Noname manuscript No.
(will be inserted by the editor)

Specifying and Verifying Usage Control Models and Policies in TLA$^+$

Christos Grompanopoulos · Antonios Gouglidis · Anastasia Mavridou

the date of receipt and acceptance should be inserted later

Abstract Novel computing paradigms, e.g. the Cloud, introduce new requirements with regard to access control such as utilisation of historical information and continuity of decision. However, these concepts may introduce an additional level of complexity to the underpinning model, rendering its definition and verification a cumbersome and prone to errors process. Using a formal language to specify a model and formally verify it may lead to a rigorous definition of the interactions amongst its components, and the provision of formal guarantees for its correctness. In this paper, we consider a case study where we specify a formal model in TLA$^+$ for both a policy-neutral and policy-specific UseCON usage control model. Through that, we anticipate to shed light in the analysis and verification of usage control models and policies by sharing our experience when using TLA$^+$ specific tools.

Keywords Authorisation models · Concurrency · Model checking · UseCON

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

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

Anastasia Mavridou
SGT / NASA Ames Research Center
Moffett Field
CA 94035, USA
E-mail: anastasia.mavridou@nasa.gov

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*. Based on a set of existing policy rules an authorisation process is required to take a decision for granting or denying a *subject* to access an *object* in a system. Thus, access control systems are considered to be amongst the most critical of security components. Their importance is highlighted in a special publication by NIST, where formal verification for various access control policies is proposed for application [12].

New computing paradigms, e.g. the Cloud [7], require further investigation of access control systems. This eventually introduces the need for models, e.g. usage control [9], 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, e.g. utilisation of historical information and continuity of decision. By utilising historical information, the allowance of a usage does not depend only on the participant's properties, but also on the previous usages that have been exercised, resulting in 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 sometime. 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 evolve the concept of access control to that of usage control.

Formal methods have proven to be particularly useful in the process of system design and software devel-

opment [2]. With regard to usage control models, temporal logic can be used to provide unambiguous semantics of the functions supported by the models, such as utilisation of historical information of usages and continuity of decision [17]. Usage control systems are concurrent and characterised by non-determinism due to the potential of a *subject* to arbitrarily request or terminate the use of an *object*. As discussed in [32], 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 [9] is a usage model that provides enhanced expressiveness compared to existing access/usage control models and applicable in new computing paradigms [9]. This is achieved by introducing the *use* entity that is the theoretical representation of a real-world usage exercised by a subject on an object. Therefore, by utilising information from uses, the UseCON model is capable of taking into consideration historical data during the usage control decision process.

In our case study, we selected to use the enhanced version of Temporal Logic of Actions (TLA$^+$) [16] in order to specify UseCON and further analyse and verify the proposed model using TLA$^+$ tools, i.e. TLA$^+$ Toolbox. TLA$^+$ has been selected over other specification languages (e.g. Alloy) due to its support of temporal quantifiers and reliance on the notion of actions to express dynamism. Although dynamic properties may be supported by Alloy, that may be an error-prone task since dynamism has to be modelled explicitly by the user [19].

In [6], we have defined and verified the core operations of UseCON. In this paper, we provide a specification of the policy-neutral UseCON model that does not include any constraints on the number of requested uses (which was set in [6] to a maximum of 3). Furthermore, we provide policy-specific examples based on the two supported authorisation models in UseCON, and we also introduce faults in the previous examples. We further analyse and verify the new specifications through the extended use of TLA$^+$ Toolbox. Specifically, the TLC model checker was used for a) the verification of the model's internal management procedures (for the policy-neutral UseCON model); b) the verification of a list of properties for the policy-specific examples; c) the successful detection of the introduced faults in the policies.

The remainder of this paper is organised as follows: Section 2 elaborates on related work (i.e. usage control models and verification) and provides a comparison of that with our approach. Section 3 provides necessary background information on the UseCON model and the TLA$^+$ specification language. Section 4 presents a TLA$^+$ specification of UseCON. Section 5 provides examples of UseCON-supported access control policies formalised with TLA$^+$. In Section 6, we verify the specified models and policies; elaborate on the detection of introduced faults in the policies; and provide performance results of the TLC model checker. Section 7 concludes the paper.

2 Related work

A number of formal definitions for usage control models have been specified to express the newly introduced concepts (e.g. continuity of decision, use of historical information) in the usage decision making process. A model defined in [33] by Zhang et al. was specified using TLA to highlight the enhanced expressiveness of UCON compared to existing models. In general, a first task when specifying a system is to choose what part of a system will be specified and then decide on the abstraction level of that specification. Thus, atomicity is an important aspect of that level of abstraction, with grain of atomicity described to be *'... the choice of what system changes are represented as a single step of a behaviour.'* [16]. The grain of atomicity in Zhang's et al. specification is set at the internal actions of the model, separating the actions that update the state of a usage from those that update the attribute value of an entity. The specification described in [33] assumes a single usage process, and thus, it may raise questions with regard to the operational phase of usage control systems, when considering multiple usages running concurrently.

In [35] [36] Zhang et al. investigate a safety problem related to the creation of new objects in a UCON system. The defined formal model permits the existence of multiple usages. However, it sets the grain of atomicity to that of an entire usage to eliminate the impact of concurrent operations into the internal state transitions of the usage states. The language used to specify the system appears to be a Z-like language. Both specifications proposed by Zhang et al. are capable of expressing not only the model itself, but also the policies supported by the specified models, which is achieved by specifying both the authorisation predicates and attribute update procedures.

A specification of a usage control model was developed in [14] by Janicke et al. to support the need for continuous on-going scenarios. Specifically, within an initial usage (i.e. a session), a number of *'internal'* usage requests may be performed. The latter is required to be labelled by atomic actions to support immediate revocation of usages. This specification is developed in Interval Temporal Logic (ITL) [4] and supports concurrent usages only within sessions. An additional formal

Table 1 Comparison of UseCON with related work

Model	Main attributes	On-going	Concurrency	Grain of atomicity	Fairness	Policy	Language	Model checker
Zhang et al. [33]	Single usage process	Yes Yes	No	Internal actions	No	Yes	TLA	No
Zhang et al. [35] [36]	Component creation	Yes	No	Single usage	No	Yes	Z-like	Custom
Janicke et al. [14]	Series of usages in a session	Yes	Yes (session)	Internal actions	No	No	ITL	No
Martinelli et al. [20]	Define activities in a UCON process	Yes	No	Internal actions	No	Yes	POLPA	No
UseCON [6]	Extended expressiveness	Yes	Yes	Internal actions	Yes	Yes	TLA$^+$	TLC

model based on the POlicy Language based on Process Algebra (POLPA) is presented in [20]. This specification presents only the basic features of UCON and does not deal with any concurrency issues or present policy implementation paradigms. Properties that may be verifiable in the above models appear to be missing from both [14] and [20].

In [33], both the expressiveness and flexibility of the usage control model were presented through several policies (e.g. RBAC, Chinese Wall). The formal model in [35] studied the safety problem in usage control, concluding that only a very restricted pre-authorization subset of usage control models may support the safety problem. Work in [27] also examined the safety problem in UCON policies, which appears to generalise the restrictions imposed by [35]. Specifically, the specification proposed in [27] permits the concurrent execution of usages and allows attribute values to get values over infinitive domains using an algebraic structure. A comparison of UseCON's capabilities described in [6] against the above mentioned work is depicted in Table 1.

A first attempt to formally verify usage control policies with a model checker is presented in [25]. Specifically, a formal model of simple usage control policies using an extension of Linear Temporal Logic (LTL). Policies were verified using the NuSMV model checker [5]. The authors in [26] verified a policy implementation of a usage control system using the SPIN model checker [10]. The implementation was designed for a Web based conference management application and supports concurrent applications through a common communication channel. However, the usage scenario lacks of support for ongoing rules. Separation of duties policies in usage control are studied in [18], where initially they define a set-based specification of separation of duty policies and then they adapt it to attribute values of usage control. This resulted in the application of already existing static mutually exclusive attribute (SMER) constraints.

