Refinement-based Design of a Group-centric Secure Information Sharing Model

Wanying Zhao, Jianwei Niu, William H. Winsborough
The University of Texas at San Antonio
One UTSA Circle, San Antonio, Texas, USA 78249
{wzhao, niu}@cs.utsa.edu, wwinsborough@acm.org

ABSTRACT

This paper presents a formal, state machine-based specification (*stateful specification*) of a group-centric secure information sharing (g-SIS) model. The stateful specification given here is a refinement of a prior specification that is given in first-order linear temporal logic (FOTL). Such FOTL specification defines authorization based solely on group operations, but gives little guidance regarding implementation. The current specification is the result of a second step in a multi-step design process that separates concerns and provides multiple opportunities to detect unintended policy characteristics. We show that our stateful specification is consistent with the prior FOTL specification by using a combination of model-checking and manual techniques.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection – Access controls

General Terms

Security

Keywords

Access Control, Formal Specification, Formal Verification, System Refinement, Secure Information Sharing

1. INTRODUCTION

As a society, we enjoy a great opportunity to make information much more highly available than it has been historically. This can facilitate emergency preparedness by supporting rapid dissemination of information concerning immanent natural or man-made threats or response requirements. Information sharing can also support more general publish/subscribe relationships, as well as enable collaborations that otherwise might require physical proximity. However, all these applications require that some degree of access control (AC) be enforced over shared information. This requirement has sometimes been dubbed, "share, but protect."

Information can be shared between individuals, as with limited access blogs or forums, or between representatives of various kinds of organizations. In the latter case, the metaphor of a secure meeting room may be apt. Participants may be working toward some specific goal, such as designing a new product, arriving at an acquisition or merger agreement, or conducting a joint military operation.

Group-centric secure information sharing (g-SIS) [8] is a recently introduced AC model that brings users and information together in a group to provide for access. It is a simple, yet flexible sharing framework suited for use in highly dynamic environments. The g-SIS framework defines authorizations in terms of action histories involving *group operations* (namely, user join, user leave, object add, and object remove). The temporal order in which those operations occur determines whether a user is authorized to access an object. One particular g-SIS policy, presented by Krishnan *et al.*, is given by what is called the π *specification* [9], and is expressed in many-sorted, first-order linear temporal logic (FOTL)[1]. However, the FOTL specification is highly abstract. It would be difficult for security systems' stakeholders to comprehend and implement a AC system directly from it.

To alleviate users efforts, we develop a separate specification of authorization policy, which is given in terms of attributes and data structures that record appropriate information to enable efficient authorization decisions. The π specification was the first step in the multi-step design process. Our current specification is the result of a second step in the multi-step design process that bridges the gap between abstract FOTL specification and enforcement model. The current specification introduces and maintains data structures that summarize sufficient information about the history of group operations to enable authorization decisions to be made directly from the data structure values. This in and of itself is a significant step toward generating a specification from which a developer could implement. However, it is not the last. Later steps, not undertaken here, will address distribution, communication, latency issues, *etc.*

The specification we present here is given by a state machine (SM). The notion of an SM differs from that of a conventional finite state machine only in that the number of

CODASPY'12, February 7–9, 2012, San Antonio, Texas, USA.

[1]FOTL resembles the more familiar propositional linear temporal logic (PLTL) that is used in model checkers. The differences are analogous to those between many-sorted first-order predicate calculus and propositional logic. [2]

(reachable) states is potentially countably infinite. An SM is intuitive and expressive, yet programming-language neutral. To provide a shorthand by which to contrast these two approaches to specification, we call the FOTL specification a *stateless specification* and an SM-based specification a *stateful specification*. Our stateful specification is a *refinement* of the stateless π specification in the sense that our specification is consistent with the π specification, but provides additional detail.

We develop a combination of methodologies to formally verify that a stateful specification and its corresponding stateless specification are consistent (refinement equivalent). We use a model checking tool, NuSMV [3], to verify if a state machine specification consisting of a single user and a single object within a group satisfies the stateless specification expressed in temporal logic. As the g-SIS system can have unbounded numbers of objects, users, and groups, we develop a manual reasoning technique to generalize automated model checking results to infinite universes.

Our contributions are as follows. First, we present our stateful specification and its data structures. Our specification also shows how to handle requests to perform group operations, and how to filter out requests to perform illegal operations. We also provide invariants that relate the values of the data structures in the SM to the histories under which the data structure have those values. Second, we show a soundness result, which says that any sequence of group operations accepted by the stateful specification causes the SM to arrive at the same authorization decisions as are arrived at by the π specification. Thirdly, we show a completeness result. Specifically, we show that every sequence of group operations that satisfies the π specification is accepted by our SM. (We already know from soundness that the authorization decisions agree.)

The rest of the paper is organized as follows. Section 2 summarizes the key elements of the prior g-SIS policy given by the stateless π specification. Section 3 presents our stateful specification of that policy. Section 4 demonstrates that our stateful specification is consistent with the stateless π specification in the sense outlined above. Section 5 discusses how to handle requests to perform group operations that are illegal. Section 6 discusses factors that limit scalability of our design and on-going work seeking to mitigate those factors. Section 7 discusses related work and section 8 presents our conclusions and discusses our future work.

2. BACKGROUND ON G-SIS AND THE π SPECIFICATION

As discussed in the introduction, g-SIS supports four basic group operations—join, leave, add, and remove—and the π specification considers *strict* and *liberal* variants of each. Let us summarize the intuition of each of these eight operations. When a user joins a group by Strict Join (SJ), only objects added to the group after join time are accessible. When Liberal Join (LJ) is used, the user can additionally access objects added previously by using Liberal Add (LA). When a user leaves a group by Strict Leave (SL), the user loses all access previously granted by membership in the group. When Liberal Leave (LL) is used, the user retains access to those objects granted by his membership immediately prior to leave time. If an object is added to a group by Strict Add (SA), only users who joined the group prior to add time can access the object; users joining later do not receive access. When Liberal Add (LA) is used, users joining later via Liberal Join also receive access. If an object is removed from a group by Strict Remove (SR), all users lose access to it. When Liberal Remove (LR) is used, users who had access to the object at remove time retain access. Since π specification supports read-only operation, the "access" in following sections of this paper is read access.

2.1 g-SIS Language

The π specification is expressed in a form of linear temporal logic, specifically many-sorted FOTL, determining whether a user is permitted to access an object in a given group based on the history of group operations at each point in time. The temporal operators of FOTL used in this paper and their intuitive meanings are summarized in Table 1.

As mentioned above, the π system supports the eight group operations that are represented by *action predicates* in $\mathcal{A} = \{\mathrm{SJ}, \mathrm{LJ}, \mathrm{SL}, \mathrm{LL}, \mathrm{SA}, \mathrm{LA}, \mathrm{SR}, \mathrm{LR}\}$. Authorization is represented by the *authorization predicate*, Authz.

