Formal Verification of Usage Control Models: A Case Study of UseCON Using TLA+

Antonios Gouglidis
School of Computing and Communications
Lancaster University
Lancaster, United Kingdom
a.gouglidis@lancaster.ac.uk

Christos Grompanopoulos
Department of Mechanical Engineering
University of Western Macedonia
Kozani, Greece
cgrompanopoulos@uowm.gr

Anastasia Mavridou
Institute for Software Integrated Systems
Vanderbilt University
Nashville, TN, USA
anastasia.mavridou@vanderbilt.edu

Usage control models provide an integration of access control, digital rights, and trust management. To achieve this integration, usage control models support additional concepts such as attribute mutability and continuity of decision. However, these concepts may introduce an additional level of complexity to the underlying model, rendering its definition a cumbersome and prone to errors process. Applying a formal verification technique allows for a rigorous analysis of the interactions amongst the components, and thus for formal guarantees in respect of the correctness of a model. In this paper, we elaborate on a case study, where we express the high-level functional model of the UseCON usage control model in the TLA+ formal specification language, and verify its correctness for ≤ 12 uses in both of its supporting authorisation models.

1 Introduction

Access control systems offer the mechanisms to control and limit the actions or operations that are performed by a user or process – referred to as *subjects* – on a set of system *objects*. Specifically, an authorisation process is required to take a decision for granting or denying a *subject* to access an *object* of the system, based on a set of existing policy rules. Thus, access control systems are considered to be amongst the most critical of security components. Their importance is recently highlighted in NIST's special publication [4], where verification approaches for various access control models are examined.

New computing paradigms, e.g., the cloud [2], require further investigation of access control systems. This eventually introduces the need for models, such as usage control [3], to cope with complex high-level requirements set by new computing paradigms. Usage control models provide an integration of access control, digital rights, and trust management, which may be applicable in environments such as the cloud. To achieve this integration, usage control models support additional concepts such as attribute mutability and continuity of decision. Attribute mutability is responsible for updating the values (e.g., attribute values) of the participating entities (e.g., *subjects*, *objects*), which results in having a dynamic – usage – control model. Continuity of decision considers that access to an *object* is no longer an instantaneous access or action, but it may last for some time. Consequently, decision factors are evaluated not only before (i.e., pre-authorisation), but also during the exercise of an access (i.e., ongoing-authorisation), and thus evolving the concept of access control to that of usage control.

Submitted to:
MeTRiD 2018

Temporal logic is applied to provide unambiguous semantics to functions supported by usage models, such as attribute mutability and continuity of decision [6]. Usage control systems are concurrent and characterised by non-determinism due to the potential of a *subject* to arbitrarily request or terminate using an *object* during the operation of a system. As discussed in [10], a number of usage control formal models are limited to the specification of a single use, which is isolated and has no interference with other uses. UseCON [3] is a usage model that provides enhanced expressiveness compared to existing access/usage control models and applicable in new computing paradigms [3]. This is achieved by introducing a new entity entitled *use*, which enables the use of historical information in usage control decision. To ensure the correctness of the UseCON model, we used an existing formal language to verify the correctness of its supported *use* management procedures. For example, ensure out of bound values are not assigned to attribute values, and that authorisation states of the usage model always comply with certain behaviours and follow predefined authorisation transitions (e.g., an authorisation must always be requested before changing into another state). The enhanced version of Temporal Logic of Actions (TLA+) [5] is selected for the formal specification of UseCON – TLA+ has been selected also in formal definitions of other usage models (e.g., UCON [11]). In this paper, we anticipate that the formal specification and verification of UseCON using an automated and error-free model checking technique will provide a comprehensive and unambiguous understanding of the concepts introduced in UseCON and ensure its correctness for a number of uses, depending on the authorisation model (pre-authorisation and ongoing-authorisation).

The remainder of this paper is organised as follows: Section 2 provides a specification of UseCON's high-level requirements in TLA+, and the verification of its system model is described in Section 3. A performance evaluation of the TLC model checker to verify deadlock, safety and liveness properties is given in Section 4. Concluding remarks and future work are provided in Section 5. Although not a prerequisite, we provide a brief overview of UseCON and TLA+ specifications in Appendix A and B, respectively.

2 Specification of UseCON in TLA+

UseCON is a usage control model that is characterised by extended expressiveness when compared with existing usage-based models (e.g., UCON [11]). Specifically, it may support authorisations that have logical relations between entities (see *direct* and *indirect* entities in Appendix A and [3]), as well as historical information, and can express complicated requests between *subjects* and *objects* through simple policies (see automated management of use entities in [3]). In the following, we provide a formal specification of the UseCON model in TLA+ (see Appendix B). This includes the specification of UseCON's main elements, decision making rules, and procedures for managing *use* elements. Furthermore, we describe two transition systems (TS) of UseCON, i.e., one TS for the *pre* and one TS for the *ongoing* authorisation model, which are used for supporting concurrent operations amongst uses in UseCON.

2.1 Main Elements

A *use* in UseCON represents a request to execute an *action (a)* from a *subject (s)* on an *object (o)*. All *subjects*, *objects* and *actions* used in the usage control system define the sets of *subjects (S)*, *objects (O)*, and *actions (A)*, respectively. The set of *subjects S*, *objects O*, and *actions A* are defined, as follows:

$$S \triangleq \{s_1, \dots, s_i\}, O \triangleq \{o_1, \dots, o_j\}, A \triangleq \{a_1, \dots, a_k\}$$