3 Background

In this section, we give an overview of UseCON and provide a specification of it in TLA$^+$. The reasons for selecting TLA$^+$ and the TLC model checker are mentioned in the introduction section. In addition to that, a complementary reason for selecting TLA$^+$ lies in the fact that based on our personal experience, we find TLA$^+$ to be a versatile language that may be used by both researchers and engineers, and thus may facilitate the development of access control systems.

3.1 An Overview of UseCON

The UseCON model is composed of 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 could be 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, which represents the security related semantics of a usage. A *use*, is created when a *subject* requests the execution of an *action* on an *object* where (s, o, a) are the *direct* entities, while the rest are the *indirect* entities of the usage. A *use* is described through at-

tributes that record the detailed security-relevant characteristics and capabilities that are associated with the requested usage. Each use is further associated with a *'state'* attribute, which embodies the status of the usage in progress (see Figure 1). The *'state'* attribute is assigned each time one of the following values:

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

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

- Contextual information and *properties* that describe 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 usage decision.
- Historical information of usages is utilised by UdAs. Specifically, in UseCON, uses can record all the information regarding the previous or concurrent usages exercised in the system. Consequently, an UdA policy rule utilises the historical information contained into the use attribute values to allow or deny a usage 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 2.

UseCON is an expressive model, as required by the 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 inherently supports the utilisation of historical information of usages through use entities. 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).

3.2 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 following [15] is:

$$s[[p]] \triangleq p(\forall\ 'u'\colon s[[u]]/u)$$

where $p(\forall 'u'\colon s[[u]]/u)$ 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 *is satisfied in state* s.

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

$$s[[a]]t \triangleq a(\forall\ 'u'\colon s[[u]]/u, t[[u]]/u')$$

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

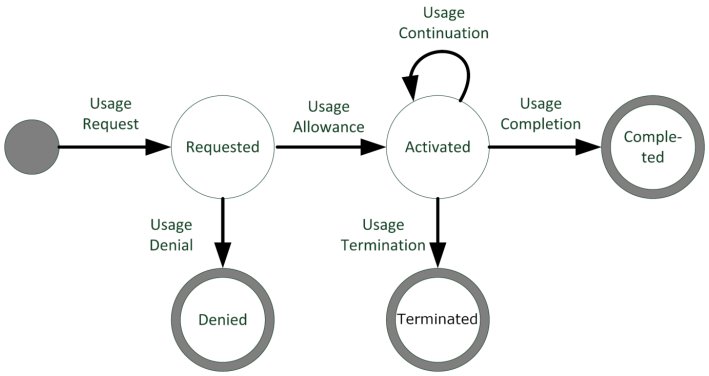

Fig. 1 Accomplishment status of a single usage

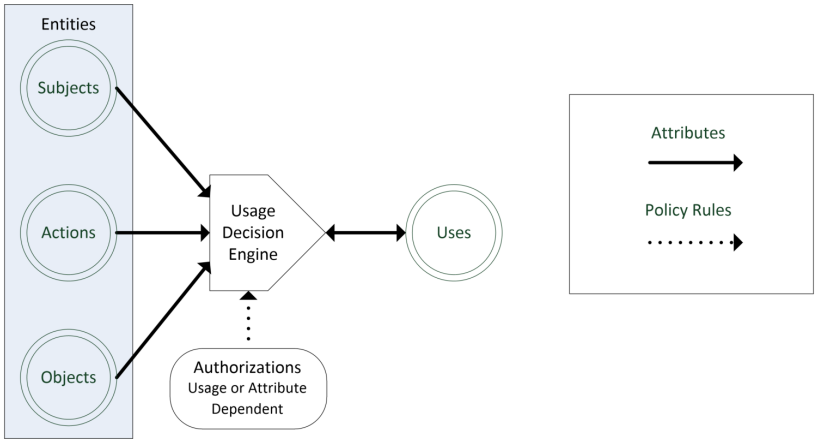

Fig. 2 UseCON usage control system

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:

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

  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 utilising the *always* temporal operator as follows:

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

- *Leads to* $\rightsquigarrow$. The formula $F \rightsquigarrow G$ asserts that whenever F is true, G is eventually true. That is G is true at the same time with F or sometime later.

The *leads to* operator can be defined utilising the previous temporal operators as:

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

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

$$Spec \triangleq Init \wedge \Box Next$$

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:

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

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

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

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. The *weak fairness* property of the action Next (denoted as $WF_{vars}(Next)$) 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 a A-step). Therefore, the specification of a system that permits stuttering steps and supports the weak fairness property for action Next has the following form:

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

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.

4 Specification of UseCON in TLA$^+$

In the following, we provide a formal specification of the UseCON model in TLA$^+$. 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.

4.1 Main Elements

A *use* in UseCON represents a request to execute an *action (a)* from a *subject (s)* on an *object (o)*. Every *s*, *a*, and *o* is characterised by a unique identification value called *sid*, *aid* and *oid*, respectively. The sets *SID*, *AID*, and OID contain all the unique identification values for subjects, actions and objects that are used in the usage control system. The values of *SID, AID* and *OID* remain the same throughout the lifetime of the usage control system and the are declared as sets with constant values in the specification with the following declaration:

CONSTANTS SID, OID, AID

To verify the model it is required to assign values to *SID*, *OID* and *AID*, using expressions of the following form:

$$AID = \{\text{“aid1”}, \text{“aid2”}, \ldots, \text{“aidn”}\}$$
$$OID = \{\text{“oid1”}, \text{“oid2”}, \ldots, \text{“oidn”}\}$$
$$SID = \{\text{“sid1”}, \text{“sid2”}, \ldots, \text{“sidn”}\}$$

In UseCON attributes can be assigned to subjects, objects and actions to provide additional characteristics to them through their attribute value. However, the usage control system does not modify automatically the attribute value of system entities. Thus, attribute values of system entities are proposed to be updated manually by the usage control administration model, which is not covered by the proposed specification. Therefore, each of the subjects, objects and actions are represented in the specification by a single value, i.e. sid, oid and aid, respectively. As it is highlighted by their specification, subjects, objects and actions have a similar role in the UseCON specification. Therefore, onward in this paper, we use the term *entity* to refer to a subject, object or action. Moreover, the set of *entities (E)* is described as follows

$$E = S \cup O \cup A$$

A usage request in UseCON results into the creation of a *use*. A tuple composed of the identity values of the *subject*, *object* and *action* participating in the usage (i.e. *sid*, *oid* and *aid*, respectively) uniquely identifies a single *use* and it is denoted as the *uid* of that *use*. The set of use identifiers (UID) is the cartesian product of the *SID*, *AID* and *OID*, defined as follows:

$$UID \triangleq (SID \times AID \times OID)$$

The characteristics of a use are represented through attribute values. More specifically, every *use* must contain a *status* attribute to represent the current status of a use. Its value may be one of the following: *'init'*, *'requested'*, *'activated'*, *'denied'*, and *'completed'* for the preUseCON authorisation model;in the ongoing authorisation model, the *'denied'* value is replaced by *'terminated'*. Thus, the set of *uses* that instantiate usage requests from subjects *s* to objects *o* for actions *a* is a TLA+ function *U* having *UID* set as its domain and range the set of records *USES* with a *status* field.

The set of records that specify the *use* entities is defined in TLA$^+$ as follows:

$$USES \triangleq [status : USTATUS]$$

and *USTATUS* is the set with all the *status* attribute values and it is defined for the pre-authorisation model as:

$$USTATUS \triangleq \{\text{“init”}, \text{“requested”}, \text{“activated”}, \text{“denied”}, \text{“completed”}\}$$

and for the ongoing-authorisation model as:

$$USTATUS \triangleq \{\text{“init”}, \text{“requested”}, \text{“activated”}, \text{“terminated”}, \text{“completed”}\}$$

Therefore, the set of all functions with domain *UID* and range a subset of *USES* is defined as $[UID \rightarrow USES]$.

