Modelling Dynamic Access Control Policies for Web-based Collaborative Systems

Hasan Qunoo and Mark Ryan

School of Computer Science, University of Birmingham, UK

Abstract. We present a modelling language, called *X-Policy*, for web-based collaborative systems with dynamic access control policies. The access to resources in these systems depends on the state of the system and its configuration. The *X-Policy* language models systems as a set of actions. These actions can model system operations which are executed by users. The *X-Policy* language allows us to specify execution permissions on each action using complex access conditions which can depend on data values, other permissions, and agent roles. We demonstrate that *X-Policy* is expressive enough to model collaborative conference management systems. We model the EasyChair conference management system and we reason about three security properties of EasyChair using our model.

1 Introduction

Web-based collaborative systems like social networking websites, conference reviewing systems, document development tools, and application processing systems are all examples of central systems that give users the ability to create and control access to their data. Access to data in these systems is dynamic; it depends on the state of the system and its configuration. Large conference management systems like iChair[1], WSAR[12], HotCRP[13] and EasyChair[23] are widely used to manage academic conferences. However, the size and the complexity of these systems make it hard to analyse their policies and their security properties. The policy of those systems are designed to preserve the system security and serve their desired purpose. Systems, however, can still fail some basic security properties[19,21]. Users can compromise the system policy and its security properties by interactions of rules, co-operations between agents and multi-step actions.

For example: consider the conference paper review system EasyChair. We build ***EC***, a model of our understanding of EasyChair derived from our experiments and its documentation as we can see in Section 4. ***EC*** consists of a set of agents and a set of papers to be reviewed by the PC members. Each agent can act in the role of program chair, program-committee (PC) member, paper's author or a subreviewer. For example the following policy rules are a subset of the ***EC*** policy:

1. PC chair can assign PC members to review a paper.
2. PC members can invite another agent to sub-review a paper that is assigned to them. Subreviewers may accept or reject the invitation.
3. Sub-reviewers send their reviews (outside the system) to the reviewer.
4. Once the reviewer receives the paper review, the PC member can submit the review to the system.

The purpose of these rules is to collect a number (usually between 3 and 4) of reviewer's opinions of a submitted paper. These opinions determine whether a paper should be accepted or rejected. For these rules to be fair, no single reviewer should be able to determine the outcome of a paper reviewing process by writing all three reviews of that paper. However, as we can see in the following

strategy the intention of these rules can be breached by interaction of rules to allow a single user to write all the three reviews of a paper. Our analysis of the system only requires one agent to be acting intentionally to circumvent the system as we can see in the following strategy:

1. Chair assigns three PC members, Alice, Charlie and Bob, to review a paper.
2. Alice assigns Eve as her sub-reviewer.
3. Bob assigns Eve as his sub-reviewer.
4. Charlie assigns Eve as his sub-reviewer.
5. Eve accepts all three roles and send Alice, Charlie and Bob three similar reviews.
6. Alice, Charlie and Bob receive Eve's reviews and submit it to the system.

Eve manages to write all three paper reviews while all the agents still comply with the system rules. The reviewers cannot read other reviewers' names in the anonymous reviewing setting. This attack will succeed on various configuration but it might go undetected in the case of anonymous reviewing. The interaction of rules in the live system can cause an unforeseen behaviour which highlights the need to model the dynamic aspect of these systems and analyse their security properties in a formal way.

In this paper we present a simple yet expressive modelling language, called *X-Policy*, to model large web-based collaborative management systems. To be able to model these dynamic systems, the *X-Policy* modelling language allows us to specify systems as a set of atomic read or write actions. Action executing policy is specified as preconditions which the user has to satisfy to execute the action. Executing an action updates the system state which might, as a result, change the execution permissions. We use ***EC*** as a case study for our language. We model ***EC*** in *X-Policy* and we reason about three security attacks on EasyChair using our model.

In Section 2 we discuss the related work. In Section 3, we present our modelling language *X-Policy* together with its formalism. We detail in Section 4 the process of constructing the ***EC*** model. We also explain how we can express ***EC*** in *X-Policy* formalism. We introduce a selection of ***EC*** actions with their execution permissions statements which we use to prove the security attacks on ***EC*** and EasyChair in Section 4.3 as our case study. The conclusion and ideas for future work are in Section 5.

2 Related Work

Recently, there has been a plethora of languages and logics[2,3,4,5,6,7,8,10,11,14,15,16,17,18] to express access control policies. These logics and languages try to solve various issues arising from decentralisation.

DeTreville was the first to propose a Datalog based security language called Binder[8]. Since then Datalog has become the foundation of recent logic-based access control policies like the RT family [16] and SecPAL[2]. Researchers are mainly attracted to Datalog[20] as they can start from a tractable and expressive language with the advantage of deducing trust relations effectively based on well developed logic programming concepts and deductive databases. Unfortunately, Datalog is stateless. Inherently, the ability of datalog-based languages to express dynamic access control policies is restricted. Cassandra[6], a Datalog-based language, has a separate mechanism to maintain the authorisation state by inserting and retracting "hasActivated" facts according to the policy rules.

Gurevich et. al. introduced Distributed Knowledge Authorisation Language DKAL[10] and DKAL2[11] that extend SecPAL's expressiveness. However, Cassandra, SecPAL, DKAL, DKAL2 and other authorisation languages lack the ability to express the dynamic aspect of access control where policies depend on and update the system state like those we have in ***EC***. They, also, cannot express the effect of actions as part of the language and it has to be hard-coded in an ad-hoc way.

More recently, SMP[5] and its successor DyNPAL[4] aim to specify dynamic policies with the ability to specify the effect of executing these actions. DyNPAL allows conditional bulk insertion and retraction of authorisation facts with transactional execution semantics (either all or none are committed). However, DyNPAL's declarative nature and minimalistic approach make it hard to follow the control flow of the actions. Also the lack of parameter typing does not allow us to establish the relation between the agent who can execute an action and the action itself. They tend to focus on answering the question "under what conditions can an action be executed?" rather than "under what conditions can an agent execute an action?". This is indeed necessary to enable us to define agent coalitions and establish which agent is executing an action. It allows us to detect attacks where we are interested in who can execute a set of actions rather than whether a set of actions can be executed regardless of the actors involved.

RW framework[24], a precursor of *X-Policy*, can analyse the consequences of multi-agent multi-step actions by performing temporal reasoning. The access control verification software suite Margrave [9] addresses the policy change-impact problem using Model-Checking techniques to compute the changes between two policy files written in XACML. RW and Margrave are both model checking based frameworks. However, RW and Margrave do not allow us to express actions with multiple assignments needed to preserve the integrity constraints of the modelled system.