Sorts enable us to distinguish users from objects, *etc.* They resemble very simple types. For users, objects, and groups, we denote the corresponding sorts by U, O, and G. We also use, respectively, the variables u, o, and g, possibly with subscripts. Sorts are assigned when a variable is quantified: $\forall u : \mathrm{U}.\phi$. The semantic values over which a variable ranges depend on the variable's sort and are drawn from a set that is called the *carrier* of that sort. For the sorts identified above we denote the corresponding carriers by $\mathcal{U}$, $\mathcal{O}$, and $\mathcal{G}$, respectively. We use the following metavariables to range over individuals: $\mathbf{u} \in \mathcal{U}$, $\mathbf{o} \in \mathcal{O}$, and $\mathbf{g} \in \mathcal{G}$. We denote the collection of carriers by $\mathcal{C} = \langle \mathcal{G}, \mathcal{U}, \mathcal{O} \rangle$. In general, these carriers may be finite or countably infinite.

We use $\Omega_{\mathcal{C}}$ to denote the set of finite interpretations of action and authorization predicates over the carriers in $\mathcal{C}$. An *interpretation*, ι, is a function mapping each predicate in the language to a relation over the appropriate carriers. Although they need not be so, the relations in the interpretations we use are finite, reflecting the fact that at any point in time, only a finite number of users, objects and groups have been introduced to the system. In our context an interpretation for π maps predicates to relations of the following types: $[\![\mathrm{SJ}]\!]\iota, [\![\mathrm{LJ}]\!]\iota, [\![\mathrm{SL}]\!]\iota, [\![\mathrm{LL}]\!]\iota \subset \mathcal{U} \times \mathcal{G}$. $[\![\mathrm{SA}]\!]\iota, [\![\mathrm{LA}]\!]\iota, [\![\mathrm{SR}]\!]\iota, [\![\mathrm{LR}]\!]\iota \subset \mathcal{O} \times \mathcal{G}$, and $[\![\mathrm{Authz}]\!]\iota \subset \mathcal{U} \times \mathcal{O} \times \mathcal{G}$. Each $\kappa \in \Omega_{\mathcal{C}}^{\omega}$ is an infinite sequence of interpretations called a *trace*. For each $i \in \mathbb{N}$, we write $[\![\mathrm{SJ}]\!]\kappa_i \subseteq \mathcal{U} \times \mathcal{G}$ to denote the relation associated with SJ by κ_i. If $\langle \mathbf{u}, \mathbf{g} \rangle \in [\![\mathrm{SJ}]\!]\kappa_i$, this means user $\mathbf{u}$ does a Strict Join of group $\mathbf{g}$ at position i. Similarly, $\langle \mathbf{u}, \mathbf{o}, \mathbf{g} \rangle \in [\![\mathrm{Authz}]\!]\kappa_i$ indicates that user $\mathbf{u}$ is authorized to access object $\mathbf{o}$ in group $\mathbf{g}$ at position i.

Let us now outline what it means for a trace to satisfy an FOTL formula. Here we illustrate the construction just below by giving the most important semantic rules. To handle variables, we need the notion of an *environment* η, which assigns to each variable an element of the carrier corresponding to the variable's sort. The statement that, at index i, κ satisfies ϕ under environment η, written $\kappa, i, \eta \models \phi$, is defined inductively on the structure of ϕ. For example, $\kappa, i, \eta \models \mathrm{SJ}(u, g)$ holds if $\langle \eta(u), \eta(g) \rangle \in [\![\mathrm{SJ}]\!]\kappa_i$. For temporal operators "since": $\kappa, i, \eta \models \phi_1 \mathcal{S} \phi_2$ holds if there exists $k \in \mathbb{N}$ such that $0 \leq k \leq i$ and $\kappa, k, \eta \models \phi_2$, and for all $j \in \mathbb{N}$ such that $k < j \leq i$, $\kappa, j, \eta \models \phi_1$. Other temporal operators can be found in [2]. For example of an-

Table 1: Intuitive summary of temporal operators

Operator	Read as	Explanation
$\ominus$	Previous	($\ominus$ p) means that formula p held in the previous state.
$\diamondsuit$	Once	($\diamondsuit$ p) means that formula p held at least once in the past.
$\mathcal{S}$	Since	(p $\mathcal{S}$ q) means that q happened in the past and p held continuously from the state following the last occurrence of q to the present.
$\boxminus$	Historically	$\boxminus$ p means p holds at all states preceding (and including) the current state.
$\Box$	Henceforth	$\Box$ p means p will continuously hold in all future states starting from the current state.

other rule, $\kappa, i, \eta \models \exists u\!:\!U.\phi$ if there exists $\mathbf{u} \in \mathcal{U}$ such that $\kappa, i, \eta[u \mapsto \mathbf{u}] \models \phi$. For a formula ϕ, ϕ is satisfied by a trace κ, written $\kappa \models \phi$, if $\kappa, 0, \eta \models \phi$ for all environments η. The complete semantics of the g-SIS language can be found in [8]

2.2 The π Specification

The π specification defines three *well formedness* requirements that every trace must satisfy at every state. Requirement τ_0 states that no user can join and leave a given group at the same time and no object can be added to and removed from the same group at the same time. Requirement τ_1 states that no user can both strictly and liberally join or leave a given group at the same time and similarly for objects. Requirement τ_2 says that no user can join a group that he currently belongs to and that he cannot leave a group unless he currently belongs to it. It also expresses the analogous requirements for objects.

The π specification determines at each point in time whether a user is permitted access to an object via a given group, based on the history of group operations involving the user and object with respect to that group. The π specification is given as follows:

$$\pi : \forall u : U.\forall o : O.\forall g : G.$$
$$\Box((\text{Authz}(u,o,g) \leftrightarrow \lambda_1(u,o,g) \vee \lambda_2(u,o,g)) \wedge \bigwedge_{0\le j\le 2} \tau_j(u,o,g))$$

in which $\lambda_1(u,o,g)$ and $\lambda_2(u,o,g)$ defined by

$$\lambda_1(u,o,g) :$$
$$(\neg\text{SL}(u,g) \wedge \neg\text{SR}(o,g))\,\mathcal{S}\,[(\text{SA}(o,g) \vee \text{LA}(o,g)) \wedge ((\neg\text{LL}(u,g) \wedge \neg\text{SL}(u,g))\,\mathcal{S}\,(\text{SJ}(u,g) \vee \text{LJ}(u,g)))]$$
$$\lambda_2(u,o,g) :$$
$$(\neg\text{SL}(u,g) \wedge \neg\text{SR}(o,g))\,\mathcal{S}\,[\text{LJ}(u,g) \wedge ((\neg\text{SR}(o,g) \wedge \neg\text{LR}(o,g)\,\mathcal{S}\,\text{LA}(o,g)))]$$

The second subformula (involving the τ's) says that traces satisfying the π specification must be well formed. The two subformulas, λ_1 and λ_2 correspond to two cases in which a user is authorized for access to an object. In case (a), handled by λ_1, the user joins the group before or at the same time as the object is added; in case (b), handled by λ_2, the object is added before the user joins. In case (a), authorization does not depend on whether the join and add are strict or liberal because the user is already a member when the object is added. The right-hand subformula of λ_1 (in square brackets) formalizes the requirement that the user is in the group when the object is added. The left-hand subformula permits authorization to continue after liberal leave or liberal remove. Case (b) allows for authorization when both the join and the add are liberal. The right-hand subformula of λ_2 (in square brackets) formalizes the requirement that the object is in the group when the user joins. As it does in λ_1, the left-hand subformula permits authorization to continue after liberal leave or liberal remove. Note that in any trace κ such that $\kappa \models \pi$, the interpretation of Authz is uniquely determined at each κ_i by the history of interpretations of the action predicates leading up to that point in the trace. In particular, $\mathbf{u}$ is authorized for access to $\mathbf{o}$ via $\mathbf{g}$ at step $i \in \mathbb{N}$ just in case $\langle \mathbf{u}, \mathbf{o}, \mathbf{g} \rangle \in [\![\text{Authz}]\!]\kappa_i$.