It is worth mentioning that in TLA$^+$ a function defines also a table. Specifically, a TLA$^+$ table is a function with index values taken from the function's domain, and table values assigned from the function's range. Therefore, the proposed specification uses only a single variable U that is a TLA$^+$ function/table. Table U assigns each index from the cartesian product of the constant sets of *SID, OID* and *AID* (i.e. all the possible usage that may be requested) to a use record in *USES*, which contains the use attributes.

The definition of U as the variable in the specification is described as:

VARIABLES U

In the beginning of a TLA$^+$ specification, several modules may be included for supporting different operators. Our specification includes *Integers* which encompass arithmetic operators, e.g. and *TLC* module is included to use TLA$^+$ specific expressions such as the *RandomElement*. This is declared as follows:

EXTENDS *Integers*, *TLC*

4.2 Decision Making in UseCON

In the current specification of the UseCON model, the U variable records all the usages that have been previously exercised in the system. Policy rules in UseCON are able to utilise information from both entities and use attribute values. Specifically, a UseCON policy rule that governs the allowance of a usage request from a subject s on an object o with an action a, is a Boolean-valued expression, i.e. a TLA$^+$ predicate, which can be one of the following types:

- *Direct Policy Rules*: The expression of a direct policy rule consists of constant values or attributes of the direct entities of the usage. The definition of such an expression is as follows:

$$DirectPolicy(uid) \triangleq expression(e_1, e_2, \ldots, e_n, l_1, l_2, \ldots, l_m)$$

  Where $e_i, i \in 1..n$ are attributes of the direct entities, $l_i, i \in 1..m$ are constant values and *uid* is the identifier of the usage that is examined for allowance or not. Access to the identifiers of a direct entity is achieved through the *uid*. Specifically, *uid*[1] is the subject's id, *uid*[2] is the action's id and *uid*[3] is the object's id.
- *Indirect Policy Rules*: The expression of an indirect policy rule consists of attribute values stemming not only from direct, but from indirect entities too. In this case, the system searches in variable U for the existence of a usage with a specific indirect entity. The definition of such an expression is as follows:

$$IndirectPolicy(uid) \triangleq \exists\, uid_{indirect} \in U : select(e_1, e_2, \ldots, e_n, l_1, l_2, \ldots, l_m)$$

  Where the *select* expression searches the existence of a specific usage, with $e_i, i \in 1..n$ being the attributes of the direct entities (discovered through *uid*) or indirect entities (discovered through $uid_{indirect}$), and $l_i, i \in 1..m$ are constant values.
  An example of an indirect policy rule could be an expression that evaluates into true or false, if the parent of a child has paid the usage of a toy in an amusement park. In this example, the child is the direct entity and the parent is an indirect entity. The select expression should take into account the fact that there is a use in U where the 'parent' id is the *sid*, having a payment action.
- *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 also to a subset of entities. Such complex policies can be supported in UseCON through complex indirect policy rules. More specifically, the system may search in variable U for all the entities that satisfy a desired (*select*)

predicate. The semantics of an expression of a complex indirect policy rule is as follows:

$$ComplexIndirectPolicy(uid) \triangleq \\ \exists\, uid_{indirect_1}, uid_{indirect_2}, \ldots, uid_{indirect_k} \in U : \\ select(e_1, e_2, \ldots, e_n, l_1, l_2, \ldots, l_m)$$

Where parameters $e_i, i \in 1..n$ are attributes of the direct entities (discovered through uid) or indirect entities (discovered through $uid_{indirect_i}, i \in 1..k$), and $l_i, i \in 1..m$ are constant values.
An example of a complex indirect policy rule is one that confirms that the balance of a number of bank accounts is over a specific amount. Information that is related to a set of bank accounts, those belonging to the corresponding account holders, is required for the evaluation of the aforementioned policy rule. Consequently, the selection expression defines the subset of the bank accounts that belongs to specific customers.

Moreover, two or more UseCON policy rules can be combined together with logical operators as follows:

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

where $\otimes$ is a logical operator (e.g. AND, OR), and $p_i, i \in 1..n$ is a policy rule. The policy rule p_i can be a direct, indirect or complex indirect policy rule.

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

The actions performed automatically by the usage control system follow:

- *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 status 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 status of the *use* does not change. In case the policy rule is not satisfied, the status of the *use* is changed to *'terminated'*.
- *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 status of the *use* from *'requested'* to *'activated'*.

Any of the previous actions, modifythe *status* attribute value and are considered to be atomic , 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 3 and 4, respectively.

Figures 3 and 4, depict that UseCON is a policy neutral model that records the usage requests as they are submitted by the system entities. However, a policy specification can be easily introduced with a predicate (see rectangle shape in figures) that determines the granting or not of a specific usage request. In this paper, we provide four specifications. Two generic specifications (i.e. policy independent) where the predicate randomly evaluates to TRUE or FALSE in both preUseCON and ongoing models, with two additional policies described in the next section. 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*. The *Init* predicate defines that all the uses in the table-variable U must be in the *'init'* status:

$$Init \triangleq \\ U = [uid \in UID \mapsto [status \mapsto \text{“init”}]$$

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

4.3.1 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 completion of a *use* that is already executed. Therefore, the *next* action, that describes all the possible next states could be a *request*, *pre-evaluate*, or *complete* action, described as follows:

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

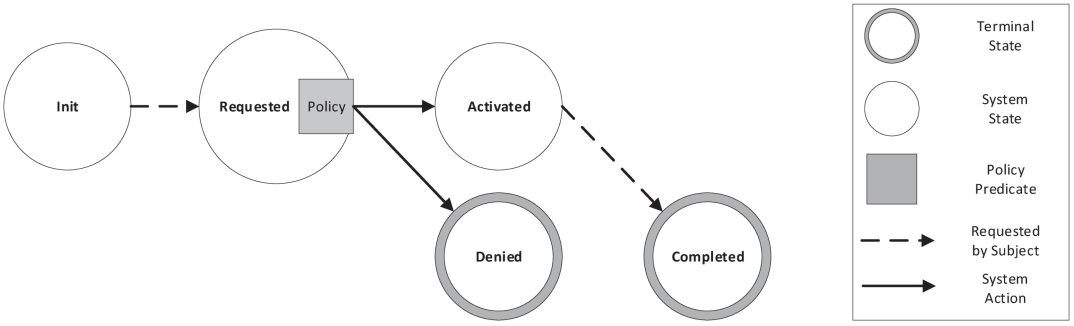

Fig. 3 Transition system of a single *use* pre-authorisation UseCON model

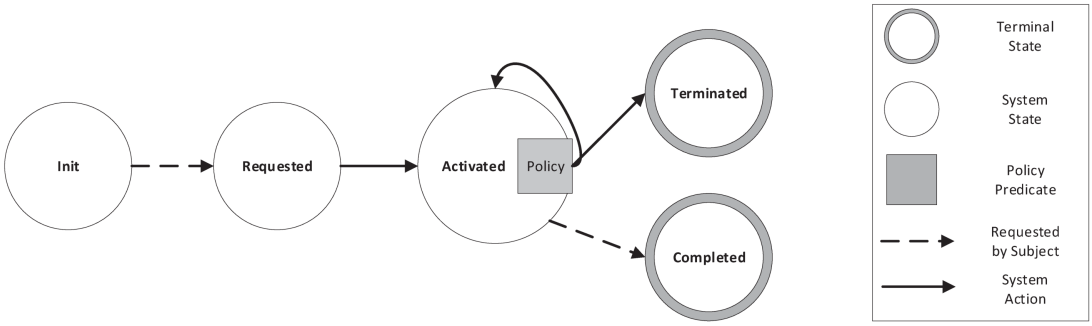

Fig. 4 Transition system of a single *use* ongoing authorisation UseCON model

More specifically, the *Request* action specifies a random usage request. Firstly, *Request* selects non deterministically [1] a use identifier *uid*, of a usage that is in the *'init' status*. Consequently, *Request* modifies the *status* attribute value from *'init'* to *'requested'*. The definition of the *Request* action is as follows:

$$
\begin{aligned}
&Request \triangleq \\
&\quad \land \exists\, uid \in UID : \\
&\quad\ \land U[uid].status = \text{“init”} \\
&\quad\ \land U' = [U \text{ EXCEPT } ![uid].status = \text{“requested”}]
\end{aligned}
$$

The *preEvaluate* action examines if there are any *uses* that have been requested, but have not been processed by the usage control system. Specifically, *preEvaluate* examines if there is a use instance with status 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 status attribute value of the use either to *'activated'* or to *'denied'*, respectively.