Lo et. al.[19,21] analyse the security features of Web Submission and Review Software WSAR[12]. They study the security properties like the system password strength and storage, its resistance to SQL injection, forced browsing and browser caching. However, their analysis does not include the access control policy of the system. To the best of our knowledge, this is the first paper to model and analyse dynamic access control policy for a large web-based collaborative system with atomic actions like EasyChair.

3 *X-Policy* Modelling Language

3.1 Syntax

Let T be a set of types, which includes a special type Agent for agents, also let P be a finite set of predicates. Each n-ary predicate has a signature $t_1 \times \cdots \times t_n \rightarrow \{\top, \bot\}$, where $t_i \in T$. For example, in the case of a conference review system, T can include Paper, and P can include the predicate

$$\mathsf{Author} : \mathsf{Agent} \times \mathsf{Paper} \rightarrow \{\top, \bot\}.$$

The full list of predicates used in ***EC*** is included in Section 4.2. A different model will require another list of predicates. We assume a set of variables V, each with a type. If $p \in P$ and $\overrightarrow{x} = x_1, \ldots, x_q$ is a sequence of variables of the appropriate type, then $p(\overrightarrow{x})$ is an *atomic formula*.

Actions. These definitions allow us to define a set of actions $\mathsf{Actions}$ which includes read actions and write actions. A read action allows the user to access the truth value of an *atomic formula* and is of the form

Action Actionname($\overrightarrow{x}$) :-
{
    return $p(\overrightarrow{y})$;
}

where $p \in P$ and the variables in $\overrightarrow{y}$ occur in $\overrightarrow{x}$. A write action allows the user to change the truth values of an atomic formula and is of the form

Action Actionname($\overrightarrow{x}$) :-
{
    writebody
}

where writebody is an expression formed from the following BNF:

writebody ::= assignment | for $(v : t)$ { writebody } | writebody writebody

where v is a variable of the type t and an assignment is of the form $p(\overrightarrow{y}) := \top$; or $p(\overrightarrow{y}) := \bot$;. We allow an *atomic formula* $p(\overrightarrow{y})$ to occur at most once at the left of ":=" in an action to avoid ambiguity in computing the action effect. The assignment statements within the same action can be written in any order. All free variables in an assignment must be declared either in a surrounding for-statement or in Actionname statement. Intuitively, a for-statement in an action is a 'macro' that is interpreted as multiple *assignment* statements.

Execution Permissions. An action permission statement exec(*act*,u) defines the conditions for an agent u∈ Agent to execute an action $act \in$ Actions. A permission statement is of the form

exec(Actionname($\overrightarrow{x}$), u)⇌ *formula*

where *formula* is a formula which is defined using atomic formulae and logical connectors: $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\rightarrow$ (implication), $\exists$ and $\forall$ (existential and universal quantification over variables of the appropriate type). The variables that occur in *formula* are required to be either in $\overrightarrow{x}$ or u. The *formula* defines the conditions for agents to execute these actions as functions on its state.

3.2 Informal Semantics

A model M defines, for each type t, a finite set of individuals σ_t. We define $\sigma = \sigma_{t_1} \cup \cdots \cup \sigma_{t_n}$ as the set of all the individuals defined by M. We assume $\sigma_{t_1} \cap \sigma_{t_2} = \emptyset$ whenever t_1 and t_2 are distinct. If p is a predicate and $\overrightarrow{\alpha}$ is a sequence of individuals of the appropriate type then $p(\overrightarrow{\alpha})$ is a ground atomic formula. State m of the model M is a valuation of the ground atomic formulae. In the rest of this paper we identify each state with the set of ground atomic formulae which are true in the state.

For loops. We describe the semantics of for-loops in the context of a model M, with $\sigma_t = \{v_1, \ldots, v_k\}$ the set of individuals in M of the type t. Let $act \in$ Actions. We then transform each for-statement to its equivalent multiple *assignment* statements. For example the following for-statement:

for $(v : t)$ $\{p(\overrightarrow{\gamma_1}, v, \overrightarrow{\gamma_2}) := \bot;\}$

is in the write action $act(\overrightarrow{x})$ where $\overrightarrow{\gamma_1}$ and $\overrightarrow{\gamma_2}$ are subsequences of other parameters. This for-statement is transformed to:

$p(\overrightarrow{\gamma_1}, v_1, \overrightarrow{\gamma_2}) := \bot;$
$\vdots$
$p(\overrightarrow{\gamma_1}, v_k, \overrightarrow{\gamma_2}) := \bot;$

We apply this process repeatedly until we have no for-statement in our action. We call the resulted loop-free action: $act^*(\overrightarrow{x})$.

Effect of Actions. Let $act \in$ Actions and $\overrightarrow{\alpha}$ a sequence of individuals of the appropriate type for act. We define the result of running the instantiated action $act(\overrightarrow{\alpha})$. We first compute $act^*(\overrightarrow{\alpha})$, as above. We then apply the functions: effect$^+$() and effect$^-$() which compute the positive and the negative effect of the instantiated loop-free action $act^*(\overrightarrow{\alpha})$ as following:

$$\text{effect}^+(act^*(\overrightarrow{\alpha})) = \{p(\overrightarrow{\beta}) \mid p(\overrightarrow{\beta}){:=}\top \text{ occurs in } act^*(\overrightarrow{\alpha})\}$$
$$\text{effect}^-(act^*(\overrightarrow{\alpha})) = \{p(\overrightarrow{\beta}) \mid p(\overrightarrow{\beta}){:=}\bot \text{ occurs in } act^*(\overrightarrow{\alpha})\}$$

where all the values of $\overrightarrow{\beta}$ are members of σ.

The action effect then will be applied to the model state. Executing a write action will transfer the model from a pre-execution state m_i in which the action is executed at to a post-execution state m_{i+1}. It adds the set of ground atomic formulae which are updated to true to the state m_i. It also subtracts the set of ground atomic formulae that are updated to false from the state m_i. All the other ground atomic formulae in the state m_i will remain unchanged. Let's say that the model M is in the state m_i when an agent u executes the write action $act(\overrightarrow{\alpha})$, then the model will be transformed from the state m_i to the state m_{i+1} where $m_{i+1} = m_i \backslash \text{effect}^-(act^*(\overrightarrow{\alpha})) \cup \text{effect}^+(act^*(\overrightarrow{\alpha}))$. Note that $\text{effect}^-(act^*(\overrightarrow{\alpha})) \cap \text{effect}^+(act^*(\overrightarrow{\alpha})) = \emptyset$. Therefore, adding and subtracting can be done in any order. Read actions do not change the model state. However, read actions can be part of an attack strategy as we will see in Section 4.3.