The set of entities E is defined as the union of S, O and A sets, as follows:

$$E \triangleq S \cup O \cup A$$

The characteristics of an entity are represented through its attribute values. An attribute is a function whose domain is a particular set (S or O or A) and its range is composed of specific attribute values, as follows:

$$att_i : X \mapsto RangeAtt_i$$

where $X = S$, or $X = O$, or $X = A$. For every system's entity, an identification attribute *id* is defined for assigning a unique value to the entity. The *id* value remains the same throughout the life-time of the usage control system. Thus, the following invariant is valid for all the behaviours of the usage control system:

$$\forall e_1, e_2 \in E : \; id[e_1] = id[e_2] \Rightarrow e_1 = e_2$$

In UseCON, the operation of the usage control system does not modify automatically the attribute values of system entities. Thus, attribute values of system entities are proposed to be updated manually by the usage control administration model. Consequently, a system entity is represented with a constant record having as fields the entity's attribute values, as follows:

$$e \triangleq [id(e) = k, att_1(e) = l_1, att_2(e) = l_2, \ldots, att_n(e) = l_n]$$

where $e \in E$ is a system entity and $att_i(e)$, with $i \in 1, \ldots, n$, represents the value l_i of one of its attributes. The first record field of each entity is an *id* attribute.

A *use* request in UseCON results into the creation of a *use*. A *use* is an entity instantiated by UseCON and describes all the usage requests in a system and it is described with attributes. More specifically, every *use* must contain a *use* id attribute having a tuple composed of the attribute values *sid*, *oid*, *aid* that describe the identities of the *subject*, *object*, and *action* participating in the instantiated *use*. A special attribute *st* is associated to every *use* instance representing the status of the *use*. Its value may be one of the following: *'requested'*, *'activated'*, *'denied'*, *'stopped'*, and *'completed'*.

A *use* u that instantiates a specific *use* request from subject s to object o for action a is represented in the specification of the model with a variable record, having as fields its attribute values as follows:

$$u \triangleq [sid(u) = s.id, oid(u) = o.id, aid(u) = a.id, st(u) = state, att_1(u) = v_1, att_2(u) = v_2, \ldots, att_n(u) = v_n]$$

where s.id, o.id and a.id are the identity values of the subject, the object, and the action, respectively. The state attribute *st* gets a value *state* which belongs to the following set: $state \in \{$ 'requested', 'activated', 'denied', 'completed', 'stopped' $\}$ and $att_i(u), i = 1, 2, \ldots, n$ are *use* attribute values. The set of all system *uses* is U. During the operation of the usage control system, U is populated due to the *use* requests. Moreover, as these *use* requests are served by the usage control system, the *use* attribute values are modified. Specifically, the U is altered when a new *use* is requested or change its progress status (e.g., from *'activated'* to *'stopped'*). Consequently, the only variable utilised in the specification is U, which is the set containing all the *uses* operated in the UseCON model and it is declared as follows:

$$VARIABLES\;\; U$$

In the beginning of a TLA+ specification, several modules may be included for supporting different operators. Our specification includes *Integers* and *FiniteSets* modules, which encompass arithmetic and set-related operators, like *Cardinality*. This is declared as follows:

$$EXTENDS\ \ Integers, FiniteSets$$

2.2 Decision Making in UseCON

Policy rules in UseCON provide an enhanced utilisation of information from entity and *use* attribute values. More precisely, the general form of a UseCON policy rule that governs the allowance of a *use* request from a *subject s* on an *object o* with an *action a*, is a boolean valued expression defined as follows:

$$Policy\_Rule(s,o,a,S,O,A) \triangleq expression(e_1,\dots,e_n)$$

where s,o,a are the particular *direct* entities of the *use*. In addition, two or more UseCON policy rules can be combined together with logical operators as follows:

$$p = p_1 \otimes p_2 \otimes \dots \otimes p_n$$

where $\otimes$ is a logical operator (e.g., AND, OR), and p_i is a policy rule, where $i = 1,\dots,n$.

The parameters $e_i\colon\ i \in 1,\dots,n$ of a policy rule that are utilised for the evaluation of the *expression* may have various origins, and thus lead to the creation of the following categories of policy rules:

- *Direct Policy Rules*: The parameters e_i in the expression of a direct policy rule are values only from attributes of *direct* entities or constant values. Specifically, all parameters e_i, are defined by the following formula:

  $$e_i \in \{s,o,a\}\ or\ e_i \triangleq l$$

  where l is a constant value.

- *Indirect Policy Rules*: The expression of an *indirect* policy rule consists of attribute values stemming not only from *direct*, but also from *indirect* entities. In this case a logical relation exists between the two types of entities, which can be retrieved by a *select* expression. The definition of such an expression is as follows:

  $$e_i \triangleq CHOOSE\, x \in E\colon select(x,s,o,a,l)$$

  An example of an indirect policy rule could be an expression that evaluates into true or false, if the father of a child has a premium membership. In this example, the child is a *direct* entity, while the father is an *indirect*. The *select* expression should take into account the fact that there is a *'father'* attribute in the child entity having the attribute value of her father's identity.

- *Complex Indirect Policy Rules* New computing paradigms introduce complex access control policies, where the usage decision is based on information related not only to a single entity, but with a subset of entities. Such complex policies can be supported in UseCON through complex indirect policy rules. More specifically, a parameter e_i of a complex indirect policy rule can be, apart from a single (i.e., direct or indirect) attribute value, an aggregation of information. This information is derived from all the entities that satisfy a desired (*select*) predicate. The semantics of a parameter e_i of a complex indirect policy rule is as follows:

$$e_i \triangleq aggregation(\{e \in E : select(e)\})$$

An example of a complex indirect policy rule is one that confirms that the balance sum of all the accounts of a bank customer is over a specific amount. Information that is related with a set of bank accounts, those belonging to the corresponding user, is required for the evaluation of the aforementioned policy rule. Consequently, the *selection* expression defines the subset of the bank accounts that belongs to the specific customer.