3. STATEFUL G-SIS SPECIFICATION

This section begins by introducing the construction of the state machine designed for stateful specification, followed by defining the data structures, and the invariants that identify historical conditions under which each value of each data structure is assumed. We then present the general formula that defines authorization in the stateful g-SIS specification based on the data structure values. We then show the transition relations for each data structure that would maintain the invariants correctly as group operations were performed. To illustrate the stateful specification, we use a case study to show data structure value updates and authorization decisions in the end of this section.

3.1 Our Stateful Specification and its Relationship to the Stateless Specification

Our stateful specification is a deterministic SM that we call $M_\pi^{\mathcal{C}}$, which produces authorization decisions identical to those made by traces that satisfy the π specification. The notion of a state machine we use is standard. It corresponds to the familiar notion of a finite state machine, with the difference being that, depending on the size of the carriers, the state space is not necessarily finite. A state machine is given by $M = \langle Q, q^0, \Sigma, \delta \rangle$. Q is a (possibly infinite) set of machine states [2],$q^0 \in Q$ is a distinguished start state, Σ is an alphabet, and $\delta \subset Q \times Q \times \Sigma$ is a deterministic transition relation. Given any sequence $\sigma \in \Sigma^\omega$, we define the *run of M induced by σ*, if it exists, to be the unique sequence $\rho \in Q^\omega$ of machine states defined inductively as follows. The initial state is $\rho_0 = q^0$ and for each $i \in \mathbb{N}$, $\delta(\rho_i, \rho_{i+1}, \sigma_{i+1})$. (This kind of SM ignores σ_0.) If there comes a point in σ at which no transition is defined, σ is *rejected* by M; otherwise it is accepted and the induced run ρ is well defined.

Our stateful specification $M_\pi^{\mathcal{C}}$ has the form $M_\pi^{\mathcal{C}} = \langle Q_\pi, q_\pi^0, \Sigma_{\mathcal{C}}, \delta_\pi \rangle$. The alphabet $\Sigma_{\mathcal{C}}$ is the set of interpretations of the action predicates in $\mathcal{A}$ (see section 2.1). Each state in Q_π is an assignment of values to each of the data structures introduced below in section 3.2. The initial state q_π^0 and the transition relation δ_π are also presented in section 3.2.

Note that the only difference between $\Omega_{\mathcal{C}}$ and $\Sigma_{\mathcal{C}}$ is that elements of the former interpret the authorization predicate

[2]In section 4 we will show that we verify the model with countably infinite carrier by using combination of model checking technique and manual proofs.

Authz, as well as the predicates in $\mathcal{A}$. We call each $\sigma \in \Sigma_{\mathcal{C}}^{\omega}$ an *action sequence*. An action sequence σ can be obtained from a trace κ by projecting each element of the trace, κ_i, onto the action predicates. In this case we write $\sigma = \kappa\restriction_{\mathcal{A}}$. Because τ_0, τ_1, and τ_2 do not use Authz, $\kappa\restriction_{\mathcal{A}}$ is well formed just in case κ is well formed. As noted at the end of the previous section, for any $\kappa \in \Omega_{\mathcal{C}}^{\omega}$ such that $\kappa \models \pi$, the interpretation of Authz at κ_i is uniquely determined by the prefix of $\sigma = \kappa\restriction_{\mathcal{A}}$ of length $i+1$. Put another way, for each well formed action sequence σ, there is a unique κ such that $\sigma = \kappa\restriction_{\mathcal{A}}$ and $\kappa \models \pi$. The run of $M_{\pi}^{\mathcal{C}}$, ρ, induced by a given $\sigma \in \Sigma_{\mathcal{C}}^{\omega}$ maintains the data structures contained in each ρ_i so that their values can be used to make authorization decisions without the need to consult σ. In section 4 we will show that (1) given any action sequence σ, there is a run of $M_{\pi}^{\mathcal{C}}$, ρ, induced by σ if and only if σ is well formed and (2) ρ generates exactly the same authorization decisions as does the unique κ such that $\sigma = \kappa\restriction_{\mathcal{A}}$.

3.2 Data Structures and Invariants

This section presents the data structures used by $M_{\pi}^{\mathcal{C}}$. The design process we used was organized by simultaneously specifying invariants that characterize properties of the history up to any given point in σ and relating those properties to values assumed by the data structures at that point. We knew we needed certain combinations of data structure values to indicate that either λ_1 or λ_2 was satisfied. We developed natural-seeming data structures for representing when this is the case, as well as corresponding invariants that together entailed λ_1 or λ_2. Table 2 summarizes the data structures of $M_{\pi}^{\mathcal{C}}$, each of the values they can take on, and the temporal invariants under which each of those values is assumed.

Boolean-valued data structures CurrUserMem($\mathbf{u}, \mathbf{g}$) and CurrObjMem($\mathbf{o}, \mathbf{g}$) represent, respectively, whether user $\mathbf{u}$ and object $\mathbf{o}$ are current members of group $\mathbf{g}$. The structures Join_Type($\mathbf{u}, \mathbf{g}$), Leave_Type($\mathbf{u}, \mathbf{g}$), Add_Type($\mathbf{o}, \mathbf{g}$) and Remove_Type($\mathbf{o}, \mathbf{g}$) record that the most recent join, leave, add, and remove events for the respective users, objects and groups were strict or liberal. Their possible values are L, S, and NULL, representing respectively liberal, strict, and the case in which no such event has yet occurred.

Recall that when a user (resp., object) experiences a liberal leave (resp., remove), users continue to have access to the objects to which they had access prior to these events. We use PrevAuthz($\mathbf{u}, \mathbf{o}, \mathbf{g}$) to record whether $\mathbf{u}$ was authorized for $\mathbf{o}$ via $\mathbf{g}$ at the time the most recent liberal leave and/or liberal remove occurred. (This is needed to handle correctly the left-hand subformulas of each of λ_1 and λ_2.)

Recall that when an object is added prior to a user joining, if either of these actions is strict, the user does not gain access to the object. To make authorization decisions correctly in this case, the stateful specification uses timestamps (not shown in Table 2—see use in definition of Authz_DS, section 3.3); the structures Join_TS($\mathbf{u}, \mathbf{g}$) and Add_TS($\mathbf{o}, \mathbf{g}$) record timestamps associated with the most recent join of $\mathbf{u}$ and add of $\mathbf{o}$, respectively. For simplicity, we assume here that timestamps are simply natural numbers with 0 representing that the event has not occurred, and with other values increasing monotonically with time (*e.g.*, the current index in the action sequence). Thus, the case in which the object add precedes the user join can be identified simply by comparing the associated timestamp values. This is the mechanism used in the authorization decisions made by the stateful specification, presented in the next section.