4 Modelling EasyChair Conference Management System

4.1 Modelling conventions for EasyChair system

In this section, we discuss a number of modelling conventions we have followed in constructing ***EC***. The model ***EC*** is based on our understanding of a fragment of EasyChair. We restrict ***EC*** to a single conference system. These conventions can be adopted to model other web-based systems.

System policy as set of read and write actions. Using *X-Policy*, we specify ***EC*** as *X-Policy* actions which can be either *write actions* that change the state of the system or *read actions* that give the user/agent knowledge about the state of the system. A read action is allowed to retrieve the value of a single model variable. Actions cannot read and change the state of the system at the same time. In EasyChair, there are some cases in which a mix of read and write operations are executed in a single request. When a PC member requests an agent to subreview a paper and before the agent accepts

or rejects the subreviewing request, the agent can read the submission[1]. The fact that the agent has read the submission is recorded. This case is modelled in ***EC*** as two separate actions, reading the submission and recording the read. We link the two actions by allowing the sub-reviewer to read the submission if she recorded the read in advance. This is a sensible heuristic for modelling web-based systems. We also restrict our model to the system states and do not consider any possible logging system as part of our model.

System read operations that return multiple system values. We model each of these operations W as a set of actions $W_1, \ldots, W_n$. Each action W_i returns one of the values returned by W. The execution rights of W_i are the same as W. For example: EasyChair operation ShowReviews(p) which will return all the reviews on the paper p is modelled as the set of the read actions ShowReview(p,a_1)... ShowReview(p,a_n) which returns the review of agent $a_1 \ldots a_n$ on paper p.

Modelling EasyChair "log in as another pc member" functionality. In EasyChair, the system allows the PC chair to act on behalf of another PC member using "log in as another pc member". For example a PC chair can submit a review for a paper assigned for another PC member to review. The actions executed on the PC member's behalf are indistinguishable from the ones that are executed by the PC member herself. Nevertheless, EasyChair restricts the PC chair from changing the PC member account details or accessing/editing her sub-reviewing requests. We model these actions in ***EC*** by conjoining the conditions agent u has to satisfy to act as another PC member - in this case being a chair - and the conditions that the PC member has to satisfy to execute that action. One might also consider using relations like "CanActAs", as in [2,11]. However, when we say A CanActAs B then we mean that A is capable of performing all the actions that B can perform which is not applicable in this case.

Intermediate condition. In some cases, the system checks some intermediate conditions during an update operation like validation conditions or maintenance conditions to preserve an integrity constraint. For example, EasyChair insures that a conflict of interest is respected when a chair assigns reviewers to a paper. We express these intermediate conditions as execution preconditions. Where the checking operations reveal a system value by an error message, this value is readable by the agent requesting the operation.

Conference configuration settings. We model the conference configuration settings as 0-ary atomic formulae. The value of these settings affects the conference permissions globally. In specifying the system execution policy if the user can learn about the system configuration settings from the behaviour of the system even though she cannot read the settings directly, we consider her to be able to read that variable. In some cases the user might learn partial information about a single configuration variable. For example when the list of submissions can only be viewed by PC chairs only or nobody, the PC member learns that she is not allowed to see the list of the submissions but she cannot distinguish between the two possibilities. We model this case in ***EC*** by designating a variable that represents the fact that the PC member can read the list of submissions. The PC-member can infer the value of that variable by using the system.

[1] This is an intermediate step before deciding to accept or reject the subreviewing assignment. We decided not to include it in the case study, for brevity, but it is available in the full model[22].

4.2 *EC* model in *X-Policy* formalism

In this Section we express the ***EC*** model in *X-Policy* formalism. We define $T = \{\mathsf{Agent}, \mathsf{Paper}\}$. To model relations between these two types, we need a number of predicates, P, as follows:

For a,b of type Agent, p of type Paper, P includes:

Chair-review-en()	Review menu is enabled for Chair. It enables Chair(s) to manage the reviews of submitted papers.
Chair-status-en()	Status menu is enabled for Chair. It enables Chair(s) to manage the status of submitted papers.
PCM-access-reviews-en()	PC members can access (view) other papers reviews.
PCM-review-editing-en()	PC members can add/modify reviews.
PCM-review-menu-en()	Review menu is enabled for PC members. It enables PC members to manage paper reviews.
PCM-status-en()	Status menu is enabled for PC members. It enables PC members to manage paper status.
Review-assig-enabled()	Review assignments enabled.
Show-reviewer-name()	Reviewer's name is readable by other PC members.
Sub-anonymous()	Submissions are anonymous. The name of authors are obscured.
Sub-open()	Submission system is open and accepts new papers.
View-sub-by-chair-permitted()	PC chairs can view the list of submissions.
View-sub-by-PCM-permitted()	PC members can view the list of submissions.
View-sub-title-permitted()	PC members can view the submission title for papers not assigned to them.
View-sub-txt-permitted()	PC members can view the submission text for papers not assigned to them.
Author(p,a)	Agent a is an author of paper p.
Chair(a)	Agent a is the chair of the PC.
Conf-of-interest(p,a)	Agent a has a conflict of interest with the paper p.
Decided-subrev(p,a,b)	Agent b has decided whether to accept or reject the subreviewing request for paper p issued by agent a.
PCmember(a)	Agent a is a PC member.
Requested-subrev(p,a,b)	Agent a has requested agent b to be his subreviewer for paper p.
Reviewer(p,a)	Paper p is assigned to PC member a for reviewing.
Submitted-review(p,a,b)	Agent b's review of Paper p has been submitted by agent a.
Subreviewer(p,a,b)	Agent b has accepted the subreviewing request for paper p issued by agent a.
Updated-review(p,a,b)	Agent b's review of Paper p has been updated by PC member a.