2.3 Use Attribute Update Procedures

Attribute values are utilised by UseCON policy rules through the usage decision making process. Consequently, the implementation of a high level policy should also cope with the definition of *use* attribute value update procedures because these values determine the outcome of the policy rule. In UseCON the mutation of attribute values only records security related information that are related with the *use*. Moreover, a categorisation of the use attributes according to the nature of information they record is as follows:

- *Induced Attributes*: Information that can be inferred from entities involved in a *use* (i.e., *subject*, *object*, and *action*) that a specific use instantiates.An example of an *induced* use attribute could be the price of an offered service.
- *Observed Attributes*: Information recorded during the exercise of a *use*. Such info cannot be derived directly from the entities involved in the *use*. Examples of *observed* use attributes are the duration of a *use*, etc.

Update procedures for *induced* attributes can be specified during the definition of the implementation of a high level policy by the policy administrator. However, update procedures for *observed* attributes require the existence of a system function that returns the required value. For example, the specification of a policy, which requires recording the system time whenever a *use* is permitted, is as follows:

$$preUpdate \triangleq [u\ EXCEPT\ !.st = \text{“}activated\text{”},\ !.allowedtime = SystemTime()]$$

where *SystemTime()* is an internal function provided by the system framework that provides the current time. In TLA+, *EXCEPT* is a special purpose operator representing the modification of a function from a state to the next. In that next state all function values are left unchanged unless stated otherwise [1].

Attribute update procedures in UseCON are performed whenever a *use* changes its state (e.g., from *‘requested’* to *‘activated’*). Therefore, during the execution of a *use*, the times of use attribute update in UseCON are represented in Figure 1.

2.4 Transition Systems

Actions in the UseCON model are categorised to those triggered by a *subject’s* request and those operated automatically by the usage control system. More specifically, for every *use* supervised by the usage control system the following actions can be triggered by a *subject*.

- *Request*: This action performs the transition from the *‘init’* state of the *use* to *‘requested’*. Moreover, *request* creates a particular use instance that instantiates the requested *use* and also assigns values to the *id* attribute of the *use*.

[1]For a comprehensive definition of EXCEPT operator we refer the reader to [5].

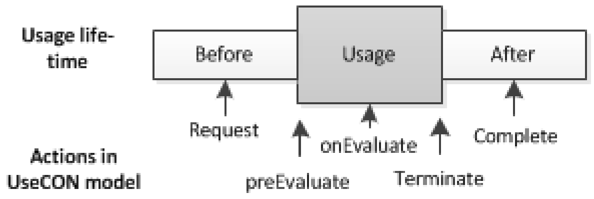

Figure 1: Use attributes updates during the exercise of a *use* in UseCON

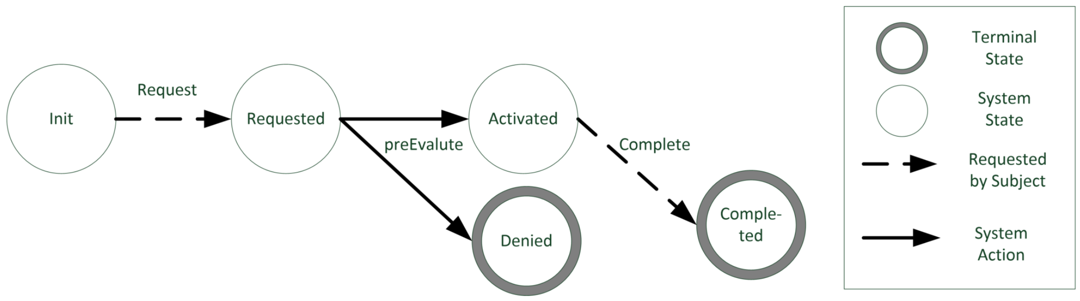

Figure 2: Transition system of a single *use* pre-authorisation UseCON model

- *Complete*: This action changes the state of the *use* from *'activated'* to *'completed'*.

The actions performed automatically by the usage control system, follows:

- *preEvaluate*: This action is performed by the usage control system only when the allowance of the *use* is governed by a pre-authorisation rule. This action changes the state of the *use* to either *'activated'* or *'denied'*, depending on the outcome of the examined policy rule.
- *onEvaluate*: In case the allowance of a *use* is governed by an ongoing authorisation rule, the *onEvaluate* action is performed by the usage control system. If the particular policy rule is satisfied then the state of the *use* does not change. In case the policy rule is not satisfied, the state of the *use* is changed to *'stopped'*.
- *Activate*: This action is performed only when the allowance of the *use* is governed by an ongoing authorization rule. It follows the execution of the *Request* action and changes the state of the *use* from *'requested'* to *'activated'*.

Any of the previous actions, apart from modifying the *st* attribute value, may also update other *use* attribute values. Such modifications in the use attribute values are considered to be implementation specific, as already described in Section 2.3. Moreover, despite the existence of a great number of updates on attribute values, the execution of any of the previous actions is considered to be atomic [2], i.e., a single behavioural step. The transition system for a single *use* UseCON system controlled by a pre and an ongoing authorisation policy rule is depicted in Figures 2 and 3, respectively.

In its initial state, no *uses* are exercised in the system, and thus the first state of every behaviour must satisfy the TLA+ predicate *init*, which defines that the set-variable U is an empty set:

$$Init \triangleq U = \{\}$$

[2]This constraint is realistic due to the fact that a use is recorded centrally on the policy decision point and there is no need for attribute updates of other entities (*subject* or *object*).