Timestamps are a natural solution in an implementation because their values can be established in a distributed, decentralized manner[3]. However, they are not conducive to verification via model checking because, theoretically, they grow without bound as the system runs. Model checking can be applied directly only to finite state machines. To overcome this, we use a standard technique called *abstraction* [4] wherein the unbounded structure is replaced by a bounded one that contains only the information that is essential to the state machine. Specifically, we introduce a structure JoinT_LE_AddT($\mathbf{u}, \mathbf{o}, \mathbf{g}$), shown in Table 2, which takes on the value 1 when $\mathbf{u}$ joined before or at the same time as $\mathbf{o}$ was added. (If $\mathbf{u}$ or $\mathbf{o}$ has been a group member multiple times, the structure reflects the most recent join and add.) If the add occurred first, the value is 0. Three additional values represent the cases in which one or both of the actions have never occurred: No_Join, No_Add, and NULL, the latter indicating that neither has occurred. We use abstraction JoinT_LE_AddT for verification (in model checking) purpose only.

For all $\mathbf{u} \in \mathcal{U}$, $\mathbf{o} \in \mathcal{O}$ and $\mathbf{g} \in \mathcal{G}$, for all well formed action sequence $\sigma \in \Sigma_{\mathcal{C}}^{\omega}$, and for all $i \in \mathbb{N}$, letting ρ be the run induced by σ, the invariants listed in Table 2 hold. Due to space limit, here we just explain the first invariant in the table as an example. At a given point i, the value of CurrUserMem($\mathbf{u}, \mathbf{g}$) is 1 if user $\mathbf{u}$ never leaves group $\mathbf{g}$ since the last time he joined $\mathbf{g}$. Otherwise, the value is 0. Note that each of the invariants has been verified by using model checking. As we discuss further in section 4, because the formulas being model checked refer to group operations involving only one user and/or object and group, it is sufficient to check that the formulas hold for very small carriers containing only one element each. This enables us to convert the formulas to propositional form, making them amenable to model checking. The invariants also assist in designing the transition relations that we will introduce in section 3.4.

3.3 Stateful Specification of Authorization

In this section, we show how authorization decisions are made based on the data structures we introduced in the previous section. We denote the function that yields these authorization decisions by Authz_DS, which is defined as follows. (Because Authz_DS is not itself a data structure, in the following, ρ_i can be viewed as a parameter to the function, much as when a method is invoked on an object in object oriented programming.) For all $\mathbf{u} \in \mathcal{U}$, $\mathbf{o} \in \mathcal{O}$ and $\mathbf{g} \in \mathcal{G}$, for all well formed action sequence $\sigma \in \Sigma_{\mathcal{C}}^{\omega}$, and for all $i \in \mathbb{N}$, letting ρ be the run induced by σ, we define ρ_i.Authz_DS by

$$\rho_i.\text{Authz\_DS}(\mathbf{u}, \mathbf{o}, \mathbf{g}) = \begin{cases} 1 & \text{if} \quad \rho_i.\phi_1(\mathbf{u}, \mathbf{o}, \mathbf{g}) = 1 \text{ and} \\ & \qquad \rho_i.\phi_2(\mathbf{u}, \mathbf{o}, \mathbf{g}) = 1 \text{ and} \\ & \qquad \rho_i.\phi_3(\mathbf{u}, \mathbf{o}, \mathbf{g}) = 1 \\ 0 & \text{otherwise} \end{cases}$$

[3]Issues such as clock synchronization and appropriate management of latency will be introduced in later steps of the design process, and are not handled here.

Table 2: Data Structures, Values, and Invariants. Boolean value "true" is represented by 1, "false", by 0

$$\rho_i.\text{CurrUserMem}(\mathbf{u},\mathbf{g}) = \begin{cases} 1 & \text{if } \sigma, i, [u \mapsto \mathbf{u}, g \mapsto \mathbf{g}] \models \neg(\text{SL}(u,g) \vee \text{LL}(u,g)) \,\mathcal{S}\, (\text{SJ}(u,g) \vee \text{LJ}(u,g)) \\ 0 & \text{otherwise} \end{cases}$$

$$\rho_i.\text{CurrObjMem}(\mathbf{o},\mathbf{g}) = \begin{cases} 1 & \text{if } \sigma,\ i,\ [o \mapsto \mathbf{o}, g \mapsto \mathbf{g}] \models \neg(\text{SR}(o,g) \vee \text{LR}(o,g)) \,\mathcal{S}\, (\text{SA}(o,g) \vee \text{LA}(o,g)) \\ 0 & \text{otherwise} \end{cases}$$

$$\rho_i.\text{Join\_Type}(\mathbf{u},\mathbf{g}) = \begin{cases} \text{L} & \text{if } \sigma,\ i,\ [u \mapsto \mathbf{u}, g \mapsto \mathbf{g}] \models \neg\text{SJ}(u,g) \,\mathcal{S}\, \text{LJ}(u,g) \\ \text{S} & \text{if } \sigma,\ i,\ [u \mapsto \mathbf{u}, g \mapsto \mathbf{g}] \models \neg\text{LJ}(u,g) \,\mathcal{S}\, \text{SJ}(u,g) \\ \text{NULL} & \text{if } \sigma,\ i,\ [u \mapsto \mathbf{u}, g \mapsto \mathbf{g}] \models \boxminus\neg(\text{LJ}(u,g) \vee \text{SJ}(u,g)) \end{cases}$$

$$\rho_i.\text{Leave\_Type}(\mathbf{u},\mathbf{g}) = \begin{cases} \text{L} & \text{if } \sigma,\ i,\ [u \mapsto \mathbf{u}, g \mapsto \mathbf{g}] \models \neg\text{SL}(u,g) \,\mathcal{S}\, \text{LL}(u,g) \\ \text{S} & \text{if } \sigma,\ i,\ [u \mapsto \mathbf{u}, g \mapsto \mathbf{g}] \models \neg\text{LL}(u,g) \,\mathcal{S}\, \text{SL}(u,g) \\ \text{NULL} & \text{if } \sigma,\ i,\ [u \mapsto \mathbf{u}, g \mapsto \mathbf{g}] \models \boxminus\neg(\text{LL}(u,g) \vee \text{SL}(u,g)) \end{cases}$$

$$\rho_i.\text{Add\_Type}(\mathbf{o},\mathbf{g}) = \begin{cases} \text{L} & \text{if } \sigma,\ i,\ [o \mapsto \mathbf{o}, g \mapsto \mathbf{g}] \models \neg\text{SA}(o,g) \,\mathcal{S}\, \text{LA}(o,g) \\ \text{S} & \text{if } \sigma,\ i,\ [o \mapsto \mathbf{o}, g \mapsto \mathbf{g}] \models \neg\text{LA}(o,g) \,\mathcal{S}\, \text{SA}(o,g) \\ \text{NULL} & \text{if } \sigma,\ i,\ [o \mapsto \mathbf{o}, g \mapsto \mathbf{g}] \models \boxminus\neg(\text{LA}(o,g) \vee \text{SA}(o,g)) \end{cases}$$