We now define the set of actions Actions and their execution permissions using the formula exec(act, u) for each action act $\in$ Actions. The execution permission statements define whether or not u of type Agent is allowed to execute such an action and in what state. In the following, we list a sub-set of ***EC*** actions and their permission statements which are used in our properties analysis in *X-Policy*:

1. When the review menu is enabled and the submitted paper is not deleted: **(a)** A PC chair can read all the paper reviews. **(b)** A PC member can read a review for a paper p if she is a reviewer of that paper and has submitted her review. **(c)** A PC member can read a review for a paper to which she is not assigned, when PC members are permitted to access the titles and reviews of submitted papers. She also must have no conflict of interest with that paper.

```
Action ShowReview(p,a,b):-
{
        return Submitted-review(p,a,b);
}
```

$$\text{exec}(\mathsf{ShowReview(p,a,b)},\mathsf{u}) \rightleftharpoons \left(\begin{array}{c} \left(\begin{array}{c}\mathsf{Chair(u)} \wedge \mathsf{Chair\text{-}review\text{-}en()}\\ \wedge\exists \mathsf{d} : \mathsf{Agent}\ .\ \mathsf{Author(p,d)}\end{array}\right) \\ \vee \left(\begin{array}{c}\mathsf{PCmember(u)} \wedge \mathsf{Reviewer(p,u)}\\ \wedge\mathsf{PCM\text{-}review\text{-}menu\text{-}en()}\\ \wedge\exists\ \mathsf{c} : \mathsf{Agent}\ .\ \mathsf{Submitted\text{-}review(p,u,c)}\\ \wedge\exists \mathsf{d} : \mathsf{Agent}\ .\ \mathsf{Author(p,d)}\end{array}\right) \\ \vee \left(\begin{array}{c}\mathsf{PCmember(u)} \wedge \neg\mathsf{Reviewer(p,u)}\\ \wedge\mathsf{PCM\text{-}review\text{-}menu\text{-}en()}\\ \wedge\mathsf{View\text{-}sub\text{-}title\text{-}permitted()}\\ \wedge\mathsf{PCM\text{-}access\text{-}reviews\text{-}en()}\\ \wedge\neg\mathsf{Conf\text{-}of\text{-}interest(p,u)}\\ \wedge\exists \mathsf{d} : \mathsf{Agent}\ .\ \mathsf{Author(p,d)}\end{array}\right)\end{array}\right)$$

2. When the review menu is enabled and the submitted paper is not deleted: **(a)** A PC chair can submit a review for any paper as himself. **(b)** A PC chair can submit a review for a paper as another PC member using "log in as another pc member" if the PC member is allowed to submit a review for that paper. **(c)** A PC member can review a paper if she is assigned to review that paper. **(d)** A PC member can review a paper to which she is not assigned when PC members are permitted to access the titles and reviews of submitted papers. She also must have no conflict of interest with that paper.

Action AddReview(p,a,b):-
{
Submitted-review(p,a,b):= $\top$;
}

$$\text{exec}(\mathsf{AddReview(p,a,b)},\mathsf{u})) \rightleftharpoons \left(\begin{array}{c} \left(\begin{array}{c}\mathsf{Chair(u)} \wedge \mathsf{Chair\text{-}review\text{-}en()}\\ \wedge\mathsf{a} = \mathsf{u} \wedge \exists \mathsf{d} : \mathsf{Agent}\ .\ \mathsf{Author(p,d)}\end{array}\right) \\ \vee \left(\begin{array}{c}(\mathsf{a} = \mathsf{u} \vee \mathsf{Chair(u)}) \wedge \exists \mathsf{c} : \mathsf{Agent}\ .\ \mathsf{Author(p,c)}\\ \wedge \left(\begin{array}{c}\left(\begin{array}{c}\mathsf{PCmember(a)} \wedge \mathsf{Reviewer(p,a)}\\ \wedge\mathsf{PCM\text{-}review\text{-}menu\text{-}en()}\\ \wedge\mathsf{PCM\text{-}review\text{-}editing\text{-}en()}\end{array}\right)\\ \wedge \left(\begin{array}{c}\mathsf{PCmember(a)} \wedge \neg\mathsf{Reviewer(p,a)}\\ \wedge\mathsf{PCM\text{-}review\text{-}menu\text{-}en()}\\ \vee \left(\begin{array}{c}\mathsf{View\text{-}sub\text{-}title\text{-}permitted()}\\ \wedge\mathsf{PCM\text{-}access\text{-}reviews\text{-}en()}\\ \wedge\neg\mathsf{Conf\text{-}of\text{-}interest(p,a)}\end{array}\right)\end{array}\right)\end{array}\right)\end{array}\right)$$

3. Given that paper assignments are enabled, a PC chair can assign/de-assign a submitted paper to a PC member or a PC chair for reviewing, when she has no conflict of interest with that paper.

Action AddReviewerAssignment(p,a):-
{
Reviewer(p,a) := $\top$;
}

$$\text{exec}(\mathsf{AddReviewerAssignment(p,a)},\mathsf{u}) \rightleftharpoons \left(\begin{array}{c}\mathsf{Chair(u)} \wedge (\mathsf{PCmember(a)} \vee \mathsf{Chair(a)})\\ \wedge\mathsf{Review\text{-}assig\text{-}enabled()}\\ \wedge\neg\mathsf{Conf\text{-}of\text{-}interest(p,a)}\\ \wedge\exists \mathsf{c} : \mathsf{Agent}\ .\ \mathsf{Author(p,c)}\end{array}\right)$$

4. When the review menu is enabled and the submitted paper is not deleted: **(a)** A PC chair can request another agent to subreview any paper. **(b)** A PC member can invite another agent to subreview a paper: (1) if she is the reviewer of the paper or (2) if the system is configured to give PC members access to the paper submission titles and reviews. The invited agent can decide

whether to accept or reject the reviewing request as long as the paper has not been withdrawn. A PC member cannot cancel the subreviewing request but can accept or reject the request on behalf of the invited agent. Once the decision is made, only the PC member can change the decision.

Action RequestReviewing(p,a,b):-
{
Requested-subrev(p,a,b):= $\top$;
}

Action AcceptReviewingRequest(p,a,b):-
{
Decided-subrev(p,a,b):=$\top$;
Subreviewer(p,a,b):=$\top$;
}

Action RejectReviewingRequest(p,a,b):-
{
Decided-subrev(p,a,b):=$\top$;
Subreviewer(p,a,b):=$\bot$;
}

$$\text{exec}(\text{RequestReviewing}(p,a,b),u) \rightleftharpoons \left(\begin{array}{l} \left(\begin{array}{c} \text{Chair-review-en}() \wedge \text{Chair}(u) \\ \wedge \exists c : \text{Agent} \ . \ \text{Author}(p,c) \end{array} \right) \\ \vee \left(\begin{array}{c} \text{PCmember}(u) \wedge \text{Reviewer}(p,u) \\ \wedge \text{PCM-review-menu-en}() \\ \wedge \exists c : \text{Agent} \ . \ \text{Author}(p,c) \end{array} \right) \\ \vee \left(\begin{array}{c} \text{PCmember}(u) \wedge \neg \text{Reviewer}(p,u) \\ \wedge \text{PCM-review-menu-en}() \\ \wedge \text{View-sub-title-permitted}() \\ \wedge \text{PCM-access-reviews-en}() \\ \wedge \neg \text{Conf-of-interest}(p,u) \\ \wedge \exists c : \text{Agent} \ . \ \text{Author}(p,c) \end{array} \right) \end{array} \right)$$

$$\text{exec}(\text{AcceptReviewingRequest}(p,a,b),u) \rightleftharpoons \left(\begin{array}{c} \text{Requested-subrev}(p,a,b) \\ \wedge \exists c : \text{Agent} \ . \ \text{Author}(p,c) \\ \wedge \left(\neg \text{Decided-subrev}(p,a,u) \vee u = a \right) \end{array} \right)$$

$$\text{exec}(\text{RejectReviewingRequest}(p,a,b),u) \rightleftharpoons \left(\begin{array}{c} \text{Requested-subrev}(p,a,b) \\ \wedge \exists c : \text{Agent} \ . \ \text{Author}(p,c) \\ \wedge \left(\neg \text{Decided-subrev}(p,a,u) \vee u = a \right) \end{array} \right)$$

We include the full model of ***EC*** at [22].