In order to verify the correctness of the *use* entity management of our specification we specify a policy-neutral predicate by utilising the *RandomElement* predicate of TLA$^+$ which arbitrarily chose the value *TRUE* of *FALSE*, as follows:

$$PolicyNeutral(uid) \triangleq RandomElement(\{TRUE, FALSE\})$$

Therefore the specification of the preEvaluate action follows:

$$
\begin{aligned}
&preEvaluate \triangleq \\
&\qquad \exists\, uid \in UID : \\
&\qquad\quad \land U[uid].status = \text{“requested”} \\
&\qquad\quad \land U' = [U \text{ EXCEPT } ![uid].status = \\
&\qquad\qquad \text{IF } (PolicyNeutral(uid)) \\
&\qquad\qquad\ \text{THEN “activated”} \\
&\qquad\qquad\ \text{ELSE “denied”}]
\end{aligned}
$$

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

$$
\begin{aligned}
&Complete \triangleq \\
&\quad \exists\, uid \in UID : \\
&\qquad \land U[uid].status = \text{“activated”} \\
&\qquad \land U' = [U \text{ EXCEPT } ![uid].status = \text{“completed”}]
\end{aligned}
$$

[1] The non-determinism property is implied by the use of the $\exists$ operator. For comprehensive information refer to [16].

4.3.2 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 [2], an ongoing action is executed *onEvaluate*. Thus, the specification of the *Next* action on an ongoing authorisation model has the following definitions:

$$Next \triangleq \\ Request \lor onActivate \lor onEvaluate \lor Complete$$

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

$$onActivate \triangleq \exists\, uid \in UID : \\ \land\, U[uid].status = \text{“requested”} \\ \land\, U' = [U \text{ EXCEPT } ![uid].status = \text{“activated”}]$$

Whereas *onEvaluate* action evaluates an ongoing policy rule and based on the result: 1) it does not update the use's state, which remains equal to *'activated'* or 2) it modifies its state attribute value to *'terminated'*, as follows (*PolicyNeutral* predicate is also utilised):

$$onEvaluate \triangleq \exists\, uid \in UID : \\ \land\, U[uid].status = \text{“activated”} \\ \land\, U' = [U \text{ EXCEPT } ![uid].status = \\ \quad \text{IF } (PolicyNeutral(uid)) \\ \quad\quad \text{THEN “activated”} \\ \quad\quad \text{ELSE “terminated”}]$$

The source code of the specification in TLA$^+$ is available on Github at [8].

5 Examples of Policies

The development of an access/usage control system is a multi-layer process that results in the definition of an access/usage control policy, model and mechanism [29]. A policy declares the high level directives that regulate access to resources, while the model is a formal representation of the system. Moreover, the mechanism defines the low-level hardware and software functions that implement the desired policy. UseCON is a general purpose usage control model that is capable of supporting a wide range of high level policies [9]. However, to implement a specific high-level policy using UseCON, it requires to specify policy rules. Policy rules are required because they are responsible for getting a decision regarding a usage allowance and they are represented in our specification by predicates. A motivating scenario that uses the pre-authorisation UseCON model follows:

Usage Scenario 1: A shareholder advice company requires from its clients to sign a non-disclosure agreement for every content they request to view. The non-disclosure agreement action can be modeled using an *"aid1"* action and the view content action with *"aid2"*. Thus, the *PolicyNeutral* predicate in the *preEvaluate* action should be replaced by another predicate to implement the above policy. The predicate should be evaluated to true for the non-disclosure agreement action; or will be evaluated to true for any other action only if the non-disclosure agreement action has been completed for the same *s*, *o* tuple. Such a predicate can be described as follows:

$$Policy1(u1) \triangleq \lor\, u1[2] = \text{“aid1”} \\ \lor \exists\, u2 \in UID : (\\ \quad \land\, SameSO(u1,\, u2) \\ \quad \land\, u2[2] = \text{“aid1”} \\ \quad \land\ \ U[u2].status = \text{“completed”}$$

where *SameSO* is an expression able to validate that usages *u1* and *u2* consist of the same subject and object tuple. The semantics of *SameSO* expression, can be defined as follows:

$$SameSO(u1,\, u2) \triangleq \quad \land\, u1[1] = u2[1] \\ \land\, u1[3] = u2[3]$$

The rationale behind the *Policy1* expression is: if subject *s* requests to perform action *a* on object *o*, the requested action permitted if and only if one of the following holds: 1) action *a* is the non-disclosure agreement (*"aid1" id*); or 2) if action *a* wants to view the object (*"aid2" id*), the same subject should have already completed a non-disclosure agreement action in the past on the same object.

Another usage scenario that highlights the expressiveness of the UseCON model and uses the ongoing authorization model follows.

Usage Scenario 2: A multimedia content provider has two categories of users: *'premium'* and *'free'*. A policy rule determines that whenever a free licence user has access to an object along with a premium user, the free licence user may be denied use of that object to ensure a better user experience for the premium user.

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

This usage scenario follows the ongoing UseCON model, which means that when a usage is requested it is always activated. However, periodically [3], the model evaluates a predicate on all the activated usages. If the predicate is evaluated to true the usage continues, otherwise the use is immediately terminated. Here we assume that subjects *"sid1"* and *"sid2"* refer to a free and premium member, respectively, and access may be requested by both users on the same object [4]. Thus, the *PolicyNeutral* predicate in the *OnEvaluate* action should be replaced by another predicate to implement the above policy. The predicate should evaluate to true if the usage is exercised by a premium user; or if the usage is exercised by a free member, there is no usage activated on the same object by a premium user. The semantics of a predicate that implements the above logic may be specified as follows:

$$
\begin{aligned}
Policy2(u1) \triangleq\ & \lor\ u1[1] = \text{“sid2”} \\
& \lor\ \land\ u1[1] = \text{“sid1”} \\
& \quad \land \neg(\exists\, u2\ \in UID : \\
& \qquad \land\ u2[1] = \text{“sid2”} \\
& \qquad \land\ u1[3] = u2[3] \\
& \qquad \land\ U[u2].status = \text{“activated”})
\end{aligned}
$$

6 Model Checking with TLC

Toolbox is an Integrated Development Environment (IDE), which is designed for the definition and verification of TLA$^+$ specifications. 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. A specification represents all the behaviours that have to be checked. Moreover, the values assigned to constant parameters are utilised for the instantiation of a specification. TLC can check a model for deadlocks, invariants and properties. 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 [16]. 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.

6.1 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 is a table, TLA$^+$ function, with domain the set of use identifiers, and range the set records with the use attributes. The *invariant* property that defines type correctness in the UseCON model, is defined as follows:

$$TypeOK \triangleq\ U \in [UID \rightarrow USES]$$

where *UID* is the set of uses IDs and *USES* is the set of records that specifies the use entities, as these are described in Section 4.1.

