monitor them. PIP is also used to trigger a change in the value of an attribute. A PIP implementation depends on the specific attribute retriever it interacts with; hence, a PIP will be implemented based on the application-specific requirements. To support PIP, we added an additional component, PIP Registry, to manage PIPs and track which PIPs are responsible for which attributes. Attribute Retriever (AR) is another component that supports this process by enabling querying and updating attribute values.

PDP is a *fixed* component that is responsible for evaluating a request and based on a policy. The result of this evaluation can either be a *PERMIT* (access is authorized), *DENY* (access is denied), *NOT APPLICABLE* (decision cannot be taken due to semantic reasons, e.g., no rules in the policy are matched), or *INDETERMINATE* (decision cannot be taken because either the policy or the request is malformed). To form a decision, the PDP parses the request, resolves the relevant attributes, and then matches them with the values in the policies conditions. To perform the evaluation, PDP may leverage different expression evaluators. The core evaluator in UCS+ is AbacEE to resolver ABAC expressions. UCS+ can also be extended with custom evaluators for particular expressions, such as the one contributed in [4] for Trust Level Evaluation Engine (TLEE) expressions. To handle conflicts, i.e., a single request having several applicable rules, combining algorithms are used to determine the priorities among rules and resolve conflicts.

Obligation Manager (OM) is a *programmable* component that handles a crucial part of the UCON+ model, the obligations. Obligations indicate additional actions that must be performed together with access decision enforcement. Technically, obligations are abstract actions that are not bound to a specific format. For example, obligations can be used to control the values of attributes; in such cases, they are called attribute-update obligations.

Attribute Table (AT) is an auxiliary *configurable* component that we added to manage attributes. It is especially needed in environments with faulty attribute retrievers where there is a possibility that values are not promptly available. While PIP collects the values of the attributes, the AT is responsible for caching these values to handle the periodic polling of values as part of the continuous attribute monitoring. In case the attribute retriever does not include any subscription mechanisms, which would allow the PIP to be notified when an attribute value changes, the viable strategy is to query the attribute retriever periodically. In such a case, AT registers the polling time needed for each attribute according to its criticality.

Lastly, the Policy Enforcement Point (PEP) is the component that integrates the UCS+ with the target application. It is responsible for intercepting usage and access requests within the application and directing them to the UCS+ by interacting with CH. Then, it handles the results of these interactions, e.g., allowing the access, or performing an obligation.

To wrap up this section, we present a sequence diagram of the core workflow of UCS+. Figure 1.3 illustrates a simplified flow of how UCS+ evaluates.

1.4 Implementation

UCS+ implementation has been split into three main software components making it extensible, customizable and portable. This structure has been designed to have self-contained codebases and to maximize portability and performance in target environments including IoT, mobile and cloud. The three components are libalfa, ucon-c and ucon-service as described below.

`libalfa` is a library that incorporates the features of the ALFA policy language and encompasses PDP functionalities (e.g., policy parsing and eval-