4.3 Case Study: Analysis of *EC* security properties

In this Section, we will discuss a number of security issues in ***EC***. We have discovered these issues while using EasyChair. In each case, we show an attack strategy to achieve an undesirable state. Each strategy is an execution sequence of read and write actions which takes the model from an initial state m_0 to a goal state m_g. A strategy can be executed by more than one agent where agents collaborate to reach the goal. We show that these strategies work on our model and reach the goal state. We have also tried these attacks on EasyChair and they have succeeded as of the 1^{st} of September 2009. In the following, we report the results of each attack and make some suggestions on how the system could fix these issues.

To analyse the system, we have to define a number of individuals of types Agent and Paper and use these individuals to define an initial state which we refer to as m_0. For our ***EC*** model, we create the following configuration:

1. The system has five agents: Alice, Bob, Eve, Carol and Marvin. The system has two submitted papers: p1 and p2. We express the configuration as following: $\sigma_{Paper} = \{\mathsf{p1}, \mathsf{p2}\}$ and $\sigma_{Agent} = \{$Alice, Bob, Carol, Eve, Marvin$\}$.
2. Alice is the Chair of PC. Bob and Carol are PC members. Paper p1 is submitted by the author Marvin while p2 is submitted by the author Eve. Reviewers' names are obscured from each other by enabling the anonymous reviewing option. Authors' names are obscured from the PC members and the reviewers. The conference submission is configured in the anonymous submission mode. The list of submissions can be viewed by PC chairs only. Non-chairs do not have access to reviews of papers not assigned to them. In this case, we choose an up-and-running state of ***EC*** to keep our proof to a minimum. However, we can derive the system from an earlier state. We express these settings in the following *X-Policy* configuration:

   $m_0 = \{$ Chair(Alice), PCmember(Bob), PCmember(Carol), Author(p1,Marvin), Author(p2,Eve), PCM-review-editing-en(), View-sub-by-chair-permitted(), Chair-review-en(), PCM-review-menu-en(), Sub-anonymous(),Review-assig-enabled()$\}$

   Now that we have defined the initial state m_0, we can analyse the following properties. Each of these properties will start from m_0 and derive the model using a strategy S_i to reach the goal state $m_g^{S_i}$.

Property 1: A single subreviewer should not be able to determine the outcome of a paper reviewing process by writing two reviews of the same paper. We show that we can derive an attack against ***EC*** involving 4 agents: Alice, Bob, Carol, and Eve. We explain the attack scenario as a sequence of actions executed by these agents as follows:

1. Alice acts as chair. She executes the actions: AddReviewerAssignment(p1,Bob) to assign Bob to review the paper p1. She also executes AddReviewerAssignment(p1,Carol) to assign Carol to review the paper p1.
2. Bob and Carol both assign Eve as their sub-reviewer for paper p1 by executing the actions RequestReviewing(p1,Bob,Eve) and RequestReviewing(p1,Carol,Eve) respectively.
3. Eve accepts the two paper subreviewing requests. Eve then sends Bob and Carol two similar reviews using AcceptReviewingRequest(p1,Carol,Eve) and AcceptReviewingRequest(p1,Bob,Eve).
4. Bob and Carol receive Eve's reviews and submit them to the system using AddReview(p1,Bob,Eve) and AddReview(p1,Carol,Eve).

Note that the names of authors and other reviewers are not known to the PC members. While we show how Eve can write two reviews of the paper, the attack can be exploited in the same way to enable Eve to write all three reviews. The detailed derivation for this attack on property 1 can be found in Appendix A

One possible fix for this attack is as follows. Every time an agent a invites another agent b to subreview a paper, EasyChair should check whether agent b has been invited by another agent to subreview the same paper. We conjoin the condition $\neg\exists$ d : Agent . Requested-subrev(p, d, b) to the permission statement exec(RequestReviewing(p,a,b),u). In this case Carol cannot execute RequestReviewing(p1,Carol,Eve) as Requested-subrev(p1,Bob,Eve) is in the previous state.

Property 2: A paper author should not review her own paper. As before, we explain the attack scenario as a sequence of actions executed by the agents Alice, Bob and Eve:

1. Alice acts as Chair and assigns Bob, who is a PC member, to review the paper p2 submitted by Eve by executing the action AddReviewerAssignment(p2,Bob).
2. Bob executes the action RequestReviewing(p2,Bob,Eve) to assign Eve as his sub-reviewer as she is a good researcher in the field.
3. Eve accepts the request using AcceptReviewingRequest(p2,Bob,Eve).
4. Bob submits the review using AddReview(p2,Bob,Eve).

Note that the names of authors and other reviewers are not known to the PC members. One possible fix for this attack is that every time an agent a invites another agent b to subreview a paper, EasyChair should check whether agent b is actually an author of that paper. We add the condition $\neg$Author(p, a) to the permission statement exec(RequestReviewing(p,a,b),u). In this case Bob cannot execute RequestReviewing(p2,Bob,Eve) as Author(p2,Eve) is in $m_1^{S_2}$. The detailed derivation for this attack on property 2 can be found in Appendix B

Property 3: Users should be accountable for their actions. This property is violated in several ways, all of which involve the use of "log in as another pc member". For example, the system should not allow the chair to submit a review for a paper as another PC member without making it clear that it is actually the chair who has submitted the review and not the PC member. The following attack scenario involves Alice and Bob:

1. Alice is the chair. She executes AddReviewerAssignment(p1,Bob) to assign Bob to review the paper p1.
2. Bob submits his review using AddReview(p1,Bob,Bob).
3. Alice reads Bob's review of paper p1 by executing the action ShowReview(p1,Bob,Bob).
4. Alice submits a review for the paper p1 as if she is Carol who is a very famous and sought after academic by executing AddReview(p1,Carol,Carol).