A *use* is capable of recording detailed historical information about the operation of system usages. 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 3 and 4 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'* or *'activated'* or *'denied'* in its *status* attribute, if it has previously been evaluated as *'completed'*. The semantics in TLA$^+$ that verify safety properties in a *pre* and *ongoing* authorisation model

[3] The time period in which the ongoing predicates are evaluated is implementation-specific by the UseCON model

[4] Since the specification of attributes is not present in the current model, we express the different categories of users through their IDs.

Table 2 Safety properties in UseCON

	Safety properties	
	Current state	**□¬ state**
Pre	Completed	Init or Requested or Activated or Denied
	Activated	Init or Requested or Denied
	Denied	Init or Requested or Activated or Denied
	Requested	Init
Ongoing	Completed	Init or Requested or Activated or Denied
	Activated	Init or Requested
	Terminated	Init or Requested or Activated or Completed
	Requested	Init

Table 3 Liveness properties in UseCON

	Liveness properties	
	Current state	**◇ state**
Pre	Activated	Completed
	Init	Requested
	Requested	Activated or Denied
Ongoing	Requested	Activated
	Init	Requested
	Activated	Completed or Terminated

(depicted in Table 2) are expressed by the following temporal formulae:

$$\begin{aligned}
&\mathit{Safety_Pre_Completed} \triangleq \Box(\forall\, \mathit{uid_n} \in \mathit{UID} :\\
&\quad U[\mathit{uid_n}].\mathit{status} = \text{“completed”} \implies\\
&\qquad \Box\ \neg(\vee\ U[\mathit{uid_n}].\mathit{status} = \text{“init”}\\
&\qquad\quad \vee\ U[\mathit{uid_n}].\mathit{status} = \text{“requested”}\\
&\qquad\quad \vee\ U[\mathit{uid_n}].\mathit{status} = \text{“activated”}\\
&\qquad\quad \vee\ U[\mathit{uid_n}].\mathit{status} = \text{“denied”}))\\
&\mathit{Safety_Pre_Activated} \triangleq \Box(\forall\, \mathit{uid_n} \in \mathit{UID} :\\
&\quad U[\mathit{uid_n}].\mathit{status} = \text{“activated”} \implies\\
&\qquad \Box\ \neg(\vee\ U[\mathit{uid_n}].\mathit{status} = \text{“init”}\\
&\qquad\quad \vee\ U[\mathit{uid_n}].\mathit{status} = \text{“requested”}\\
&\qquad\quad \vee\ U[\mathit{uid_n}].\mathit{status} = \text{“denied”}))\\
&\mathit{Safety_Pre_Denied} \triangleq \Box(\forall\, \mathit{uid_n} \in \mathit{UID} :\\
&\quad U[\mathit{uid_n}].\mathit{status} = \text{“denied”} \implies\\
&\qquad \Box\ \neg(\vee\ U[\mathit{uid_n}].\mathit{status} = \text{“init”}\\
&\qquad\quad \vee\ U[\mathit{uid_n}].\mathit{status} = \text{“requested”}\\
&\qquad\quad \vee\ U[\mathit{uid_n}].\mathit{status} = \text{“activated”}\\
&\qquad\quad \vee\ U[\mathit{uid_n}].\mathit{status} = \text{“completed”}))\\
&\mathit{Safety_Pre_Requested} \triangleq \Box(\forall\, \mathit{uid_n} \in \mathit{UID} :\\
&\quad U[\mathit{uid_n}].\mathit{status} = \text{“requested”} \implies\\
&\qquad \Box\ \neg(U[\mathit{uid_n}].\mathit{status} = \text{“init”}))\\
&\mathit{Safety_On_Completed} \triangleq \forall\, \mathit{uid_n} \in \mathit{UID} :\\
&\quad U[\mathit{uid_n}].\mathit{status} = \text{“completed”} \implies\\
&\qquad \Box\ \neg(\vee\ U[\mathit{uid_n}].\mathit{status} = \text{“init”}\\
&\qquad\quad \vee\ U[\mathit{uid_n}].\mathit{status} = \text{“requested”}\\
&\qquad\quad \vee\ U[\mathit{uid_n}].\mathit{status} = \text{“activated”}\\
&\qquad\quad \vee\ U[\mathit{uid_n}].\mathit{status} = \text{“terminated”})\\
&\mathit{Safety_On_Activated} \triangleq \forall\, \mathit{uid_n} \in \mathit{UID} :\\
&\quad U[\mathit{uid_n}].\mathit{status} = \text{“activated”} \implies\\
&\qquad \Box\ \neg(\vee\ U[\mathit{uid_n}].\mathit{status} = \text{“init”}\\
&\qquad\quad \vee\ U[\mathit{uid_n}].\mathit{status} = \text{“requested”})\\
&\mathit{Safety_On_Terminated} \triangleq \forall\, \mathit{uid_n} \in \mathit{UID} :\\
&\quad U[\mathit{uid_n}].\mathit{status} = \text{“terminated”} \implies\\
&\qquad \Box\ \neg(\vee\ U[\mathit{uid_n}].\mathit{status} = \text{“init”}\\
&\qquad\quad \vee\ U[\mathit{uid_n}].\mathit{status} = \text{“requested”}\\
&\qquad\quad \vee\ U[\mathit{uid_n}].\mathit{status} = \text{“activated”}\\
&\qquad\quad \vee\ U[\mathit{uid_n}].\mathit{status} = \text{“completed”})\\
&\mathit{Safety_On_Requested} \triangleq \forall\, \mathit{uid_n} \in \mathit{UID} :\\
&\quad U[\mathit{uid_n}].\mathit{status} = \text{“requested”} \implies\\
&\qquad \Box\ \neg(U[\mathit{uid_n}].\mathit{status} = \text{“init”})
\end{aligned}$$

In addition, the definition of *liveness* properties in the UseCON model determine all the valid state transitions regarding any use instance. For example, Figure 3 presents that a *use* instance that has at any given state a *status* 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 3. For the *pre* and *ongoing* authorisation UseCON model these properties can be expressed in TLA$^+$ as follows:

$$\begin{aligned}
&\mathit{Liveness_Pre_Activated} \triangleq\\
&\quad \forall\, \mathit{uid_n} \in \mathit{UID} : U[\mathit{uid_n}].\mathit{status} = \text{“activated”}\\
&\qquad\qquad \rightsquigarrow\ U[\mathit{uid_n}].\mathit{status} = \text{“completed”}\\
&\mathit{Liveness_Pre_Init} \triangleq\\
&\quad \forall\, \mathit{uid_n} \in \mathit{UID} : U[\mathit{uid_n}].\mathit{status} = \text{“init”}\\
&\qquad\qquad \rightsquigarrow\ U[\mathit{uid_n}].\mathit{status} = \text{“requested”}\\
&\mathit{Liveness_Pre_Requested} \triangleq\\
&\quad \forall\, \mathit{uid_n} \in \mathit{UID} : U[\mathit{uid_n}].\mathit{status} = \text{“requested”}\\
&\qquad\qquad \rightsquigarrow (\vee\ U[\mathit{uid_n}].\mathit{status} = \text{“activated”}\\
&\qquad\qquad\quad \vee\ U[\mathit{uid_n}].\mathit{status} = \text{“denied”})\\
&\mathit{Liveness_On_Requested} \triangleq
\end{aligned}$$

$$\forall\, uid\_n \in UID : U[uid\_n].status = \text{“requested”} \rightsquigarrow U[uid\_n].status = \text{“activated”}$$

$$Liveness\_On\_Init \triangleq \forall\, uid\_n \in UID : U[uid\_n].status = \text{“init”} \rightsquigarrow U[uid\_n].status = \text{“requested”}$$

$$Liveness\_On\_Activated \triangleq \forall\, uid\_n \in UID : U[uid\_n].status = \text{“activated”} \rightsquigarrow (\lor U[uid\_n].status = \text{“completed”} \lor U[uid\_n].status = \text{“terminated”})$$

6.2 Usage scenarios policies

Usage scenario 1 is based on the pre-authorisation and usage scenario 2 is based on the ongoing authorisation UseCON model. Therefore, the safety and liveness properties of those models, as defined in the previous subsection, should also be valid in these two scenarios.