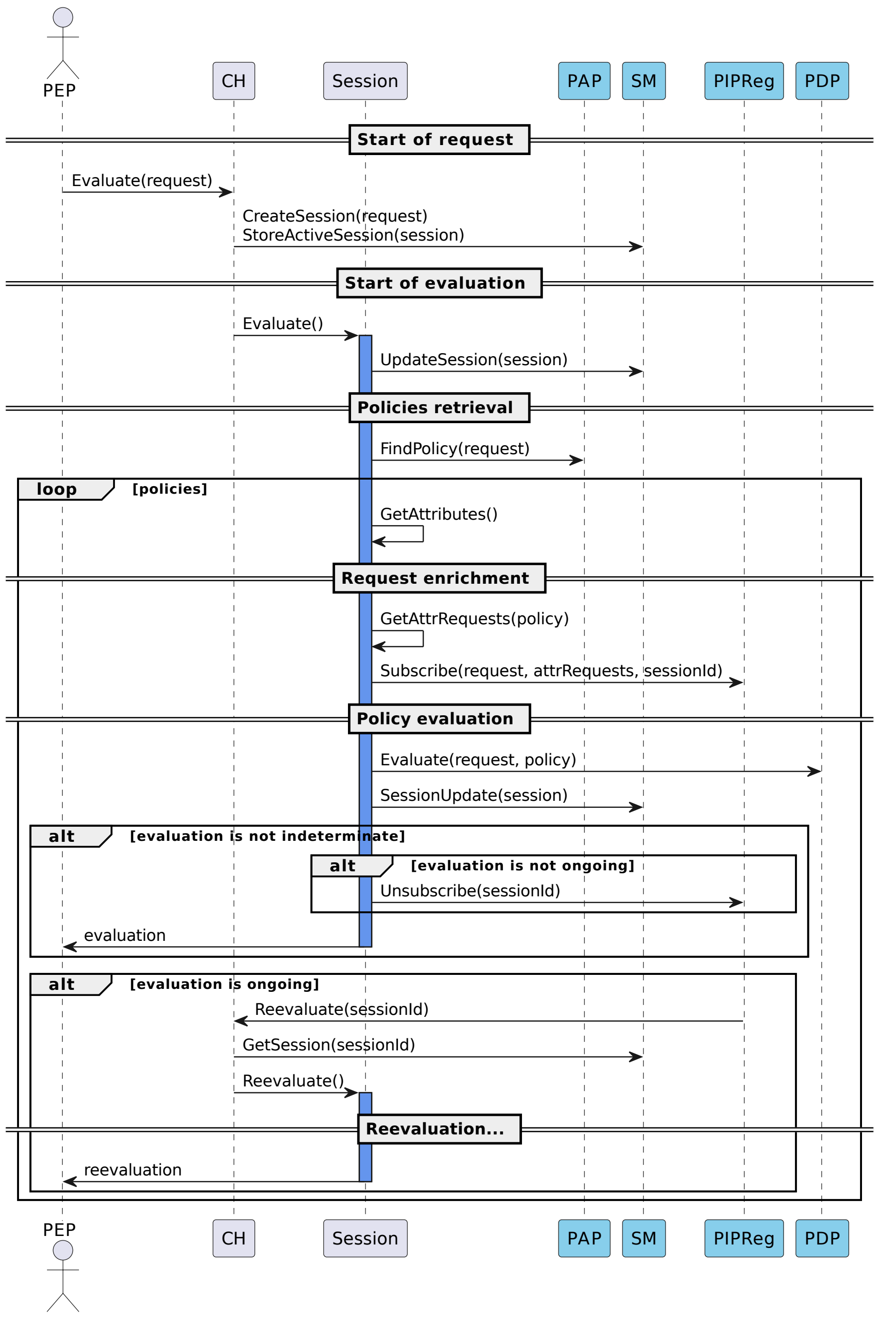

Fig. 1.3 A sequence diagram of the core UCS+ workflow.

uation) and part of the PAP functionalities. It has been entirely developed in C++ with no external dependencies, making it deployable on many different devices and with high performance. libalfa is a self-contained module for ABAC, which enables its use for scenarios where simple access control is needed without session support or external attributes. libalfa also supports the following two XACML profiles:

1. Multiple Decision Profile (MDP): used in use-cases where access to multiple resources can be requested within a single request.
2. ADP: used to define a root of trust for policies creating a chain of authority to track responsibility and limit attack surface.

`ucon-c` is a library that implements UCON+ novelties such as session-based monitoring and policy re-evaluation according to the architecture described in Section 1.3. It has also been entirely developed in C++ in order to allow an easy integration with libalfa, maximize performance, and enable its deployment on different and low-end devices. ucon-c structure mirrors the block diagram in Figure 1.2. Programmable components have been implemented as abstract classes to guide and facilitate their implementation for the requirements of domain-specific use-cases. For instance, PIPs are context-dependent and their behavior varies with the ARs they are connected to, but some of their functionalities are common and do not need to be re-implemented such as caching latest attribute values or enriching requests. ucon-c is *language agnostic* making it adaptable to different policy-languages. Therefore, libalfa can be substituted with other libraries that implement other policy languages such as Open Policy Language (OPA) working with Rego or XACML working with Balana/AuthzForce.