$$\rho_i.\text{Remove\_Type}(\mathbf{o},\mathbf{g}) = \begin{cases} \text{L} & \text{if } \sigma,\ i,\ [o \mapsto \mathbf{o}, g \mapsto \mathbf{g}] \models \neg\text{SR}(o,g) \,\mathcal{S}\, \text{LR}(o,g) \\ \text{S} & \text{if } \sigma,\ i,\ [o \mapsto \mathbf{o}, g \mapsto \mathbf{g}] \models \neg\text{LR}(o,g) \,\mathcal{S}\, \text{SR}(o,g) \\ \text{NULL} & \text{if } \sigma,\ i,\ [o \mapsto \mathbf{o}, g \mapsto \mathbf{g}] \models \boxminus\neg(\text{LR}(o,g) \vee \text{SR}(o,g)) \end{cases}$$

$$\rho_i.\text{PrevAuthz}(\mathbf{u},\mathbf{o},\mathbf{g}) = \begin{cases} 1 & \text{if } \sigma,\ i,\ [u \mapsto \mathbf{u}, o \mapsto \mathbf{o}, g \mapsto \mathbf{g}] \models \neg(\text{SL}(u,g) \vee \text{SR}(o,g)) \,\mathcal{S} \\ & [\ominus\text{Authz}(u,o,g) \wedge ((\text{LL}(u,g) \vee \text{LR}(o,g)) \wedge \neg\text{SL}(u,g) \wedge \neg\text{SR}(o,g))] \\ 0 & \text{otherwise} \end{cases}$$

$$\rho_i\text{JoinT\_LE\_AddT}(\mathbf{u},\mathbf{o},\mathbf{g}) = \begin{cases} 1 & \text{if } \sigma, i, [u \mapsto \mathbf{u}, o \mapsto \mathbf{o}, g \mapsto \mathbf{g}] \models \neg(\text{SJ}(u,g) \vee \text{LJ}(u,g)) \,\mathcal{S} \\ & ((\text{SA}(o,g) \vee \text{LA}(o,g)) \wedge \diamondminus(\text{SJ}(u,g) \vee \text{LJ}(u,g))) \\ \text{No\_Join} & \text{if } \sigma, i, [u \mapsto \mathbf{u}, o \mapsto \mathbf{o}, g \mapsto \mathbf{g}] \models \boxminus\neg(\text{SJ}(u,g) \vee \text{LJ}(u,g)) \wedge \diamondminus(\text{SA}(o,g) \vee \text{LA}(o,g)) \\ \text{No\_Add} & \text{if } \sigma, i, [u \mapsto \mathbf{u}, o \mapsto \mathbf{o}, g \mapsto \mathbf{g}] \models \boxminus\neg(\text{SA}(o,g) \vee \text{LA}(o,g)) \wedge \diamondminus(\text{SJ}(u,g) \vee \text{LJ}(u,g)) \\ \text{NULL} & \text{if } \sigma, i, [u \mapsto \mathbf{u}, o \mapsto \mathbf{o}, g \mapsto \mathbf{g}] \models \boxminus(\neg(\text{SJ}(u,g) \vee \text{LJ}(u,g)) \wedge \neg(\text{SA}(o,g) \vee \text{LA}(o,g))) \\ 0 & \text{otherwise} \end{cases}$$

in which ϕ_1, ϕ_2, and ϕ_3 are as given by

$$\rho_i.\phi_1(\mathbf{u},\mathbf{o},\mathbf{g}) = \begin{cases} 1 & (\rho_i.\text{Join\_Type}(\mathbf{u},\mathbf{g}) = L \\ & \wedge\rho_i.\text{Add\_Type}(\mathbf{o},\mathbf{g}) = L)\vee \\ & ((\rho_i.\text{Join\_TS}(\mathbf{u},\mathbf{g}) \leq \rho_i.\text{Add\_TS}(\mathbf{o},\mathbf{g}))\wedge \\ & (\rho_i.\text{Join\_TS}(\mathbf{u},\mathbf{g}) \neq 0) \wedge (\rho_i.\text{Add\_TS}(\mathbf{o},\mathbf{g}) \neq 0))\vee \\ & (\rho_i.\text{PrevAuthz}(\mathbf{u},\mathbf{o},\mathbf{g}) = 1\wedge \\ & (\rho_i.\text{Join\_TS}(\mathbf{u},\mathbf{g}) > \rho_i.\text{Add\_TS}(\mathbf{o},\mathbf{g})) \\ 0 & \text{otherwise} \end{cases}$$

$$\rho_i.\phi_2(\mathbf{u},\mathbf{o},\mathbf{g}) = \begin{cases} 1 & (\rho_i.\text{CurrUserMem}(\mathbf{u},\mathbf{g}) = 1)\vee \\ & (\rho_i.\text{Leave\_Type}(\mathbf{u},\mathbf{g}) = L\wedge \\ & \rho_i.\text{PrevAuthz}(\mathbf{u},\mathbf{o},\mathbf{g}) = 1) \\ 0 & \text{otherwise} \end{cases}$$

$$\rho_i.\phi_3(\mathbf{u},\mathbf{o},\mathbf{g}) = \begin{cases} 1 & (\rho_i.\text{CurrObjMem}(\mathbf{o},\mathbf{g}) = 1)\vee \\ & (\rho_i.\text{Remove\_Type}(\mathbf{o},\mathbf{g}) = L\wedge \\ & \rho_i.\text{PrevAuthz}(\mathbf{u},\mathbf{o},\mathbf{g}) = 1) \\ 0 & \text{otherwise} \end{cases}$$

Let us summarize the intuition behind ϕ_1, ϕ_2, and ϕ_3, each of which must be true for $\mathbf{u}$ to have access to $\mathbf{o}$ via $\mathbf{g}$. Formula ϕ_1 ignores the question of whether $\mathbf{u}$ and $\mathbf{o}$ are currently members of $\mathbf{g}$; that issue is handled by ϕ_2 and ϕ_3. Formula ϕ_1 is true if any of the following three cases holds: the most recent join and add were both liberal; $\mathbf{u}$ joined before or at the same time as $\mathbf{o}$ was added; $\mathbf{u}$ joined after $\mathbf{o}$ was added, but the last time they were both in $\mathbf{g}$, $\mathbf{u}$ had access to $\mathbf{o}$. Formula ϕ_2 requires that $\mathbf{u}$ is currently a member of $\mathbf{g}$ unless $\mathbf{u}$'s the most recent leave was liberal and $\mathbf{u}$ had access $\mathbf{o}$ immediately prior to leaving. Formula ϕ_3 states for $\mathbf{o}$ the same requirement that ϕ_2 states for $\mathbf{u}$.

Note that neither ϕ_2 nor ϕ_3 depends on the most recent join or add type because the π specification uses what is called *lossless* versions of join and add [9]. This means that when $\mathbf{u}$ joins $\mathbf{g}$ or $\mathbf{o}$ is added to $\mathbf{g}$, this never causes $\mathbf{u}$ to lose access $\mathbf{o}$ if $\mathbf{u}$ had access immediately prior to the join or add operations. This intuition is stated formally in the π specification. So if $\mathbf{u}$ to had access to $\mathbf{o}$ via $\mathbf{g}$ prior to the join (respectively, add), then the access will be retained at least until the subsequent leave (respectively, remove—whichever comes first).

3.4 Transition Relations

We present transition relations of data structures used by $M_\pi^{\mathcal{C}}$. To illustrate, here we show initial states and transition relations of two data structures: CurrUserMem and PrevAuthz.