EasyChair fails this property and allows the chair to read another reviewer's review for a paper and then submits a review for that paper as another PC member without being detected by the other PC members or the other chairs. This attack is possible because the system does not register the name of the user who updated the review. It will appear to others as if Carol has submitted the review herself. One possible fix for this attack is for AddReview() to have an additional parameter. Alice would then need to execute the action AddReview(p,a,b,c) where agent a is the chair acting on behalf of b who is the PCmember submitting the review written by agent c. The predicate Submitted-review() also has to be changed accordingly. In this case, if a chair submits a review of a paper the system highlights the fact that the chair is actually the one submitting the review. The detailed derivation for this attack on property 3 can be found in Appendix C

5 Conclusion and Future Work

In this paper we present a modelling language, *X-Policy*, to model the dynamic execution permissions of large web-based collaborative systems. We demonstrate the applicability of *X-Policy* to real-life web-based collaborative systems like EasyChair. We propose a number of modelling conventions for EasyChair which can be applied to other web-based systems. The full ***EC*** model is available

at [22]. It contains 49 actions and permission statements. This is relatively concise given the size and complexity of EasyChair. The way the system functionality is split into actions is decided by our understanding of how the system is actually designed. The ability to specify multi-assignment actions enables us to maintain the integrity constraint so that, for example, when a PC-member is deleted, her reviewing assignments are also deleted. Using *X-Policy*, we can reason about the security properties of our model. We presented a case study of three security properties for EasyChair and described the possible attacks on these properties as well as ways the system could be changed to prevent these attacks. We have informed the developer of EasyChair of our findings.

The motivation behind *X-Policy* is to provide a simple modelling language giving policy designers and system managers the ability to specify collaborative system policies like those discussed in the paper, and to reason about their real-life security properties. *X-Policy*, with its ability to establish the relation between an action and the agent who is executing it, allows us to analyse security properties that require collaboration between a specific set of agents who are allowed to act to achieve an attack on the system. *X-Policy*'s ability to specify read and write actions also allows us to reason about pieces of data being read as part of the attack. These two features distinguish *X-Policy* from other dynamic policy specification languages like DyNPAL.

A direct comparison between the expressiveness of *X-Policy* and Datalog-based languages like SecPAL, Cassandra and others is inappropriate as the latter are designed to solve a different set of problems, mainly relating to authorisation in decentralised settings.

In future work, we would like to model and analyse more systems. The reasoning about security properties in this paper was performed manually. We plan to develop and implement an algorithm to automate the analysis of these systems using model checking techniques. Our choice of finite sets for each type is motivated by our desire to design such a tool. Such a restriction is common in reachability analysis in access control systems[4,5,9,24]. We also plan to design a query language that expresses high-level security properties such as those discussed in our case study. While these properties can be expressed as reachability goals, other high-level properties, like property no. 3, require the ability to prove the observational equivalence between two strategies. For example, we should be able to query whether actions committed by the chair acting as a PC member are indistinguishable from actions committed by the PC member herself. Such a query requires the ability to reason about the value and the readability of state variables during the strategy execution.

References

1. Thomas Baignères and Matthieu Finiasz. iChair conference management system. `http://lasecwww.epfl.ch/iChair/`.
2. Moritz Becker, Cedric Fournet, and Andrew Gordon. Design and semantics of a decentralized authorization language. In *Computer Security Foundations Symposium, 2007. CSF '07. 20th IEEE*, pages 3–15, 2007.
3. Moritz Y. Becker. *Cassandra: flexible trust management and its application to electronic health records.* PhD thesis, Computer Laboratory, University of Cambridge, 2005.
4. Moritz Y. Becker. Specification and analysis of dynamic authorisation policies. *Computer Security Foundations Symposium, IEEE*, 0:203–217, 2009.
5. Moritz Y. Becker and Sebastian Nanz. A logic for state-modifying authorization policies. In *European Symposium on Research in Computer Security*, 2007.
6. Moritz Y. Becker and Peter Sewell. Cassandra: distributed access control policies with tunable expressiveness. 5th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY), 2004.
7. Moritz Y. Becker and Peter Sewell. Cassandra: flexible trust management, applied to electronic health records. 17th IEEE Computer Security Foundations Workshop (CSFW), 2004.

8. John DeTreville. Binder, a logic-based security language. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, 2002.
9. Kathi Fisler, Shriram Krishnamurthi, Leo A. Meyerovich, and Michael Carl Tschantz. Verification and change-impact analysis of access-control policies. In *ICSE'05*, St. Louis, Missouri, USA, May 2005.
10. Yuri Gurevich and Itay Neeman. DKAL: Distributed-knowledge authorization language. In *CSF '08: Proceedings of the 2008 21st IEEE Computer Security Foundations Symposium*, volume 0, pages 149–162, Washington, DC, USA, 2008. IEEE Computer Society.
11. Yuri Gurevich and Itay Neeman. DKAL 2 : A simplified and improved authorization language. Technical report, Microsoft Research - Cambridge, 2009.
12. Shai Halevi. Sourceforge.net: Web submission and review software. Available online at `http://sourceforge.net/projects/websubrev`.
13. Eddie Kohler. HotCRP conference management software. `http://www.cs.ucla.edu/~kohler/hotcrp/index.html`.
14. N. Li. *Delegation Logic: A Logic-based Approach to Distributed Authorization*. PhD thesis, New York University, New York, Sep 2000.
15. Ninghui Li and John C. Mitchell. Datalog with constraints: A foundation for trust management languages. In *Proceedings of the Fifth International Symposium on Practical Aspects of Declarative Languages*, January 2003.
16. Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust management framework. In *Proc. IEEE Symposium on Security and Privacy, Oakland*, May 2002.
17. Ninghui Li, John C. Mitchell, William H. Winsborough, Kent E. Seamons, Michael Halcrow, and Jared Jacobson. RTML: a role-based trust-management markup language. Technical report, Purdue University, 2004. CERIAS TR 2004-03.
18. Ninghui Li, William H. Winsborough, and John C. Mitchell. Distributed credential chain discovery in trust management: extended abstract. In *ACM Conference on Computer and Communications Security*, pages 156–165, 2001.
19. Swee-Won Lo, Raphael C.-W. Phan, and Bok-Min Goi. On the Security of a Popular Web Submission and Review Software (WSaR) for Cryptology Conferences. In *WISA '07: the 8th International Workshop on Information Security Applications*, Lecture Notes in Computer Science. Springer, 2007.
20. D. McDermott and J. Doyle. Nonmonotonic logic 1. *Artificial Intelligence*, 13:41–72, 1980.
21. Raphael C.-W. Phan and Huo-Chong Ling. On the insecurity of the Microsoft Research Conference Management Tool (MSRCMT) system. In *CITA 2005: Proceedings of 4th International Conference on IT in Asia*, pages 75–79, 2005.
22. Hasan Qunoo and Mark Ryan. EC model in X-policy. online at `http://www.cs.bham.ac.uk/~hxq/X-policy/`, Dec 2009.
23. Andrei Voronkov. EasyChair conference system. `http://www.easychair.org/`.
24. Nan Zhang, Mark Ryan, and Dimitar P. Guelev. Synthesising verified access control systems in XACML. In *2004 ACM Workshop on Formal Methods in Security Engineering*, pages 56–65, Washington DC, USA, Oct 2004. ACM Press.