`ucon-service` is a service that exposes an API that enables the use of UCS+ in cloud environments. ucon-service has been developed using Golang due to scalability, performance and resilience requirements. ucon-service exposes a gRPC API that allows managing PIPs and policies as well as initiating usage sessions by external PEPs. The service specifically exposes the following gRPC endpoints:

- `Usage` endpoint invoked to start a usage session. It returns a sequence of responses corresponding to the three phases of UCON+ sessions. The endpoint may also respond with an updated usage decision such as revoke, whenever a change occurs and policies are re-evaluated.
- `PAP` endpoint used to create, delete and retrieve policies.
- `PIP` endpoint used to subscribe, unsubscribe and update attributes.

1.4.1 ADP Reduction Algorithm

According to the ADP, a Usage Policy evaluation result will not be enforced unless evaluation of Administrative Policies result into a permit. Thus, a Usage Policy can be *Admissible* if its enforcement is authorized by all upper layer policies, or *NotAdmissible* if one of the Administrative Policies in the hierarchy cannot be considered. This is expressed in the evaluation outcomes truth table illustrated in Table 1.1, which shows that whenever Administrative Policies are admissible, a final result is obtained. On the other hand, if one of the Administrative Policies is NotAdmissible, then another policy needs to be found. Indeed, in some cases, multiple policies may be Admissible and have a Permit/Deny result.

A policy reduction algorithm is used to manage inconsistencies and to guide editing of administrative policies in order to avoid a situation where all evaluations end up in a NotAdmissible state. To achieve this, once an administrative policy is added, compliance with the above layers can be checked and suggestions for improvements may be provided. Policy reduction is a process by which authority of a policy associated with an issuer is verified. A reduction tree is built on the basis of the issuer ID of each policy in a policyset. When an access/usage policy is evaluated, the reduction graph is searched for a trusted administrative policy, based on which the combined evaluation effect will be taken for consideration by the PDP. Listing 1.3 describes the reduction algorithm given a policy set and a request. The policies are combined as usual with the defined combining algorithms.

Listing 1.3 Evaluation and Reduction process

```
<given a policy set PS and a request R>
evaluate R against PS
find applicable policies Papp
for each applicable policy Pa in Papp
    if deny
        continue
    if PolicyIssuer is absent
        then combine
    else
        [selection step] : select administrative policy Padmin for Pa
            for each Pi in Padmin
                create administrative request Ra using R and Pi
                evaluate Ra against Pi
                if deny
                    no edge
                else
                    add edge
                    if !policyIssuer
                        potential path is found
                    else
                        go to selection step with Pi as Pa

combine edge results with PS combining algorithm
```

1.4.2 Performance

UCS+ has been designed to work on resource-limited devices such as IoT as well as cloud and high-performance environments. We evaluated the performance of UCS+ in the following different circumstances:

- *Cold start*: No attributes are installed in the system. We measured both the end to end time to obtain an evaluation and re-evaluation time, considering the communication penalty and assuming all missing attributes come in the same time.
- *Standard run*: All attributes are already installed.
- *Re-evaluations*: Attributes are supplied one by one at different times in order to evaluate how much time the system takes to perform re-evaluations if attributes are not all immediately supplied.

We also evaluated the performance of the *PDP* to show the time it takes to perform a policy parsing and evaluation compared to Balana[1] XACML evaluator as shown in Figure 1.4 and Table 1.2. We only measured the performance of ABAC part of UCS+ PDP in order to be coherent in our performance evaluation as Balana does not support UCON. The results show that our C++ implementation is more than 40 times faster than Balana.