Moreover, additional policy-specific properties can be verified in usage scenarios 1 and 2. Specifically, a property that examines: if a subject is viewing an object then it must already exercise the non-disclosure agreement action. Such a property can be verified for usage scenario 1. The aforementioned property is expressed as an invariant in TLA$^+$ as follows:

$$\begin{aligned} Safety1 \triangleq\ & \forall\, u1 \in UID : \\ & (\land\ u1[2] \neq \text{“aid1”} \\ & \land\ U[u1].status = \text{“activated”} \implies \\ & (\exists\, u2 \in UID : \\ & \land\ u2[2] = \text{“aid1”} \\ & \land\ U[u2].status = \text{“completed”} \\ & \land\ SAMESO(u1,\, u2))) \end{aligned}$$

where the *SameSO* expression has the same semantics as in the *Policy1* predicate expression.

Similarly, there are three policy-specific properties that are expressed as TLA$^+$ temporal formulas for usage scenario 2. The first property defines that a premium user will always complete her usage, and the usage will never be terminated by the usage control system, unless completed. The aforementioned property can be expressed as follows:

$$\begin{aligned} Liveness1Scenario2 \triangleq\ & \forall\, uid \in UID : \\ & (uid[1] = \text{“sid2”} \land U[uid].status \neq \text{“completed”}) \rightsquigarrow \\ & (U[uid].status = \text{“completed”}) \end{aligned}$$

A second property could state that the usage of a free licence user can be completed or terminated. The aforementioned property can be expressed as follows:

$$\begin{aligned} Liveness2Scenario2 \triangleq\ & \forall\, uid \in UID : (\\ & uid[1] = \text{“sid1”} \land (U[uid].status \neq \text{“completed”} \\ & \lor\ U[uid].status \neq \text{“terminated”})) \rightsquigarrow \\ & (U[uid].status = \text{“completed”} \lor \\ & U[uid].status = \text{“terminated”}) \end{aligned}$$

An additional third property defines that a free licence user can always complete his usage if he begins that after being completed by a premium user. The aforementioned property can be expressed as follows:

$$\begin{aligned} Liveness3Scenario2 \triangleq\ & \Box(\forall\, uid1,\, uid2 \in UID : \\ & \land\ uid1[3] = uid2[3] \\ & \land\ uid1[1] = \text{“sid1”} \\ & \land\ uid2[1] = \text{“sid2”} \\ & \land\ U[uid1].status = \text{“requested”} \\ & \land\ U[uid2].status = \text{“completed”} \implies \\ & \Diamond\Box(U[uid1].status = \text{“completed”})) \end{aligned}$$

6.3 Faulty policies

In this subsection, we consider the insertion of on a number of faulty policies in the specification. In the following, we revisit the above mentioned policy examples, modify them (insert faults) and investigate the outcome of the verification process.

The predicate used to implement policy 1 (preUseCON model) has been modified as follows:

$$\begin{aligned} MPolicy1(u1) \triangleq\ & \\ & \lor\ \land\ u1[2] = \text{“aid1”} \\ & \lor\ \exists\, u2 \in UID : (\\ & \quad \land\ u1[1] = u2[1] \\ & \quad \land\ u2[2] = \text{“aid1”} \\ & \quad \land\ \ \lor\ U[u2].status = \text{“activated”} \\ & \quad\quad \lor\ U[u2].status = \text{“completed”}) \end{aligned}$$

Compared with the original policy, this predicate guarantees that a usage request from *”sid1”* with any action other than *”aid1”* (e.g. view document) on an object is allowed only if previously a usage of *”sid1”* with action *”aid1”* (i.e. non-disclosure agreement) has been completed. However, the predicate fails in the sense the above mentioned usage requests are not restricted on the same object. Therefore, it is possible that subject *”sid1”* may complete a non-disclosure agreement usage (i.e. action *”aid1”*) on any object (i.e. *”oid1”* object) and then will be able to view any object.

The model run with the following entities, i.e. subject:*”sid1”*, actions:*”aid1”*, *”aid2”* and objects:*”oid1”*, *”oid2”*) and TLC detects a violation on invariant *Safety1*. Additionally, in the error trace field of the Toolbox an

Table 4 Performance evaluation

Authorisation model	Uses	Diameter	States found	Distinct states	Termination	Safety	Liveness	Policies
Pre	8	25	1870480	389919	00:00:22	–	–	–
	8	25	1870220	389889	–	00:04:39	–	–
	8	25	1870830	389969	–	–	00:01:18	–
	8	25	393217	65536	–	–	–	00:00:22
	10	31	58575316	9763305	00:00:55	–	–	–
	10	31	58576741	9763492	–	02:52:26	–	–
	10	31	58577250	9763553	–	–	00:35:08	–
	10	31	7864321	1048576	–	–	–	00:00:25
Ongoing	8	25	2499671	390570	00:00:24	–	–	–
	8	25	2499449	390538	–	00:05:04	–	–
	8	25	2499451	390538	–	–	00:01:15	–
	8	25	793153	104976	–	–	–	00:00:57
	10	31	78123847	9765474	00:00:55	–	–	–
	10	31	78122850	9765362	–	03:24:40	–	–
	10	31	78122882	9765372	–	–	00:41:21	–
	10	31	17845921	1889568	–	–	–	00:26:56

* Time in HH:MM:SS

example behaviour that violates *Safety1* is provided. Variable U holds in a state the following values:

```
( <<"sid1", "aid1", "oid2">> :>
            [status |-> "completed"] @@
 <<"sid1", "aid2", "oid2">> :>
            [status |-> "init"] @@
 <<"sid1", "aid1", "oid1">> :>
            [status |-> "init"] @@
  <<"sid1", "aid2", "oid1">> :>
            [status |-> "requested"] )
```

In the next state of the above behaviour, the last usage (i.e. having UID *"sid1, aid2, oid1"*) change its status attribute value from *"requested"* to *"activated"*. This is caused since *"sid1"* has performed *"aid1"* on *"oid1"* (i.e. having UID *"sid1, aid1, oid2"*). However, this behaviour violates our policy since it is required for every subject to complete a non-disclosure agreement (compulsory) on the same object.

The predicate used to implement policy 2 (ongoing authorisation model) has been altered in two versions. The specification of the first follows:

$$
\begin{aligned}
MPolicy2\_1(u1) \triangleq\ & \\
& \lor\ u1[1] = \text{“sid2”} \\
& \lor\ \land\ u1[1] = \text{“sid1”} \\
& \quad \land \neg(\exists\, u2\ \in \mathit{UID} : \\
& \qquad \land\ u2[1] = \text{“sid2”} \\
& \qquad \land\ U[u2].\mathit{status} = \text{“activated”})
\end{aligned}
$$

Compared with the original predicate, this one terminates any usage requested from subject *"sid1"* if a usage from subject *"sid2"* is activated on any object. Therefore, a usage of *"sid1"* will be terminated even if *"sid2"* exercises a usage on a different object.

The previous alteration of the predicate violates temporal formula *Liveness3Scenario2*. That formula states that *"sid2"* completes the usage of a specific object, when *"sid1"* requests to use that object in the future, that usage should be allowed for completion (i.e. reach *"complete"* state). This is because *"sid2"* will never ask the same usage again.