A Proof for the attack on property 1

In the following, we use Alice:AddReviewerAssignment(p1,Bob) to denote that the agent Alice executes the action AddReviewerAssignment(p1,Bob). Starting from m_0 we show how the model state evolves through the attack strategy. At each step, we explicitly list, when appropriate, the list of ground atomic formulae that has to be absent or present to execute the following action from the model state:

Alice:AddReviewerAssignment(p1,Bob);
requires the presence of Chair(Alice), PCmember(Bob), Review-assig-enabled().
requires the absence of Conf-of-interest(p1,Bob).
$m_1^{S_1}$ = {Chair(Alice), PCmember(Bob), PCmember(Carol), Author(p1,Marvin), Sub-anonymous(), Author(p2,Eve), PCM-review-editing-en(), View-sub-by-chair-permitted(), Chair-review-en(), PCM-review-menu-en(), Review-assig-enabled(), Reviewer(p1,Bob)}
Alice:AddReviewerAssignment(p1,Carol);

requires the presence of Chair(Alice), PCmember(Carol), Review-assig-enabled().
requires the absence of Conf-of-interest(p1,Carol).

$m_2^{S_1}$ = {Chair(Alice), PCmember(Bob), PCmember(Carol), Author(p1,Marvin), Sub-anonymous(), Author(p2,Eve), PCM-review-editing-en(), View-sub-by-chair-permitted(), Chair-review-en(), PCM-review-menu-en(), Review-assig-enabled(), Reviewer(p1,Bob), Reviewer(p1,Carol)}

Bob:RequestReviewing(p1,Bob,Eve);

requires the presence of PCmember(Bob), Reviewer(p1,Bob), PCM-review-menu-en().

$m_3^{S_1}$ = {Chair(Alice), PCmember(Bob), PCmember(Carol), Author(p1,Marvin), Sub-anonymous(), Author(p2,Eve), PCM-review-editing-en(), View-sub-by-chair-permitted(), Chair-review-en(), PCM-review-menu-en(), Review-assig-enabled(), Reviewer(p1,Bob), Reviewer(p1,Carol), Requested-subrev(p1,Bob,Eve)}

Carol:RequestReviewing(p1,Carol,Eve);

requires the presence of PCmember(Carol), Reviewer(p1,Carol), PCM-review-menu-en().

$m_4^{S_1}$ = {Chair(Alice), PCmember(Bob), PCmember(Carol), Author(p1,Marvin), Sub-anonymous(), Author(p2,Eve), PCM-review-editing-en(), View-sub-by-chair-permitted(), Chair-review-en(), PCM-review-menu-en(), Review-assig-enabled(), Reviewer(p1,Bob), Reviewer(p1,Carol), Requested-subrev(p1,Bob,Eve), Requested-subrev(p1,Carol,Eve)}

Eve:AcceptReviewingRequest(p1,Bob,Eve);

requires the presence of Requested-subrev(p1,Bob,Eve).

$m_5^{S_1}$ = {Chair(Alice), PCmember(Bob), PCmember(Carol), Author(p1,Marvin), Sub-anonymous(), Author(p2,Eve), PCM-review-editing-en(), View-sub-by-chair-permitted(), Chair-review-en(), PCM-review-menu-en(), Review-assig-enabled(), Reviewer(p1,Bob), Reviewer(p1,Carol), Requested-subrev(p1,Bob,Eve), Requested-subrev(p1,Carol,Eve), Decided-subrev(p1,Bob,Eve), Subreviewer(p1,Bob,Eve)}

Eve:AcceptReviewingRequest(p1,Carol,Eve);

requires the presence of Requested-subrev(p1,Carol,Eve).

$m_6^{S_1}$ = {Chair(Alice), PCmember(Bob), PCmember(Carol), Author(p1,Marvin), Sub-anonymous(), Author(p2,Eve), PCM-review-editing-en(), View-sub-by-chair-permitted(), Chair-review-en(), PCM-review-menu-en(), Review-assig-enabled(), Reviewer(p1,Bob), Reviewer(p1,Carol), Requested-subrev(p1,Bob,Eve), Requested-subrev(p1,Carol,Eve), Decided-subrev(p1,Bob,Eve), Subreviewer(p1,Bob,Eve), Decided-subrev(p1,Carol,Eve), Subreviewer(p1,Carol,Eve)}

Bob:AddReview(p1,Bob,Eve);

requires the presence of PCmember(Bob), Reviewer(p1,Bob), PCM-review-menu-en(), PCM-review-editing-en().

$m_7^{S_1}$ = {Chair(Alice), PCmember(Bob), PCmember(Carol), Author(p1,Marvin), Sub-anonymous(), Author(p2,Eve), PCM-review-editing-en(), View-sub-by-chair-permitted(), Chair-review-en(), PCM-review-menu-en(), Review-assig-enabled(), Reviewer(p1,Bob), Reviewer(p1,Carol), Requested-subrev(p1,Bob,Eve), Requested-subrev(p1,Carol,Eve), Decided-subrev(p1,Bob,Eve), Subreviewer(p1,Bob,Eve), Decided-subrev(p1,Carol,Eve), Subreviewer(p1,Carol,Eve), Submitted-review(p1,Bob,Eve)}

Carol:AddReview(p1,Carol,Eve);

requires the presence of PCmember(Carol), Reviewer(p1,Carol),PCM-review-menu-en(), PCM-review-editing-en().