For all $\mathbf{u} \in \mathcal{U}$ and $\mathbf{g} \in \mathcal{G}$, the initial value of $\text{CurrUserMem}(\mathbf{u},\mathbf{g})$ is assigned as 0, as we assume that when the system starts, no user is in any group. To express the state transition relation, we use the convention that values of structures in the destination state are denoted by adding primes to the structure names; unprimed names refer to the

structures' values in the source state. The state transition relation for CurrUserMem is given by following.

$$((\text{CurrUserMem}(\mathbf{u},\mathbf{g}) = 1) \wedge (\text{SL}'(\mathbf{u},\mathbf{g}) \vee \text{LL}'(\mathbf{u},\mathbf{g})) \wedge (\text{CurrUserMem}'(\mathbf{u},\mathbf{g}) = 0)) \vee$$
$$((\text{CurrUserMem}(\mathbf{u},\mathbf{g}) = 0) \wedge (\text{SJ}'(\mathbf{u},\mathbf{g}) \vee \text{LJ}'(\mathbf{u},\mathbf{g})) \wedge (\text{CurrUserMem}'(\mathbf{u},\mathbf{g}) = 1)) \vee$$
$$(\neg(\text{SJ}'(\mathbf{u},\mathbf{g}) \vee \text{LJ}'(\mathbf{u},\mathbf{g}) \vee \text{SL}'(\mathbf{u},\mathbf{g}) \vee \text{LL}'(\mathbf{u},\mathbf{g})) \wedge (\text{CurrUserMem}'(\mathbf{u},\mathbf{g}) = \text{CurrUserMem}(\mathbf{u},\mathbf{g})))$$

Note that the formula defining the transition relation for CurrUserMem is false when a user attempts to leave a group to which he does not belong. This reflects the fact that there is no legal transition in this case and that the action sequence is rejected by $M_\pi^{\mathcal{C}}$. This is appropriate because such an action sequence is not well formed.

For all $\mathbf{u} \in \mathcal{U}$, $\mathbf{o} \in \mathcal{O}$ and $\mathbf{g} \in \mathcal{G}$, the initial value of $\text{PrevAuthz}(\mathbf{u},\mathbf{o},\mathbf{g})$ is 0, as no action occurred in the past and previously $\mathbf{u}$ is not authorized to access $\mathbf{o}$. The state transition relation for PrevAuthz is given by following.

$$(((\text{LL}'(\mathbf{u},\mathbf{g}) \vee \text{LR}'(\mathbf{o},\mathbf{g})) \wedge \neg\text{SL}'(\mathbf{u},\mathbf{g}) \wedge \neg\text{SR}'(\mathbf{o},\mathbf{g})) \wedge (\text{PrevAuthz}'(\mathbf{u},\mathbf{o},\mathbf{g}) = \text{Authz\_DS}(\mathbf{u},\mathbf{o},\mathbf{g})) \vee$$
$$((\text{SL}'(\mathbf{u},\mathbf{g}) \vee \text{SR}'(\mathbf{o},\mathbf{g})) \wedge (\text{PrevAuthz}'(\mathbf{u},\mathbf{o},\mathbf{g}) = 0)) \vee$$
$$(\neg(\text{LL}'(\mathbf{u},\mathbf{g}) \vee \text{LR}'(\mathbf{o},\mathbf{g}) \vee \text{SL}'(\mathbf{u},\mathbf{g}) \vee \text{SR}'(\mathbf{o},\mathbf{g})) \wedge (\text{PrevAuthz}'(\mathbf{u},\mathbf{o},\mathbf{g}) = \text{PrevAuthz}(\mathbf{u},\mathbf{o},\mathbf{g})))$$

Recall that we use $\text{PrevAuthz}(\mathbf{u},\mathbf{o},\mathbf{g})$ to record whether $\mathbf{u}$ was authorized for $\mathbf{o}$ via $\mathbf{g}$ at the time the most recent liberal leave and/or liberal remove took place. The $\text{PrevAuthz}'(\mathbf{u},\mathbf{o},\mathbf{g})$ value in the destination state of transition relation depends on the $\text{Authz\_DS}(\mathbf{u},\mathbf{o},\mathbf{g})$ value in the source state. As defined in 3.3, the $\text{Authz\_DS}(\mathbf{u},\mathbf{o},\mathbf{g})$ value can be computed by values of all data structures on the source state.

3.5 Case Study

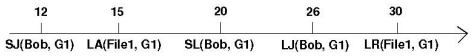

Figure 1: Case Study

In this section we present a case study to show how data structures are updated in stateful specification and how authorization is determined in a given point by using data structure values.

We assume that each user, object and group have their respective unique identifier in the system. Assume that we have a research group G_1. Although there may be many users and objects in G_1, in our case study, we only consider user Bob and object $File_1$ in G_1.

In the initial state, the data structures are assigned values as following,

- $\text{CurrUserMem}(Bob, G_1) = 0$,
- $\text{Join\_Type}(Bob, G_1) = \text{NULL}$,
- $\text{Leave\_Type}(Bob, G_1) = \text{NULL}$,
- $\text{Join\_TS}(Bob, G_1) = 0$,
- $\text{CurrObjMem}(File_1, G_1) = 0$,
- $\text{Add\_Type}(File_1, G_1) = \text{NULL}$,
- $\text{Remove\_Type}(File_1, G_1) = \text{NULL}$,
- $\text{Add\_TS}(File_1, G_1) = 0$,
- $\text{PrevAuthz}(Bob, File_1, G_1) = 0$.

As shown in figure 1. Bob joins G_1 by SJ at timestamp 12. It causes the following data structures to be updated, and other data structures remain unchanged since initial assignment.

- $\text{CurrUserMem}(Bob, G_1) = 1$,
- $\text{Join\_Type}(Bob, G_1) = S$,
- $\text{Join\_TS}(Bob, G_1) = 12$.

At timestamp 15, $File_1$ is added to G_1 by LA, It causes the following data structures to be updated, and other data structures remain unchanged.

- $\text{CurrObjMem}(File_1, G_1) = 1$,
- $\text{Add\_Type}(File_1, G_1) = L$,
- $\text{Add\_TS}(File_1, G_1) = 15$.

Bob leaves G_1 by SL at timestamp 20, the following data structures are updated.

- $\text{CurrUserMem}(Bob, G_1) = 0$,
- $\text{Leave\_Type}(Bob, G_1) = S$,

Note that we only listed the data structure values that are updated and different from the values in previous state.

Suppose we need to make a authorization decision for Bob to read $File_1$ in G_1 at this state (at timestamp 20). According to Authz_DS defined in section 3.3, the value is determined by the values of data structures at this state. We compute that $\text{Authz\_DS}(Bob, File_1, G_1) = 0$. (As $\phi_2 = 0$).

At timestamp 26, Bob rejoins G_1 by LJ. The following data structures are updated as:

- $\text{CurrUserMem}(Bob, G_1) = 1$,
- $\text{Join\_Type}(Bob, G_1) = L$,
- $\text{Join\_TS}(Bob, G_1) = 26$.

At timestamp 30, $File_1$ is removed from G_1 by LR. The following data structures are updated.

- $\text{CurrObjMem}(File_1, G_1) = 0$,
- $\text{Remove\_Type}(File_1, G_1) = L$,
- $\text{PrevAuthz}(Bob, File_1, G_1) = 1$.