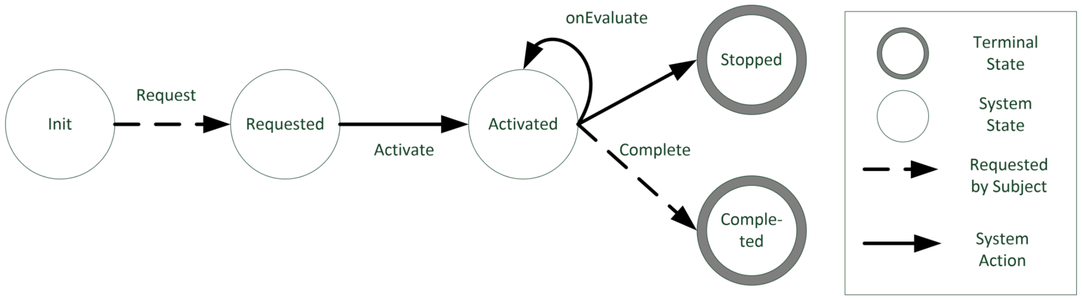

Figure 3: Transition system of a single *use* ongoing authorisation UseCON model

Moreover, according to the time period that a *use* request evaluation is performed, a pre-authorisation and ongoing-authorisation transition systems is created.

2.4.1 Pre-authorisation

The possible actions that can be performed on a pre-authorisation UseCON system are the *use* request for a new *use*, the evaluation of an already requested *use*, or the termination of a *use* that is already executed. Therefore, the *next* action, that describes all the possible next states could be a *request*, *evaluate*, or *complete* action, described as follows:

$$Next \triangleq Request \lor preEvaluate \lor Complete$$

More specifically, the *Request* action selects non-deterministically [3] a new *use x*. This is verified by searching the set of uses U, and returning one that has not already been requested by the subjects or processed by the usage control system. Consequently, in case that this particular x exists, the *request* action creates the corresponding *use* instance that instantiates *use x* and inserts it to the set U. The definition of the *Request* action is as follows:

$$\begin{aligned} Request \triangleq \exists u \in (S \times A \times O):\ (& \\ & \land \forall x \in U:\ (x.sid \neq u[1].id \lor x.aid \neq u[2].id \lor x.oid \neq u[3].id) \\ & \land U' = U \cup \{createUse(u)\}) \end{aligned}$$

The *preEvaluate* action examines if there are any *uses* that have been requested but have not been processed by the usage control system. More specifically, *preEvaluate* examines if there is any use instance with state attribute value equals to *'requested'*. Consequently, the action evaluates the policy rule that governs the allowance of the *use* that the specific use instance instantiates. Based on the outcome of that policy rule the action modifies the state of the use either to *'activated'* or to *'denied'* with *preUpdate*

[3]The non-determinism property is implied by the use of $\exists$ operator. For comprehensive information refer to [5].

and *denUpdate* use attribute update procedures respectively as follows:

$$\begin{aligned} preEvaluate \triangleq \exists u \in U : (&\wedge u.state = \text{“}requested\text{”} \\ &\wedge IF\,(PolicyRule)\; THEN \\ &\qquad U' = (U \setminus \{u\}) \cup \{preUpdate(u)\} \\ &\quad ELSE \\ &\qquad U' = (U \setminus \{u\}) \cup \{denUpdate(u)\}) \end{aligned}$$

The *Complete* action simulates a *subject's* request to terminate the execution of a currently active *use*. If such a use exists, its state attribute value should be equal to *'activated'*. Consequently, the *completed* action modifies the state attribute value from *'activated'* to *'completed'* with the *comUpdate* use attribute update procedure. The *Complete* action is defined as follows:

$$\begin{aligned} Complete \triangleq \exists u \in U : (&\wedge u.state = \text{“}activated\text{”} \\ &\wedge U' = (U \setminus \{u\}) \cup \{comUpdate(u)\}) \end{aligned}$$

CreateUse is the procedure that creates a use instance that has the identities of the *direct subject*, *object*, *action*. The definition of the *createUse* procedure follows:

$$\begin{aligned} createUse(x) \triangleq [&sid(x) = x[1].id, aid(x) = x[2].id, oid(x) = x[3].id, \\ &state(x) = requested, att(x) = k] \end{aligned}$$

All the other use attribute update procedures perform a dual role. Firstly, they alter the *state* use attribute to the desired value (e.g., *preUpdate* to *'activated'* or *comUpdate* to *'completed'*). Secondly, the use attribute update procedures modify the values of other use attributes according to the requirements of the usage control system that they are called to describe. For example, a possible *preUpdate* procedure can be the following:

$$preUpdate \triangleq [u\ EXCEPT\ !.st = \text{“}activated\text{”},\ !.att = value]$$

The same definitions apply to the *denUpdate, comUpdate* procedures.

2.4.2 Ongoing-authorisation

The transition system of the ongoing-authorisation UseCON model is differentiated from the pre-authorisation model in a number of ways. Firstly, in an ongoing model, a *use* that is requested is permitted to be activated without the evaluation of any policy rule. Secondly, at a given time interval [4], an ongoing action is executed *onEvaluate*. Thus, the specification of the *Next* action on an ongoing authorisation model has the following definitions:

$$Next \triangleq Request \vee Activate \vee onEvaluate \vee Complete$$

[4] The determination of the exact interval is left open as an implementation issue.

The *Activate* action searches for the existence of a use with state attribute value equal to *'requested'* and consequently updates it to *'activated'* by executing the *preUpdate* procedure.

$$\begin{aligned} Activate \triangleq \exists\, u \in U :\ (&\wedge u.state = \text{“}requested\text{”} \\ &\wedge U' = (\{U \setminus \{u\}) \cup \{preUpdate(u)\}) \end{aligned}$$

Whereas *onEvaluate* action evaluates an ongoing policy rule, based on this result it either leaves use to *'activate'* state, but it can possibly update the rest use attribute values with *onUpdate* procedure, or modify its state attribute value to *'stopped'* with *termUpdate* use attribute update procedure as follows:

$$\begin{aligned} onEvaluate \triangleq \exists\, u \in U :\ (&\wedge u.state = \text{“}activated\text{”} \\ &\wedge IF(PolicyRule)\ THEN \\ &\qquad U' = (\{U \setminus \{u\}) \cup \{onUpdate(u)\} \\ &\quad ELSE \\ &\qquad U' = (\{U \setminus \{u\}) \cup \{stopUpdate(u)\}) \end{aligned}$$

Moreover, in the case where there is no need for an ongoing use attribute update procedure, the *onUpdate* procedure can be substituted with the *UNCHANGED U* TLA+ operator that leaves the variable *U* unmodified. Here, the semantics of *'request'* and *'complete'* actions are considered to be the same with the pre-authorisation model.