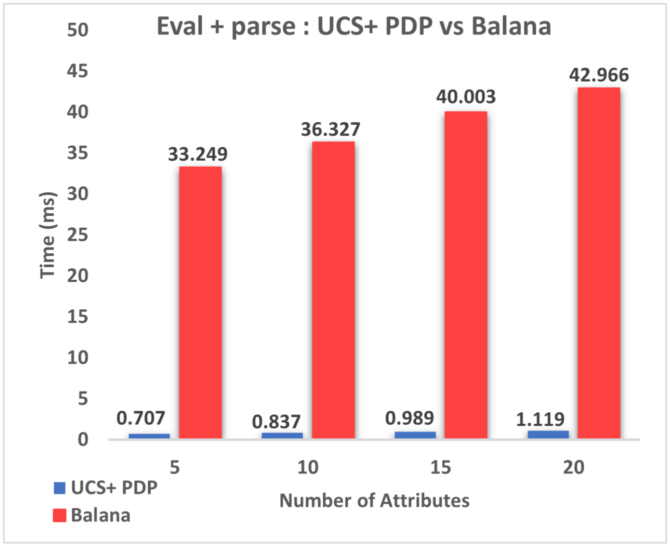

Fig. 1.4 Performance of UCS+ ALFA Evaluator vs Balana XACML Evaluator

[1] https://github.com/wso2/balana

Table 1.2 Policy length comparison (in KB)

Number of attributes	ALFA	XACML
5	1.1	8.2
10	1.8	16
15	2.6	24
20	3.4	32

1.5 Use-cases

This section describes to example use-cases for leveraging UCON+ in smart home and smart vehicle environments.

1.5.1 Smart Home

One of the possible use-cases is a smart home environment where it is important to provide safety and protection to people and things since these are connected to the Internet and access to the outside world need to be controlled. For example, multimedia devices used by the kids at home, such as game consoles and smart TV, connect to the Internet and expose kids to a variety of contents that are extremely difficult to manage and control. UCON+ could make significant improvements if combined with the smart home devices and sensors compared to the existing parental controller model where parents have to create a child account for each multimedia device and set limited factory-defined policies. Thus, UCON+ security policies can be specified for safety purposes. For example, if the camera detects that the parents are sleeping or away from home, the child is not allowed to use the smart TV or access the Internet. Moreover, UCON+ obligations enable parents to define specific actions that must be completed before allowing their kids to use such smart devices. For instance, parents can specify an obligation that notifies them when a child attempts to use a smart device, and wait for their approval before allowing the child to use the device. UCON+ supports such cases by monitoring the environment through a variety of devices such as cameras and sensors and using their input as attributes for policy evaluation, allowing dynamic control in the smart home environment.

1.5.2 Smart Vehicle

Vehicles are real-time applications that are accessed by different entities (e.g., drivers, passengers, applications) in different environments. Thus, they require a dynamic authorization mechanism that can provide continuous usage control to their resources as the context changes. UCON+ can be leveraged to enable such functionality in smart vehicles where access policies are re-evaluated when the location of the car changes as it moves. For instance, the minimum age to drive is 17 in Denmark and 18 in Sweden. Thus, UCON+ policies can specify that a 17-year old driver must be notified when they cross the border from Denmark to Sweden. UCON+ policies may also incorporate obligations that can gracefully stop the vehicle. Car rental is another example where car owners can restrict the use of their vehicles based on rental agreements. A car owner, for example, can designate a specific area in which the driver is permitted to drive. The car owner can also specify obligations that notify them if the driver breaks the conditions of the rental agreement. The owner can also specify a maximum speed or a specific driving period.

1.6 Conclusions and Future Work

The UCON model was introduced as a generalization of access control to support attribute mutability and continuity of control. Specifically, UCON continuously monitors attributes, then re-evaluates the security policy when an attribute value changes and revokes the authorization if the policy is no longer satisfied. UCON, however, does not enact continuous monitoring before granting 0r after revoking the authorization, which is necessary for proactive and safety-critical systems.

We proposed UCON+ to solve the limitations of UCON, enhance it with policy administration and delegation as well as auxiliary evaluators, and extend it with continuous monitoring that covers pre- and post-authorization interactions. UCON+ uses ALFA as a baseline policy language and solves the problems of U-XACML as described in Section 1.2.3. We presented a reference architecture for UCON+, designated as UCS+, which is adapted from the architecture of UCON. We also described an optimized and lightweight implementation of UCS+, and leveraged it in previous works in different domains such as IoT [4], smart vehicles [5] and data protection [1]. Our implementation incorporates the world's first ALFA evaluator, which also supports policy administration and delegation. We fuse our previous works on UCON+ with our recent progress on the subject and present them in this chapter.