Because an LR occurs, PrevAuthz needs to be updated. Note that the PrevAuthz is based on the value of Authz_DS in the previous state, that is, at timestamp 26. (Because no group action for Bob or $File_1$ occurred and no data structure were updated since then.) In the previous state, $\text{Join\_Type}(Bob, G_1) = L$, $\text{Add\_Type}(File_1, G_1) = L$, $\text{CurrUserMem}(Bob, G_1) = 1$, $\text{CurrObjMem}(File_1, G_1) = 1$. So we get $\text{Authz\_DS}(Bob, File_1, G_1) = 1$. (All ϕ_1, ϕ_2 and ϕ_3 have value 1.) According to the transition relation of PrevAuthz, $\text{PrevAuthz}(Bob, File_1, G_1)$ is updated to 1 based on Authz_DS value in the previous state.

Consider we need to make a authorization decision for Bob to read $File_1$ in G_1 at timestamp 35. The $\text{Authz\_DS}(Bob, File_1, G_1)$ is determined by the values of data structures at this state. Because no group action for Bob or $File_1$ ever occurred since timestamp 30, all data structure value remain unchanged since then. Because $\text{Remove\_Type}(File_1, G_1) = L$ and $\text{PrevAuthz}(Bob, File_1, G_1) = 1$, the Authz_DS gets the value 1. (All ϕ_1, ϕ_2 and ϕ_3 have value 1.)

The PrevAuthz records authorization decision at the time the most recent leave or remove occurs. Whenever authorization decision is to be made, we do not need to refer to the entire event history. Instead, we use the data structures

that summarized the event history information to make an authorization decision.

In contrast, stateless specification contains no such data structures to provide summarized event history information, every action in the history need to be kept track of. And whenever a authorization decision is to be made, the entire trace need to be inspected. Since there may be only LR and LL in the group, and the overlapping membership period for user and object can be arbitrarily far in the past.

In the current design step, we assume that the timestamps are managed by a global clock. Clock synchronization and latency issues will be addressed in later steps of development process, but not in this paper.

4. THE REFINEMENT EQUIVALENCE OF STATELESS AND STATEFUL POLICY

The correctness requirements of the stateful specification $M_\pi^{\mathcal{C}}$ are (1) that for all action sequence $\sigma \in \Sigma_{\mathcal{C}}^\omega$, $M_\pi^{\mathcal{C}}$ accepts σ if and only if σ is well formed and (2) the stateless Authz and the stateful Authz_DS agree at each step i. We refer to the "only if" part of (1) together with (2) as *soundness*. We refer to the "if" part of (1) as *completeness*.

We introduce a distinguished collection of small carriers, $\mathcal{C}_{\text{sm}} = \langle \mathcal{G}_{\text{sm}}, \mathcal{U}_{\text{sm}}, \mathcal{O}_{\text{sm}} \rangle$, in which each carrier contains exactly one element. As we argue just below Lemma 4.1, both soundness and the correctness of our invariants hold for arbitrary carriers if they hold for $\mathcal{C}_{\text{sm}}$. This enables us to use model checking to verify these results by converting FOTL formulas interpreted over $\mathcal{C}_{\text{sm}}$ to PLTL formulas. We have verified the following lemma by using the NuSMV model checker [3].

LEMMA 4.1 (SMALL-CARRIER SOUNDNESS). *Let* $\mathcal{U}_{\text{sm}} = \{u\}$, $\mathcal{O}_{\text{sm}} = \{o\}$, $\mathcal{G}_{\text{sm}} = \{g\}$, *and* $\mathcal{C}_{\text{sm}} = \langle \mathcal{U}_{\text{sm}}, \mathcal{O}_{\text{sm}}, \mathcal{G}_{\text{sm}} \rangle$. *For all traces* $\kappa \in \Omega_{\mathcal{C}_{\text{sm}}}^\omega$, *if a run* ρ *of* $M_\pi^{\mathcal{C}_{\text{sm}}}$ *is induced by* $\kappa \restriction_A$, *then (1)* κ *is well formed and (2) if* $\kappa \models \pi$, *then for all* $i \in \mathbb{N}$, $\langle \mathbf{u}, \mathbf{o}, \mathbf{g} \rangle \in [\![\text{Authz}]\!]\kappa_i$ *if and only if* $\rho_i.\text{Authz\_DS}(\mathbf{u}, \mathbf{o}, \mathbf{g})$.

The lemma generalizes to large carriers for two reasons. First, by inspecting the formula, in the π specification, $\text{Authz}(\mathbf{u}, \mathbf{o}, \mathbf{g})$ depends only on group operations involving these individuals. Second, the transition relations for each data structure in $M_\pi^{\mathcal{C}}$ refer to only one user and/or object and to one group, both with respect to the data structure indices and with respect to the group operations. It follows from these observations that in both the stateless and the stateful specification the authorization of a given user for a given object via a given group is independent of group operations involving any other users, objects, or groups. Hence, the size of the carriers is immaterial to the validity of the proof. Thus we have the following.

THEOREM 4.2 (SOUNDNESS OF ARBITRARY CARRIERS). *Given an arbitrary collection of countable carriers* $\mathcal{C} = \langle \mathcal{G}, \mathcal{U}, \mathcal{O} \rangle$, *for all traces* $\kappa \in \Omega_{\mathcal{C}}^\omega$, *if a run* ρ *of* $M_\pi^{\mathcal{C}}$ *is induced by* $\kappa \restriction_A$, *then (1)* κ *is well formed and (2) if* $\kappa \models \pi$, *then for all* $\langle \mathbf{u}, \mathbf{o}, \mathbf{g} \rangle \in \mathcal{U} \times \mathcal{O} \times \mathcal{G}$ *and for all* $i \in \mathbb{N}$, $\langle \mathbf{u}, \mathbf{o}, \mathbf{g} \rangle \in [\![\text{Authz}]\!]\kappa_i$ *if and only if* $\rho_i.\text{Authz\_DS}(\mathbf{u}, \mathbf{o}, \mathbf{g})$.

We have verified the following lemma for the small carrier case using model checking. The result generalizes to large carriers for the same reasons discussed above.

LEMMA 4.3 (CORRECTNESS OF INVARIANTS). *For a system with carriers* $\mathcal{C} = \langle \mathcal{G}, \mathcal{U}, \mathcal{O} \rangle$, *the invariants of all data structures are all satisfied.*

For completeness proof, we need to show that every trace that satisfies the stateless specification can be generated by the SM. From theorem 4.2, we know that every time the stateless specification confirms Authz, the stateful specification also confirms Authz_DS, and vise versa. So for completeness, we need only prove that every trace that satisfies the well formed constraints is accepted by the SM. Since it can not be proved by using model checking, we show completeness by using manual proof techniques instead. This manual proof makes use of lemma 4.3.

THEOREM 4.4 (COMPLETENESS). *For a system with carriers* $\mathcal{C} = \langle \mathcal{G}, \mathcal{U}, \mathcal{O} \rangle$, *any well formed action sequence* $\sigma \in \Sigma_{\mathcal{C}}^\omega$ *induces a run* ρ *of* $M_\pi = \langle Q_\pi, q_\pi^0, \Sigma_{\mathcal{C}}, \delta_\pi \rangle$.