3 Model Checking with TLC

Toolbox is an Integrated Development Environment (IDE), which is designed for the definition and verification of TLA+ specifications [1]. Specifically, the *toolbox* editor provides functionality for the definition and alteration of TLA+ specifications, and supports syntax highlighting. Additionally, an automatic parser checks the defined specifications for syntax errors and presents them accordingly by marking them in the used modules.

The tool in use for the verification of a TLA+ specification in *toolbox* is the TLC model checker. Specifically, TLC explicitly generates and computes all the possible states of a system. However, many times the specification of a system might contain an infinite number of states. TLC handles such specifications, by choosing a finite model of the system and in turn checks it thoroughly. Specifically, the creation of a system's model in TLC requires the definition of its specifications, properties and values of constant parameters [1]. A specification represents all the behaviours that have to be checked. Moreover, the values assigned to constant parameters are utilized for the instantiation of a specification. TLC can check a model for deadlocks, invariants and properties [1]. A deadlock occurs when the model reaches a state in which its next-state action allows no successor states. An invariant is a predicate that is evaluated on a system state. Consequently, an invariant holds on a system specification, if and only if, every state of all the behaviours of a system satisfy that predicate. Properties are temporal formulas that must be evaluated to true for all the behaviours of the model. TLC has some limitations regarding the handling of a subclass of TLA+ specifications and properties that it can check [5]. A very helpful feature of TLC is the fact that when it identifies an error during the verification process, it provides an error trace viewer that allows the exploration in a structured view of the debugging information. Moreover, TLC supports an arbitrary evaluation of states and action formulas in each step of the trace.

Table 1: Safety and liveness properties in UseCON

	Safety		**Liveness**	
	Former state	**Latter state**	**Former state**	**Latter state**
Pre	Completed	Any other state	Requested	Activated or Denied
	Activated	Requested or Denied	Requested	Completed or Denied
	Denied	Any other state	Activated	Completed
Ongoing	Completed	Any other state	Requested	Activated
	Activated	Requested	Requested	Completed or Stopped
	Stopped	Any other state	Activated	Completed or Stopped

3.1 Use management

One of the fundamental properties that can be verified in a system is that of type correctness. Specifically, type correctness is considered to be an invariant which determines that all the variables of the system are assigned with values originating only from a specific set of values. The UseCON specification uses a single variable U which corresponds to the set of system uses. The *invariant* property that defines type correctness in the UseCON model, is defined as follows:

$$TypeCorrectness \triangleq U \subseteq Uses$$

where *Uses* is the set of all records that have the following form (i.e., all the record fields are assigned with values originating from their domains):

$$[s.id\colon SubjectIDS, o.id\colon ObjectIDS, a.id\colon ActionIDS, st\colon USTATE, att\colon ATTDOMAIN]$$

Moreover, the definition of the domains *SubjectIDS, ObjectIDs, ActionIDs* and *USTATE* is:

$$SubjectIDs \triangleq \{s.id\colon\ s \in S\}, ObjectIDs \triangleq \{o.id\colon\ o \in O\}, ActionIDs \triangleq \{a.id\colon\ a \in A\}$$

where

$$USTATE \triangleq \{\text{requested, activated, denied, stopped, completed}\}$$

A *use* is capable of recording detailed historical information about the operation of *uses* in the system. Consequently, a valid implementation of the UseCON model, where multiple *use* processes are operating concurrently, depends on a proper management of the *use* instances that represent these *uses*. Specifically, all *use* instances must adhere only to the state transitions depicted in Figures 2 and 3 for the pre-authorisation and ongoing-authorisation models, respectively. Based on that, a number of *safety* and *liveness* properties can be defined. For example, a *safety* property (a faulty state cannot be reached) states that a *use* instance cannot be evaluated as *'requested'* in its *st* attribute, if it has previously been evaluated as *'completed'*. The semantics, expressed in TLA+, which verify the previous property for all the uses of a system are defined by the following temporal formula:

$$Safety \triangleq \Box(\exists\, u \in U:\ u.st = \text{“completed”} \implies \Box(u.st \neq \text{“requested”}))$$