The model run with the following entities, i.e. subject:*"sid1, sid2"*, actions:*"aid1"* and objects: *"oid1, oid2"*) and TLC detects a violation of the *Liveness3Scenario2* temporal formula. Variable U holds in a state the following values:

```
( <<"sid2", "aid1", "oid2">> :>
                [status |-> "activated"] @@
  <<"sid2", "aid1", "oid1">> :>
                [status |-> "completed"] @@
  <<"sid1", "aid1", "oid1">> :>
                [status |-> "activated"] )
```

In the next state of the above behaviour, the status attribute of usage with UID *"sid1, aid1, oid1"* changes from *"activated"* to *"terminated"*. That happened since usage UID *"sid2, aid1, oid2"* is *"activated"* despite the fact that they are both operating on different objects. However, this violates the temporal formula, because *"sid2"* has already completed usage on object *"oid1"* (i.e. UID *"sid2, aid1, oid1"*). A predicate that implements an additional alteration of policy2, follows:

$$
\begin{aligned}
MPolicy2\_2(u1) \triangleq\ & \land\ u1[1] = \text{“sid1”} \\
& \land \neg(\exists\, u2\ \in \mathit{UID} : \\
& \quad \land\ u2[1] = \text{“sid2”} \\
& \quad \land\ u1[3] = u2[3] \\
& \quad \land\ U[u2].\mathit{status} = \text{“activated”})
\end{aligned}
$$

Compared with the original predicate this one always terminates usages requested by subject *"sid2"*. The aforementioned predicate violates the *Liveness1Scenario2* temporal formula, which states that every usage requested by *"sid2"* will always be completed. Indeed, running the model with the following entities, i.e. subject:*"sid1, sid2"*, actions:*"aid1"* and objects: *"oid1"*) TLC detects a violation. A counterexample behaviour is shown bellow in which the status variable of a *"sid2"* usage becomes terminated.

```
( <<"sid2", "aid1", "oid1">> :>
                [status |-> "terminated"] )
```

6.4 Performance Evaluation

The verification of the examined model was performed using TLA$^+$ Tool version 1.7.0 and running TLC version 2.15 on Ubuntu 18.04. The hardware specifications of the host system are: 2x Intel(R) Xeon(R) Gold 6130 Processors; 128 GB DDR4 2666 of physical memory; and 512GB SATA3 SSD. The verification was performed using the *'most'* option on the *'How to run'* field on *ToolBox*.

A first set of results were collected by verifying termination states for the UseCON model. Specifically, in a pre-authorisation UseCON model, we consider all the *uses* of the model to be requested, and therefore, to be either in the *'activated'* state or in the *'denied'* state. 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 6.1. The verification was finished without raising any errors[5], but this applies only for the verification of ≤ 10 uses for both the *pre* and *ongoing* authorisation models. The input data used to verify each of the authorisation models are:

- ***Pre* model (8 uses)**: 2 subjects {"sid1", "sid2"}, 2 objects {"oid1", "oid2"}, 2 actions {"aid1", "aid2"}.
- ***Pre* model (10 uses)**: 5 subjects {"sid1", "sid2", "sid3", "sid4", "sid5"}, 1 object {"oid1"}, 2 actions {"aid1", "aid2"}.
- ***Ongoing* (8 uses)**: 2 subjects {"sid1", "sid2"}, 4 objects {"oid1", "oid2", "oid3", "oid4"}, 1 action {"aid1"}.
- ***Ongoing* (10 uses)**: 2 subjects {"sid1", "sid2"}, 5 objects {"oid1", "oid2", "oid3", "oid4", "oid5"}, 1 action {"aid1"}.

Although higher numbers of *uses* were considered, the verification process had to be interrupted due to the excessive amount of time required for its completion. When considered 12 *uses* the verification exceeded the 14 hours without being finished. Based on our previous experience with TLC and large models, it might be the case that TLC would not terminate the verification for 12 uses and generate an *'java.lang.OutOfMemoryError: GC overhead limit exceeded'* error message – an indication of memory exhaustion, i.e. the Java virtual machine spent an excessive amount of time performing Garbage Collection and was able to reclaim very little heap space.

Even though we were not able to get verification results for more than 10 uses, a system instance could have more uses than the analysed ones. Thus, one could argue that our analyses offer no *a priori* guarantees for a system with more than 10 uses. However, an observation known as the *'small scope hypothesis'*[13] states that analysing small system instances suffices in practice since a high proportion of bugs can be found by verifying a system for all inputs within some (usually small) scope. A plethora of empirical studies [1,23,30] support this hypothesis. For example, Yuan et al. [30] analysed production failures in distributed data intensive systems and showed that simple testing can prevent most critical failures. In particular, the aforementioned study showed that out of the 198 bug reports that were analysed for several distributed systems, 98% of those bugs could be triggered in a verification setting of three or fewer processes. The small scope hypothesis has been well supported by empirical analysis and has served as the basis of lightweight modeling and analysis [13,31]. Based on the small scope hypothesis, we believe that our analyses for up to 10 uses suffices for identifying a high proportion of errors. To support this claim, we introduced errors in our model and checked whether we can successfully catch these within the 10 uses scope. Our methodology is described in detail in Section 6.3 (faulty policies), where we showed that our scope was sufficient for identifying all faults that were introduced.

The collected verification results (see Table 4) 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 such that $s \mapsto t$ can be a step in a behaviour. Specifically, TLC substitutes the values assigned to variables by state s

[5] Termination is successful if, for all behaviours, the specification ends.

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. Every state t, found by the former procedure, is added to set G if it does not already exist. The previous two actions are repeated until no new states can be added in G. Therefore, the verification results produced by TLC incorporate information about: 1) the *Diameter*, which describes the number of states in the longest path of G in which no state appears twice; 2) the *States found*, which describe the number of examined states; 3) the *Distinct States*, which describe the number of examined distinct states. The verification results produced by TLC for the *pre* and *ongoing* UseCON models are presented in Table 4. An additional column presents the actual running time of the TLC model checker in hh:mm:ss format.

Furthermore, looking closer at the performance results in Table 4, we elaborate on a number of observations that result in strengthening our confidence with regard to the correctness of the UseCON specification. Firstly, the evaluation of the ongoing-authorisation model creates a larger number of states compared with the pre-authorisation model. This is explained by the fact that the ongoing model contains a larger number of actions. Another useful observation is that the execution of a policy-neutral UseCON model (i.e. pre- or ongoing) has significantly more states compared with the UseCON model that implements a specific policy. This difference is explained by the fact that a policy introduces a predicate, which restricts the next steps of a *preEvaluate* or *onEvaluate* action. Finally, there is a significant difference in the verification time between the safety and the liveness properties. Specifically, the liveness properties are verified in less time compared with the safety ones. This difference is explained by the fact that safety properties are using the *'always'* operator imposing the TLC model checker to verify them in every state. While the liveness properties are verified only on the states where the first part of the operator is true due to the implication operator.

7 Conclusion

In this paper, we used TLA$^+$ tools to analyse and verify UseCON's use management processes. Specifically, the trace of the termination states verified that the defined specifications of the system operate correctly and a set of safety and liveness properties ensured the correctness of UseCON's transition systems and example policies. An advantage of using model checking techniques for the verification of usage control models is the provision of formal guarantees with regard to the correctness of the model, without requiring an implementation of it. We also demonstrated how policy rules can be verified in usage control models – specifically in UseCON – assuming both a *pre* and *ongoing* authorisation model. These are of interest since a *use* request might lead to a policy violation in 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. The insertion of faulty policies is also considered to validate the correctness of the specifications. 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 ≤ 10 uses for both the *pre* and *ongoing* authorisation models. Considering the small scope hypothesis, we argue that analysing small system instances suffices in practice to find a high proportion of bugs by verifying a system for all inputs within some (usually small) scope. However, if stronger guarantees are required, other options and tools may have to be considered such as theorem provers. We anticipate our work to shed light in the application of formal verification techniques in the domain of usage/access control, which could be beneficial when considering their application in complex systems (e.g. the Cloud [11]).

Acknowledgements

We would like to thank the anonymous reviewers for their helpful feedback that resulted in improving the overall quality of this paper.