Consider any well formed action sequence σ. The proof shows by induction on $i \in \mathbb{N}$ that $M_\pi^{\mathcal{C}}$ has a transition defined for each σ_i, as needed to complete the proof. The body of the proof then is an extensive case analysis based on the possible machine states and σ_i. The invariants associated with each possible machine state are combined with the well formedness requirements to show that a transition exists in all cases. Due to space constraints, the complete proof could be found in our technical report [13].

5. REFINING THE SPECIFICATION TO HANDLE ACTION REQUESTS

The action sequence provided as input to a given run of $M_\pi^{\mathcal{C}}$ is provided by the environment in which $M_\pi^{\mathcal{C}}$ is deployed. An implementation of an AC system cannot reject an ill-formed action sequence, as halting the system is unacceptable. In this section, we discuss informally an aspect of the stateful specification that refines the π specification, namely the treatment of group-operation *requests*. The stateful specification augments $M_\pi^{\mathcal{C}}$ with an input/output SM that acts as a transducer, converting an input action-request sequence into a well formed action sequence, which is then provided as input to $M_\pi^{\mathcal{C}}$.

Ensuring that the output of the filter SM is a legal action sequence is straightforward. (As part of our work with the model checking, we generated a filter SM and verified that it satisfies this requirement. For this purpose, our model-checking code nondeterministically generates arbitrary request sequences.) Slightly more interesting is the problem of ensuring that for each legal action sequence, there exists a request sequence that causes the filter SM to generate the given action sequence. This can be ensured by requiring that when presented with a set of requests that, taken together, are all legal, the filter SM does not drop any of them. Together, these two requirements are sufficient in the sense that the action sequences that can be generated by the filter SM are exactly the well formed action sequences. However, these requirements do not uniquely specify the filter SM. Consider the case in which a single user requests to perform both a liberal and a strict join on the same group at the same time. The simplest behavior in this case is to drop both requests. However other options exist: ask the user which one to perform; apply a predefined policy that selects between the two; or flip a coin and drop one on that basis.

6. LIMITATIONS OF SCALABILITY

The limitation of our stateful specification concerns PrevAuthz. First, the structure is inherently centralized,

as information is kept in it for each $\langle \mathbf{u}, \mathbf{o}, \mathbf{g} \rangle$ tuple. Second, each time a user operation is performed on a group $\mathbf{g}$, the value of the structure must be updated for all tuples involving that user and group and similarly for each object operation. There are environmental conditions in which these limitations may not be a major problem, namely when the system is not highly distributed and/or the rate of authorization queries greatly exceeds the rate of group operations. In particular, PrevAuthz enables authorization queries to be answered in constant time. Nevertheless, a less centralized approach would often be desirable. An alternative would be use timestamps to maintain with each user and object the history of group operations in which it participates. Authorization of a user for an object could then be decided by comparing the histories of each. The problem with this approach is that, the overlapping membership periods for the user and object can be arbitrarily far in the past (when no SL or SR has been performed). Thus the local histories can grow without bound, as can the time required to make authorization decisions. In future work, we would like to develop a hybrid approach combining these two approaches.

7. RELATED WORK

Many research efforts have been made towards developing security systems by leveraging formal methods. In particular, model checking has been increasingly employed to reason about security properties in access control systems, such as Role-Based Access Control Models (RBAC) and trust management systems. Schaad *et al.* [10] verified separation of duty properties in RBAC systems using NuSMV. Other work [5, 6, 12] also supports the use of formal tools to verify properties of security policies.

FOTL specification of security and privacy systems can exhibit a high-level of complexity, leading to difficulty in the assessment admission of system behavior and in the development of enforcement. One notable technique that is closely related to our work is the development of small model theorem [14] in the verification of parameterized systems. The small model theorem reduces the problem of reasoning about infinite-state system to the verification of a finite-state system, which can then be model checked.

Stepwise refinement techniques have been used in the design of security systems to improve the correctness. In Sprenger *et al.* [11], the refinement could help designers to master the complexity of both models and proofs by focusing on individual design aspects at each step. Alur *et al.* develop a framework of secrecy and preservation of secrecy for labeled transition systems via refinement [1]. It is unclear if their approach is scalable to handle unbounded attributes as the π specification is expressed in FOTL.

8. CONCLUSION AND FUTURE WORK

We presented a stateful g-SIS specification that is a refinement of the prior stateless π specification [9]. By using a combination of model checking and manual proofs, we show that our stateful specification makes authorization decisions identical to those made by the π specification. These steps bring us a significant step closer to our ultimate goal of constructing a specification that can be implemented by a security non-expert. In the future, we plan to address the problems that arise in handling the communication latency and extend our design to support read-write stateful specification.

Acknowledgements

Jianwei Niu is supported in part by NSF awards CNS-0964710, THECB NHARP 010115-0037-2007, and the University of Texas at San Antonio research award TRAC-2008.

9. REFERENCES

[1] R. Alur and S. Zdancewic. Preserving secrecy under refinement. In *ICALP qŕ06, volume 4052 of LNCS*, pages 107–118, 2006.

[2] A. Barth, A. Datta, J. C. Mitchell, and H. Nissenbaum. Privacy and contextual integrity: Framework and applications. In *SP '06*, pages 184–198, 2006.

[3] A. Cimatti, E.M.Clarke, F. Giunchiglia, and M. Roveri. NuSMV: A new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2(4):410–425, 2000.

[4] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *TOPLAS*, 16(5):1512–1542, 1994.

[5] M. Drouineaud, M. Bortin, P. Torrini, and K. Sohr. A first step towards formal verification of security policy properties for RBAC. In *Proceedings of Fourth International Conference on Quality Software*, pages 60–67, 2004.

[6] S. Jha, N. Li, M. Tripunitara, Q. Wang, and W. Winsborough. Towards formal verification of role-based access control policies. *IEEE Trans. Dependable Secur. Comput.*, 5(4):242–255, 2008.

[7] R. Krishnan, J. Niu, R. Sandhu, and W. Winsborough. Stale-safe security properties for group-based secure information sharing. In *6th ACM workshop FMSE*, pages 53–62, 2008.

[8] R. Krishnan, J. Niu, R. Sandhu, and W. H. Winsborough. Group-centric secure information sharing models for isolated groups. *ACM Trans. Infor. Syst. Secur.*, 14(3), 2011.

[9] R. Krishnan, R. Sandhu, J. Niu, and W. Winsborough. Foundations for group-centric secure information sharing models. In *SACMAT*, pages 115–124, 2009.

[10] A. Schaad, V. Lotz, and K. Sohr. A model-checking approach to analysing organisational controls in a loan origination process. In *SACMAT '06*, pages 139–149, 2006.

[11] C. Sprenger and D. Basin. Developing security protocols by refinement. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 361–374. ACM, 2010.

[12] N. Zhang, M. Ryan, and D. P. Guelev. Synthesising verified access control systems through model checking. *Journal of Computer Security*, 16(1):1–61, 2008.

[13] W. Zhao, J. Niu, and W. H. Winsborough. Refinement-based design of a group-centric secure information sharing model. Technical Report CS-TR-2011-16, UTSA, 2011.

[14] L. D. Zuck and A. Pnueli. Model checking and abstraction to the aid of parameterized systems (a survey). *Computer Languages, Systems & Structures*, 30(3-4):139–169, 2004.