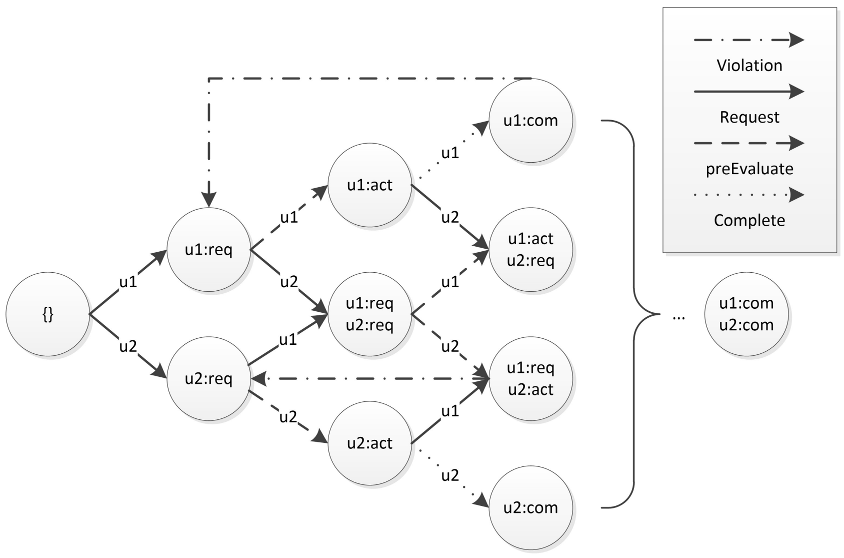

Figure 4: Violation of safety properties

Similar *safety* properties can be defined for all the possible prohibited state transitions. The complete set of prohibited state transitions is depicted in Table 1 under the header *'safety'*. Moreover, two examples that illustrate the violation of *safety* properties are depicted in Figure 4. Specifically, the transition of the *use* instance $u1$ from *'completed'* into *'requested'* results in a violation of a *safety* property. A similar violation is considered during the transition of the *use* instance $u2$ if the state transits from *'activated'* to *'requested'*. In addition, the definition of *liveness* properties in the UseCON model determine all the valid state transitions regarding any use instance. For example, Figure 2 presents that a *use* instance that has at any given state an *st* attribute value that is evaluated to *'activated'*, then its attributed value must be eventually evaluated as *'completed'*. All the possible state transitions that are eligible to be performed are depicted in Table 1 under the header *'liveness'*. This property is defined in TLA+ as follows:

$$Liveness \triangleq \forall u \in U : u.st = \text{“activated”} \leadsto u.st = \text{“completed”}$$

4 Performance Evaluation

The verification of the examined model was performed on a MacBook Pro (Mid 2014) using TLA+ Tool version 1.5.6 and running TLC version 2.12. The system operates on OS X El Capitan and its hardware specifications are: 2.5GHz Intel Core i7 with 16GB 1600 MHz DDR3.

A first set of results were collected by verifying deadlock-freedom for the model of UseCON. Specifically, in a pre-authorisation UseCON model, we consider all the *uses* of the model to be requested, and therefore, to be in either the state of *'activated'* or *'denied'*. The *uses* being activated were finally completed. Therefore, the final state in every *use* must be *'denied'* or *'completed'*. The TLC model checker evaluates all the behaviours of the model and terminates when it reaches to a deadlock, and the actions of the deadlocked behaviour are presented along with the attribute values in each state. Moreover, we performed a verification of the model against safety and liveness properties described in Section 3.1. The verification was finished without raising any errors, but this applies only for the verification of ≤ 12 uses for both the pre and ongoing authorisation models. Although higher numbers of *uses* were consid-

Table 2: Performance evaluation

Authorisation model	**Uses**	**Diameter**	**States found**	**Distinct states**	**Deadlock (seconds)**	**Safety and Liveness (seconds)**
	2	6	21	12	1	3
Pre	8	23	277969	16832	2	5
	12	31	45533665	560128	53	138
	2	7	33	16	1	3
Ongoing	8	25	367873	23808	2	7
	12	37	79112449	1224704	118	241

ered, the verification process had to be interrupted due to the excessive amount of time required for its completion given the number of *uses*. The collected verification results (see Table 2) requires a better understanding of the internal procedures TLC is applying to compute the behaviours of the model (i.e., generation of the transition system). Initially, TLC computes the states that verify the *Init* predicate and inserts them into a set G. For every state $s \in G$, TLC computes all the possible states t that $s \mapsto t$ can be a step in a behaviour. Specifically, TLC substitutes the values assigned to variables by state s for the unprimed variables of the *Next* action, and then it computes all the possible assignment of values to the primed variables that makes the *Next* action true. For every state t found by the former procedure it is added to set G if it does not already exists. The previous two actions are repeated until no new states can be added in G. Therefore, the verification results produced by TLC, are: *Diameter* expresses the number of states in the longest path of G in which no state appears twice; *States found* expresses the number of examined states; *Distinct States* expresses the number of examined distinct states. The verification results produced by TLC for the *pre* and *ongoing* UseCON models are presented in Table 2. An additional column presents the actual running time of the TLC model checker in seconds.

5 Conclusion and Future Work

The application of model checking as a technique resulted in formally verifying the use management functions of UseCON. The trace of the deadlock errors verifies that the defined specifications of the system operate correctly. An advantage of using model checking techniques for the verification of usage control models is the provision of formal guarantees with regards to the correctness of the model, without requiring an implementation of it. Nevertheless, known issues of model checking, i.e., state explosion problem, prevented the timely verification of *uses* when these increase in number. This resulted in providing formal guarantees for the correctness of UseCON for ≤ 12 uses for both the pre-authorisation and ongoing-authorisation models, as depicted in Table 2.

In the future, we aim to demonstrate the verification of complex policy rules supported by UseCON, and specifically investigate the verification of ongoing policy rules. This is of interest since a *use* request might lead to a policy violation in other concurrent *uses*. Thus, any *use* request should be followed by an evaluation of all policy rules in all *uses* in the system to avoid conflicts and violations. Finally, we are considering frameworks, such as secBIP [9], that will allow us to compositionally analyse security properties and generate secure-by-construction systems. Also, we would like to explore guaranteeing security properties by-construction for any number of *uses* in our model through the application of architectures from predefined architecture styles [7] that capture properties of specific access and usage control policies.

References

[1] Microsoft Corporation: *TLA+ Tools*. Available at `http://research.microsoft.com/en-us/um/people/lamport/tla/tools.html`. (visited on February 2018).

[2] Antonios Gouglidis, Ioannis Mavridis & Vincent C Hu (2014): *Security policy verification for multi-domains in cloud systems*. International Journal of Information Security 13(2), pp. 97–111.

[3] Christos Grompanopoulos, Antonios Gouglidis & Ioannis Mavridis (2012): *A use-based approach for enhancing UCON*. In: International Workshop on Security and Trust Management, Springer, pp. 81–96.

[4] Vincent C Hu, Rick Kuhn & Dylan Yaga (2017): *Verification and Test Methods for Access Control Policies/Models*. NIST Special Publication 800, p. 192.

[5] Leslie Lamport (2002): *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*, 1st edition. Addison-Wesley Professional.