References

1. Andoni, A., Daniliuc, D., Khurshid, S., Marinov, D.: Evaluating the "small scope hypothesis". In: In Popl, vol. 2. Citeseer (2003)
2. Backes, J., Bolignano, P., Cook, B., Dodge, C., Gacek, A., Luckow, K., Rungta, N., Tkachuk, O., Varming, C.: Semantic-based automated reasoning for aws access policies using smt. In: 2018 Formal Methods in Computer Aided Design (FMCAD), pp. 1–9. IEEE (2018)
3. Black, P.E., Okun, V., Yesha, Y.: Mutation of model checker specifications for test generation and evaluation. In: Mutation testing for the new century, pp. 14–20. Springer (2001)
4. Cau, A., Moszkowski, B., Zedan, H.: Interval temporal logic. URL: http://www.cms.dmu.ac.uk/~cau/itlhomepage/itlhomepage. html (2006)
5. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: Nusmv 2: An opensource tool for symbolic model checking. In: International Conference on Computer Aided Verification, pp. 359–364. Springer (2002)

6. Gouglidis, A., Grompanopoulos, C., Mavridou, A.: Formal verification of usage control models: A case study of usecon using tla+. arXiv preprint arXiv:1806.09848 (2018)
7. Gouglidis, A., Mavridis, I., Hu, V.C.: Security policy verification for multi-domains in cloud systems. Int. J. Inf. Sec. **13**(2), 97–111 (2014). DOI 10.1007/s10207-013-0205-x
8. Grompanopoulos, C., Gouglidis, A.: UseCON specification. https://github.com/agouglidis/UseCON-TLA_PLUS (2020)
9. Grompanopoulos, C., Gouglidis, A., Mavridis, I.: A use-based approach for enhancing UCON. In: Security and Trust Management - 8th International Workshop, STM 2012, Pisa, Italy, September 13-14, 2012, Revised Selected Papers, pp. 81–96 (2012). DOI 10.1007/978-3-642-38004-4_6
10. Holzmann, G.J.: The SPIN model checker: Primer and reference manual, vol. 1003. Addison-Wesley Reading (2004)
11. Hu, V., Iorga, M., Bao, W., Li, A., Li, Q., Gouglidis, A.: General access control guidance for cloud systems. Tech. rep., National Institute of Standards and Technology (2020)
12. Hu, V.C., Kuhn, R., Yaga, D.: Verification and test methods for access control policies/models. NIST Special Publication **800-192** (2017). DOI 10.6028/NIST.SP.800-192
13. Jackson, D.: Software Abstractions: logic, language, and analysis. MIT press (2012)
14. Janicke, H., Cau, A., Zedan, H.: A note on the formalisation of UCON. In: Proceedings of the 12th ACM symposium on Access control models and technologies, SACMAT '07, pp. 163–168. ACM, New York, NY, USA (2007)
15. Lamport, L.: The temporal logic of actions. ACM Transactions on Programming Languages and Systems (TOPLAS) **16**(3), 872–923 (1994)
16. Lamport, L.: Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley (2002)
17. Lazouski, A., Martinelli, F., Mori, P.: Usage control in computer security: A survey. Computer Science Review **4**(2), 81–99 (2010). DOI 10.1016/j.cosrev.2010.02.002
18. Lu, J., Li, R., Hu, J., Xu, D.: Static enforcement of static separation-of-duty policies in usage control authorization models. IEICE Transactions **95-B**(5), 1508–1518 (2012)
19. Macedo, N., Cunha, A.: Alloy meets tla+: An exploratory study (2016)
20. Martinelli, F., Mori, P.: On usage control for grid systems. Future Gener. Comput. Syst. **26**(7), 1032–1042 (2010). DOI 10.1016/j.future.2009.12.005
21. Mavridou, A., Stachtiari, E., Bliudze, S., Ivanov, A., Katsaros, P., Sifakis, J.: Architecture-based design: A satellite on-board software case study. In: Formal Aspects of Component Software - 13th International Conference, FACS 2016, Besançon, France, October 19-21, 2016, Revised Selected Papers, pp. 260–279 (2016). DOI 10.1007/978-3-319-57666-4_16
22. Newcombe, C., Rath, T., Zhang, F., Munteanu, B., Brooker, M., Deardeuff, M.: Use of formal methods at amazon web services. https://lamport.azurewebsites.net/tla/formal-methods-amazon.pdf (2014)
23. Oetsch, J., Prischink, M., Pührer, J., Schwengerer, M., Tompits, H.: On the small-scope hypothesis for testing answer-set programs. In: Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning (2012)
24. Park, J., Sandhu, R.S.: The UCON$_{ABC}$ usage control model. ACM Trans. Inf. Syst. Secur. **7**(1), 128–174 (2004). DOI 10.1145/984334.984339
25. Pretschner, A., Ruesch, J., Schaefer, C., Walter, T.: Formal analyses of usage control policies. In: Availability, Reliability and Security, 2009. ARES '09. International Conference on, pp. 98–105 (2009). DOI 10.1109/ARES.2009.100
26. Rajkumar, P., Ghosh, S., Dasgupta, P.: Concurrent usage control implementation verification using the spin model checker. In: N. Meghanathan, S. Boumerdassi, N. Chaki, D. Nagamalai (eds.) Recent Trends in Network Security and Applications, *Communications in Computer and Information Science*, vol. 89, pp. 214–223. Springer Berlin Heidelberg (2010). DOI 10.1007/978-3-642-14478-3_22
27. Ranise, S., Armando, A.: On the automated analysis of safety in usage control: A new decidability result. In: L. Xu, E. Bertino, Y. Mu (eds.) Network and System Security, *Lecture Notes in Computer Science*, vol. 7645, pp. 15–28. Springer Berlin Heidelberg (2012). DOI 10.1007/978-3-642-34601-9_2
28. Said, N.B., Abdellatif, T., Bensalem, S., Bozga, M.: Model-driven information flow security for component-based systems. In: From Programs to Systems. The Systems perspective in Computing, pp. 1–20. Springer (2014)
29. Samarati, P., de Vimercati, S.C.: Access control: Policies, models, and mechanisms. In: International School on Foundations of Security Analysis and Design, pp. 137–196. Springer (2000)
30. Yuan, D., Luo, Y., Zhuang, X., Rodrigues, G.R., Zhao, X., Zhang, Y., Jain, P.U., Stumm, M.: Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems. In: 11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14), pp. 249–265 (2014)
31. Zave, P.: Using lightweight modeling to understand chord. ACM SIGCOMM Computer Communication Review **42**(2), 49–57 (2012)
32. Zhang, X., Nakae, M., Covington, M.J., Sandhu, R.S.: Toward a usage-based security framework for collaborative computing systems. ACM Trans. Inf. Syst. Secur. **11**(1), 3:1–3:36 (2008). DOI 10.1145/1330295.1330298
33. Zhang, X., Parisi-Presicce, F., Sandhu, R., Park, J.: Formal model and policy specification of usage control. ACM Trans. Inf. Syst. Secur. **8**, 351–387 (2005)
34. Zhang, X., Park, J., Parisi-Presicce, F., Sandhu, R.S.: A logical specification for usage control. In: 9th ACM Symposium on Access Control Models and Technologies, SACMAT 2004, Yorktown Heights, New York, USA, June 2-4, 2004, Proceedings, pp. 1–10 (2004). DOI 10.1145/990036.990038
35. Zhang, X., Sandhu, R., Parisi-Presicce, F.: Safety analysis of usage control authorization models. In: Proceedings of the 2006 ACM Symposium on Information, computer and communications security, ASIACCS '06, pp. 243–254. ACM, New York, NY, USA (2006)
36. Zhang, X., Sandhu, R.S., Parisi-Presicce, F.: Formal model and analysis of usage control. George Mason University (2006)