$m_8^{S_1}$ = {Chair(Alice), PCmember(Bob), PCmember(Carol), Author(p1,Marvin), Sub-anonymous(), Author(p2,Eve), PCM-review-editing-en(), View-sub-by-chair-permitted(), Chair-review-en(), PCM-review-menu-en(), Review-assig-enabled(), Reviewer(p1,Bob), Reviewer(p1,Carol), Requested-subrev(p1,Bob,Eve), Requested-subrev(p1,Carol,Eve), Decided-subrev(p1,Bob,Eve), Subreviewer(p1,Bob,Eve), Decided-subrev(p1,Carol,Eve), Subreviewer(p1,Carol,Eve), Submitted-review(p1,Bob,Eve), Submitted-review(p1,Carol,Eve)}

In this case the model state m_o has evolved during the scenario to the goal state $m_8^{S_1}$. The ground atomic formulae Submitted-review(p1,Bob,Eve) and Submitted-review(p1,Carol,Eve) are in $m_8^{S_1}$. This means that Eve has managed to write two reviews for the same paper and get them submitted to

the system. Similarly, Eve could have written all the reviews of that particular paper. Consequently, EasyChair fails the property as a single reviewer can determine the outcome of a paper.

B Proof for the attack on property 2

We now show how the model state evolves through the attack strategy. At each step, we explicitly list, when appropriate, the list of ground atomic formulae that has to be absent or present to execute the following action from the model state:

Alice:AddReviewerAssignment(p2,Bob);
requires the presence of Chair(Alice), PCmember(Bob), Review-assig-enabled().
requires the absence of Conf-of-interest(p2,Bob).
$m_1^{S_2}$ = {Chair(Alice), PCmember(Bob), PCmember(Carol), Author(p1,Marvin), Author(p2,Eve), Conf-of-interest(p1,Alice), PCM-review-editing-en(), View-sub-by-chair-permitted(), Chair-review-en(), PCM-review-menu-en(), Sub-anonymous(), Review-assig-enabled(), Reviewer(p2,Bob)}
Bob:RequestReviewing(p2,Bob,Eve);
requires the presence of PCmember(Bob), Reviewer(p2,Bob), PCM-review-menu-en().
$m_2^{S_2}$ = {Chair(Alice), PCmember(Bob), PCmember(Carol), Author(p1,Marvin), Author(p2,Eve), Conf-of-interest(p1,Alice), PCM-review-editing-en(), View-sub-by-chair-permitted(), Chair-review-en(), PCM-review-menu-en(), Sub-anonymous(), Review-assig-enabled(), Reviewer(p2,Bob), Requested-subrev(p2,Bob,Eve)}
Eve:AcceptReviewingRequest(p2,Bob,Eve);
requires the presence of PCmember(Bob), Reviewer(p2,Bob), PCM-review-menu-en().
$m_3^{S_2}$ = {Chair(Alice), PCmember(Bob), PCmember(Carol), Author(p1,Marvin), Author(p2,Eve), Conf-of-interest(p1,Alice), PCM-review-editing-en(), View-sub-by-chair-permitted(), Chair-review-en(), PCM-review-menu-en(), Sub-anonymous(), Review-assig-enabled(), Reviewer(p2,Bob), Decided-subrev(p2,Bob,Eve), Subreviewer(p2,Bob,Eve)}
Bob:AddReview(p2,Bob,Eve);
requires the presence of PCmember(Bob), Reviewer(p2,Bob), PCM-review-editing-en(), PCM-review-menu-en().
$m_4^{S_2}$ = {Chair(Alice), PCmember(Bob), PCmember(Carol), Author(p1,Marvin), Author(p2,Eve), Conf-of-interest(p1,Alice), PCM-review-editing-en(), View-sub-by-chair-permitted(), Chair-review-en(), PCM-review-menu-en(), Sub-anonymous(), Review-assig-enabled(), Reviewer(p2,Bob), Decided-subrev(p2,Bob,Eve), Subreviewer(p2,Bob,Eve), Submitted-review(p2,Bob,Eve)}

In this case the model has evolved to the goal state where ground atomic formula Submitted-review(p2,Bob,Eve) is in $m_4^{S_2}$. This means that Eve has managed to submit a review for her own paper p2. EasyChair fails this property as it allows a paper's reviewers to submit a review written by the paper's author herself.

C Proof for the attack on property 3

We show how the model state evolves through the attack strategy:

Alice:AddReviewerAssignment(p1,Bob);
requires the presence of Chair(Alice), PCmember(Bob)Review-assig-enabled().
requires the absence of Conf-of-interest(p1,Bob)
$m_1^{S_3}$ = {Chair(Alice), PCmember(Bob), PCmember(Carol), Author(p1,Marvin), Author(p2,Eve), Conf-of-interest(p1,Alice), PCM-review-editing-en(), View-sub-by-chair-permitted(), Chair-review-en(), PCM-review-menu-en(), Sub-anonymous(), Review-assig-enabled(), Reviewer(p1,Bob)}

Bob:AddReview(p1,Bob,Bob);
requires the presence of PCmember(Bob), Reviewer(p1,Bob), PCM-review-menu-en(), PCM-review-editing-en().

$m_2^{S_3}$ = {Chair(Alice), PCmember(Bob), PCmember(Carol), Author(p1,Marvin), Author(p2,Eve), Conf-of-interest(p1,Alice), PCM-review-editing-en(), View-sub-by-chair-permitted(), Chair-review-en(), PCM-review-menu-en(), Sub-anonymous(), Review-assig-enabled(), Reviewer(p1,Bob), Submitted-review(p1,Bob,Bob)}

Alice:ShowReview(p1,Bob,Bob);
requires the presence of Chair(Alice), Chair-review-en().

$m_3^{S_3}$ = {Chair(Alice), PCmember(Bob), PCmember(Carol), Author(p1,Marvin), Author(p2,Eve), Conf-of-interest(p1,Alice), PCM-review-editing-en(), View-sub-by-chair-permitted(), Chair-review-en(), PCM-review-menu-en(), Sub-anonymous(), Review-assig-enabled(), Reviewer(p1,Bob), Submitted-review(p1,Bob,Bob)}

Alice:AddReview(p1,Carol,Carol);
requires the presence of Chair(Alice), PCmember(Bob), Reviewer(p1,Carol), PCM-review-menu-en(), PCM-review-editing-en().

$m_4^{S_3}$ = {Chair(Alice), PCmember(Bob), PCmember(Carol), Author(p1,Marvin), Author(p2,Eve), Conf-of-interest(p1,Alice), PCM-review-editing-en(), View-sub-by-chair-permitted(), Chair-review-en(), PCM-review-menu-en(), Sub-anonymous(), Review-assig-enabled(), Reviewer(p1,Bob), Submitted-review(p1,Bob,Bob), Submitted-review(p1,Carol,Carol)}

We can see that $m_4^{S_3}$ is the goal state $m_g^{S_3}$ as chair Alice has managed to submit a review to the paper p1 as if she were the PC member Carol.