[6] Aliaksandr Lazouski, Fabio Martinelli & Paolo Mori (2010): *Usage control in computer security: A survey*. Computer Science Review 4(2), pp. 81–99.

[7] Anastasia Mavridou, Emmanouela Stachtiari, Simon Bliudze, Anton Ivanov, Panagiotis Katsaros & Joseph Sifakis (2016): *Architecture-based design: A satellite on-board software case study*. In: International Conference on Formal Aspects of Component Software, Springer, pp. 260–279.

[8] Jaehong Park & Ravi Sandhu (2004): *The UCON ABC usage control model*. ACM Trans. Inf. Syst. Secur. 7, pp. 128–174.

[9] Najah Ben Said, Takoua Abdellatif, Saddek Bensalem & Marius Bozga (2014): *Model-driven information flow security for component-based systems*. In: From Programs to Systems. The Systems perspective in Computing, Springer, pp. 1–20.

[10] Xinwen Zhang, Masayuki Nakae, Michael J Covington & Ravi Sandhu (2008): *Toward a usage-based security framework for collaborative computing systems*. ACM Transactions on Information and System Security (TISSEC) 11(1), p. 3.

[11] Xinwen Zhang, Jaehong Park, Francesco Parisi-Presicce & Ravi Sandhu (2004): *A logical specification for usage control*. In: Proceedings of the ninth ACM symposium on Access control models and technologies, SACMAT '04, ACM, New York, NY, USA, pp. 1–10.

A An Overview of UseCON

The UseCON model is composed by three entities, namely *subjects*, *objects*, and *actions*. These three entities, together with *uses*, are the core components of UseCON. Decision factors in UseCON are the attribute dependent authorisations and the usage dependent authorisations.

Subjects and *objects* are fundamental concepts, proposed already by access control models. Specifically, a *subject* is an entity that requests the execution of an operation on *object* entities. An *action* entity represents the novel and complicated operations imposed by new computing paradigms. All the security relevant characteristics, including related contextual information of *subjects*, *objects*, and *actions* are described through their attributes. An example of an *action* entity is a money transfer operation from a bank account, where the attributes describe the amount being transferred, the date of the transaction, the currency, etc. A core component of the UseCON model is the *use* component that represents the security related semantics of a *use*. A *use*, is created when a *subject* requests the execution of an *action* on an *object*. A *use* is described through attributes that record the detailed security-relevant characteristics and capabilities that are associated with the requested *use*. Each use is further associated with a *'state'* attribute, which embodies the status of the *use* in progress (see Figure 5). The *'state'* attribute is assigned each time one of the following values:

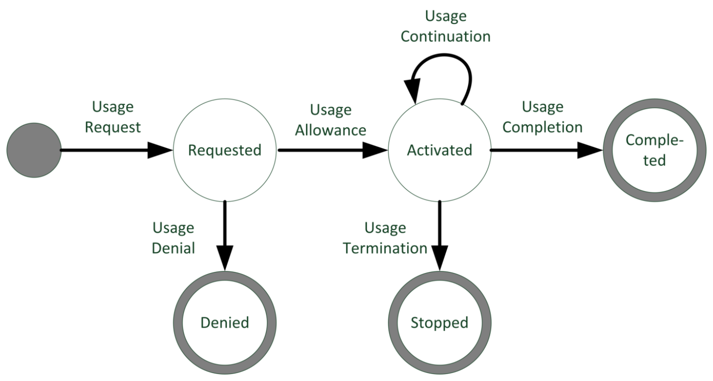

Figure 5: Accomplishment status of a single *use*

- Requested: Upon request for a *use*, the appropriate attributes are associated with the *use* and proper values are assigned to them. The pre-authorisation policy rules, which govern the requested *use*, has not been evaluated yet.
- Activated: The requested *use* has been allowed, as a result of successfully fulfilled pre-authorisation policy rules, and is being executed.
- Denied: The requested *use* has been denied, because it failed to satisfy the pre-authorisation rules.
- Stopped: The allowed/ongoing *use* has been terminated by the system due to a violation of an ongoing authorisation rule.
- Completed: The *use* that has been completed due to a subjects intervention.

An authorisation is the only decision factor in UseCON. However, for the creation of a usage decision, the UseCON model utilises three criteria viz. the (*properties*) of the entities, contextual information, and historical information about *uses*. Therefore, authorisations are categorised into Attribute dependent Authorisations (AdAs) and Usage dependent Authorisations (UdAs) as follows:

- Contextual information and *properties* that describes an entity are associated with the corresponding entity's attributes. The values of these attributes in turn are utilised by AdAs policy rules for the creation of a *use* decision.
- Historical information of *uses* is utilised by UdAs. Specifically, in UseCON, *uses* can record all the information regarding the previous or concurrent *uses* exercised in the system. Consequently, an UdA policy rule utilises the historical information contained into the use attribute values to allow or deny a *use* request.

Integrating authorisations with continuity of decision results into two UseCON sub-models. These are the pre-authorisations and the ongoing-authorisations sub-models. The UseCON elements and the relations between them are depicted in Figure 6.