Our future directions aim for formalizing the model and applying formal verification methods to it. We also plan to further develop UCON and UCS+ by capturing a more granular temporal state and adding more phases to the authorization session. Additionally, we intend to validate UCON+ in more domains such as cloud services and identity management systems and evaluate it in both centralized and distributed environments.

Acknowledgments

The authors would like to acknowledge the contributions of Yair Diaz and Athanasios Rizos who participated in the research and development of UCON+ at Huawei's Munich research center.

References

1. Bandopadhyay, S., Dimitrakos, T., Diaz, Y., Hariri, A., Dilshener, T., Marra, A.L., Rosetti, A.: DataPAL: Data Protection and Authorization Lifecycle framework. In: 2021 6th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM). IEEE (2021)
2. Chung, Ferraiolo, D., Kuhn, D., Schnitzer, A., Sandlin, K., Miller, R., Scarfone, K.: Guide to Attribute Based Access Control (ABAC) Definition and Considerations (2019). URL https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=927500
3. Colombo, M., Lazouski, A., Martinelli, F., Mori, P.: A Proposal on Enhancing XACML with Continuous Usage Control Features. In: F. Desprez, V. Getov, T. Priol, R. Yahyapour (eds.) Grids, P2P and Services Computing [Proceedings of the CoreGRID ERCIM Working Group Workshop on Grids, P2P and Service Computing, 24 August 2009, Delft, The Netherlands], pp. 133–146. Springer (2009). DOI 10.1007/978-1-4419-6794-7_11. URL https://doi.org/10.1007/978-1-4419-6794-7_11
4. Dimitrakos, T., Dilshener, T., Kravtsov, A., La Marra, A., Martinelli, F., Rizos, A., Rosetti, A., Saracino, A.: Trust Aware Continuous Authorization for Zero Trust in Consumer Internet of Things. In: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 1801–1812 (2020). DOI 10.1109/TrustCom50675.2020.00247
5. Hariri, A., Bandopadhyay, S., Rizos, A., Dimitrakos, T., Crispo, B., Rajarajan, M.: Siuv: A smart car identity management and usage control system based on verifiable credentials. In: IFIP International Conference on ICT Systems Security and Privacy Protection, pp. 36–50. Springer (2021)
6. Lazouski, A., Martinelli, F., Mori, P.: A Prototype for Enforcing Usage Control Policies based on XACML. In: International Conference on Trust, Privacy and Security in Digital Business, pp. 79–92. Springer (2012)
7. Martinelli, F., Matteucci, I., Mori, P., Saracino, A.: Enforcement of u-xacml history-based usage control policy. In: International Workshop on Security and Trust Management, pp. 64–81. Springer (2016)
8. OASIS: Abbreviated language for authorization Version 1.0 (2015). URL https://bit.ly/2UP6Jza
9. OASIS: eXtensible Access Control Markup Language (XACML) Version 3.0 Plus Errata 01 (2017). URL http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-en.html
10. Park, J., Sandhu, R.: The UCONABC usage control model. ACM Transactions on Information and System Security (TISSEC) **7**(1), 128–174 (2004)

11. Rissanen, E., Lockhart, H., Moses, T.: XACML v3.0 Administration and Delegation Profile Version 1.0. Committee Draft **4** (2014). URL https://docs.oasis-open.org/xacml/3.0/xacml-3.0-administration-v1-spec-en.html
12. Sandhu, R., Munawer, Q.: How to do discretionary access control using roles. In: Proceedings of the third ACM workshop on Role-based access control, pp. 47–54 (1998)
13. Sandhu, R.S.: Lattice-based access control models. Computer **26**(11), 9–19 (1993)
14. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. Computer **29**(2), 38–47 (1996)
15. U-XACML: XACML with Usage Control (UCON) novelties (2015). URL https://bit.ly/3FmeqE6