UseCON presents extended expressiveness, as required by new computing paradigms, not only due to the fact that is able to utilise all the three criteria (i.e., contextual information, *properties* and historical information), but also because these criteria can be related to either *direct* or *indirect* entities or even to any subset of the usage control system entities, e.g. a bank should issue new loans to a customer if and only if the sum of the existing loans of all customers is lower than a given amount. Moreover, UseCON

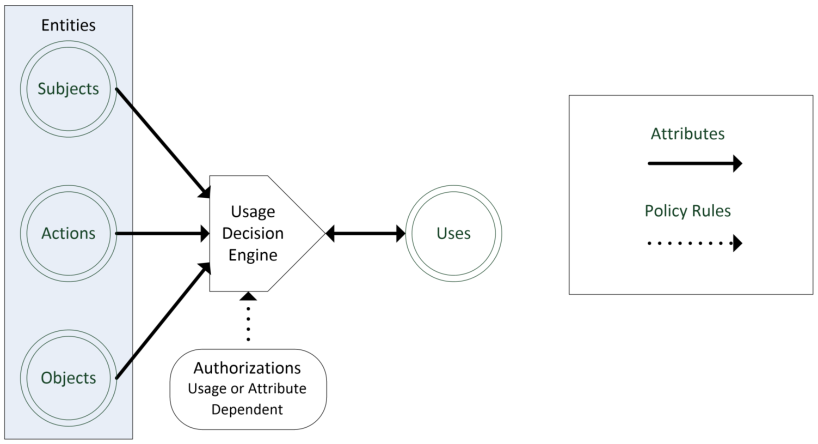

Figure 6: UseCON usage control system

inherently supports the utilisation of historical information of *uses*. Consequently, there is a strict distinction between the functional components of the internal usage control model (e.g., creation and state transition actions of use entities) and the components that define the specific policy implementations of the model (e.g., the creation of the usage decision policy rules).

B Specifications in TLA+

A system's specification in TLA+ follows the *Standard Model*, which is the description of a set of behaviours each representing a possible execution of the system. Every TLA+ specification is composed of *predicates*, *actions* and *temporal formulas*.

Specifically, a *predicate* is a boolean-valued expression built from *variables* and *constants* and is evaluated on a state. The evaluation of a *predicate* in a *state* is performed by calculating the *predicate* expression with the assigned *values* of the included *variables* in this *state*. A formal definition follows:

$$s[[p]] = p(\forall \text{'}u\text{'} : s[[u]]/u)$$

where $p(\forall\text{'}u\text{'} : s[[u]]/u)$ denotes the evaluation of predicate p by substituting every variable u with the assigned value in state s ($s[[u]]$). If a *predicate p* is evaluated as *true* in *state s* it denotes that *predicate p satisfies state s*.

An *action* denotes a relation between pairs of states of the system (denoted as system *steps*). Moreover, an *action* is a boolean-valued expression built from constants and *primed* and *unprimed* variables. *Primed* variables refer to new states while *unprimed* refer to old states. A formal definition of actions follows:

$$s[[a]]t = a(\forall \text{'}u\text{'} : s[[u]]/u, t[[u]]/u')$$

We say that *action a satisfies states s,t*, or that s,t is an *a-step* if the evaluation of the action, with unprimed variables assigned value from state s and primed variables assigned value from state t, is true. A *predicate* can be considered as a special *action* that is evaluated only on the first state of a step.

A *temporal formula* in TLA+ is a boolean valued expression that is evaluated on behaviours. More specifically, a *temporal formula F* is composed by *action* and *predicates* combined with logical and temporal operators. An *action* / *predicate* can be considered as a special *temporal formula* that is evaluated on the first *step* / *state* of the behaviour. Some of the fundamental temporal operators in TLA+ follow:

- *Always* $\Box$. The formula F must satisfy every suffix of the behaviour. The semantics of the *always* temporal operator follows:

  $$< s_0, s_1, \ldots > [[\Box F]] \triangleq \forall n \geq 0: \; < s_n, s_{n+1}, \ldots > [[F]]$$

  where the $\triangleq$ operator utilised in an expression $id \triangleq exp$ defines *id* to be synonymous with the expression *exp*. Replacing *id* by *exp* does not change the meaning of the specification.

- *Eventually* $\Diamond$. The formula F must satisfy some states of the behaviour. The semantics or *eventually* can be defined by utilizing the *always* temporal operator as follows:

  $$\Diamond F \triangleq \neg \Box \neg F$$

- *Leads to* $\leadsto$. The formula $F \leadsto G$ asserts that whenever F is true, G is eventually true. That is G is true at the same time with F or some time later. The *leads to* operator can be defined utilizing the previous temporal operators as:

  $$F \leadsto G \triangleq \Box(F \implies \Diamond G)$$

Consequently, all the accepted behaviours of a system can be specified in TLA+ with a temporal formula called *specification* of the following form:

$$Spec \triangleq Init \land \Box Next$$

where Init is a predicate and Next is an action. A behaviour satisfies Spec if the first state of the behaviour satisfies Init and every consequent step satisfies Next. However, a well-defined specification should allow *stuttering* steps (steps that leave the system variables unchanged). Thus, the specification has the following form:

$$Spec \triangleq Init \land \Box[Next]_{<v1,\ldots,vn>} \tag{1}$$

where $[Next]_{<v_1,\ldots,v_n>}$ is defined as:

$$[Next]_{<v_1,\ldots,v_n>} \triangleq Next \lor ((v_1' = v_1) \land \ldots \land (v_n' = v_n))$$

Formula (1) demands that the transition from one state to another is either a Next-step or a stuttering step that leaves the system variables unchanged.

However, the TLA+ specification expressed in (1), also describes behaviours that may stop at any point, including a behaviour that starts in a valid initial state and takes no step. *Weak fairness* property of the action Next ($WF_{vars}(Next)$) asserts that behaviours are not allowed to stop in a state in which *Next* action is enabled (Action A is enabled in a state s if there exists a state t such that $s \rightarrow t$ is an Next-step). Therefore, the specification of a system that permits stuttering steps and supports the weak fairness property for action Next has the following form:

$$Spec \triangleq Init \land \Box[Next]_{<v_1,\ldots,v_n>} \land WF_{vars}(Next)$$

A property in TLA+ is described together with a system's specification and has the form of an *invariant* or a *temporal formula*. An *invariant* is a predicate that is evaluated on a system state. Consequently, an *invariant* holds on a system specification, if and only if, every state of all the behaviours of a system satisfy that predicate. Moreover, a *temporal formula* is evaluated on a system behaviour. Consequently, a *temporal formula* hold on a specification, if and only if, it is satisfied by all the behaviours of the system.