Formal Specification of Role-based Security Policies for Clinical Information Systems

Karsten Sohr
Department of Mathematics and Computer Science
Universität Bremen
Bibliothekstr. 1
28359 Bremen, Germany
sohr@tzi.de

Michael Drouineaud
Department of Mathematics and Computer Science
Universität Bremen
Bibliothekstr. 1
28359 Bremen, Germany
mdruid@tzi.de

Gail-Joon Ahn*
Department of Software and Information Systems
University of North Carolina at Charlotte
Charlotte, NC 28223, USA
gahn@uncc.edu

ABSTRACT

Many healthcare organizations have transited from their old and disparate business models based on ink and paper to a new, consolidated ones based on electronic patient records. There are significant demands on secure mechanisms for collaboration and data sharing among clinicians, patients and researchers through clinical information systems. In order to fulfil the high demands of data protection in such systems, we believe that access control policies play an important role to reduce the risks to confidentiality, integrity, and availability of medical data. In this paper, we attempt to formally specify access control policies in clinical information systems which are highly dynamic and complex environments. We leverage characteristics of temporal linear first-order logic to cope with dynamic access control policies in clinical information systems.

Categories and Subject Descriptors

H.2.7 [**Information Systems**]: Database Administration—*Security, Integrity, and Protection*

Keywords

Authorisation constraints, LTL, healthcare environments

1. INTRODUCTION

The widespread use of clinical information systems and specifically the introduction of electronic patient records (EPRs) have brought about concerns about the confidentiality, integrity, and availability of medical data. In EU countries there are strong data protection laws that are based upon the principle of patient consent [6]. Access to patient data is only allowed if the patient has given his consent before. In paper-based systems the main threat was that an adversary breaks into a physician's office and fetches the patient records. With the advent of the EPR, this situation changed: the adversary could conveniently obtain the EPR starting his attack from a distant computer that need not even belong to the hospital at all. Nevertheless, it is important to note that the attacker might be rather an insider than a hacker from the outside.

Lack of privacy can be damaging to both the patient and the organisation concerned. This situation may be even worse when smartcards containing information about drug prescriptions or parts of the EPR will be introduced as intended by the German government [9]. Specifically, in this case there should be adequate measures to prevent unauthorised persons (like health insurers or employers) from reading certain patient data.

As argued elsewhere [23, 7] role-based access control (RBAC) is a promising technology for managing and enforcing security in large-scale distributed systems. Most clinical information systems now support RBAC concepts, but these mechanisms are mostly rudimentary such that advanced concepts like role hierarchies and authorisation constraints are still missing.

Moreover, often a centralised approach for data sharing is implemented where a security officer has the responsibility to manage sharing of sensitive information. However, a decentralised approach is more suitable where each clinician decides on her own to share her data with other clinicians without directly involving the security officer. Thus delegation of access rights is an important concept in the healthcare domain. As a consequence, we must deal with highly dynamic security policies for access control in clinical information systems.

In [25] a role-based architecture for delegation in clinical information systems has been described. Other forms of authorisation constraints relevant to the healthcare domain

*This work of Gail-J. Ahn was partially supported at the Laboratory of Information of Integration, Security and Privacy at the University of North Carolina at Charlotte by the grants from National Science Foundation (NSF-IIS-0242393) and Department of Energy Early Career Principal Investigator Award (DE-FG02-03ER25565)

SAC'05, March 13-17, 2005, Santa Fe, New Mexico, USA

are context constraints [11], temporal constraints [3] and workflow constraints [4]. Through the *combination* of different constraints, complex security policies can be formulated, specifically when dynamic security policies are considered. Often it is not clear what the consequences of these policies are. Hence the topic of this paper is the formal specification of dynamic security policies for clinical information systems. Based on these specifications, a formal verification of certain properties of these policies can be done later.

The authors consider first-order linear temporal logic (LTL) [14] a good formalism for specifying dynamic behaviour of security policies consisting of delegation constraints or constraints which mandate a certain order of task execution as needed for workflows. Furthermore, first-order LTL has been intensively studied in the literature [17, 14] and comes with standard tools like the Stanford prover [16].

The paper is organised as follows: Section 2 gives a short overview of RBAC and first-order LTL. In section 3 typical authorisation constraints for clinical information systems are discussed and formally specified in first-order LTL. Section 4 presents two example security policies for typical clinical business processes specified in first-order LTL and Section 5 summarises and gives an outlook on future work.

2. RBAC AND FIRST-ORDER LTL

We first describe role-based access control (RBAC) as a promising technology to efficiently deal with access control, introduce first-order LTL and show subsequently how RBAC security policies can be elegantly specified in first-order LTL.

2.1 RBAC

RBAC has received considerable attention as an alternative to traditional discretionary and mandatory access control. The explicit representation of roles greatly simplifies the security management and makes possible to use well-known security principles like separation of duty and least privilege [23]. Furthermore, an RBAC standard has been proposed [8], which is based on the RBAC96 model introduced by Sandhu et al. [23].

The RBAC96 model has the following basic components:

- Users, Roles, P, S (sets of users, roles, permissions, activated sessions)
- $UA \subseteq Users \times Roles$ (user assignment)
- $PA \subseteq Roles \times P$ (permission assignment)

Users may activate a subset of the roles they are assigned to in a *session*. P is the set of ordered pairs of operations and objects. In the context of security and access control all resources accessible in an IT-system (e.g., files, database tables, etc.) are named by the notion *object*. An *operation* is an active process applicable to objects (e.g., read, write, append, etc.). The relation PA assigns to each role a subset of P. So PA determines for each role the operation(s) it may execute and the object(s) to which the operation in question is applicable for the given role. Thus any user having assumed this role can apply an operation to an object if the corresponding ordered pair is an element of the subset assigned by PA to the role.

2.2 Specification of RBAC in First-order LTL

We now introduce *temporal* RBAC as a formalism that combines RBAC with states [18]. This allows for talking about things such as the execution history or order of executions, as in [13], while still being much simpler than [13]. Temporal-logic RBAC is based on temporal first-order logic, a logic that has been intensively studied in the literature [17, 14].

spec TemporalRBAC =
sorts $Users, Sessions, Roles, Operations, Objects$
rigid op $user : Sessions \rightarrow Users$;
flexible preds $UA : Users \times Roles$;
$PA : Operations \times Objects \times Roles$;
$\_active\_in\_\_ : Roles \times Sessions$;
$exec : Sessions \times Operations \times Objects$
forall $r : Roles$; $s : Sessions$; $op : Operations$; $obj : Objects$
- $(\Diamond r\ active\_in\ s) \Rightarrow UA(user(s), r)$
- $\Box(exec(s, op, obj) \Rightarrow \exists r : Roles\,.\ r\ active\_in\ s \wedge PA(op, obj, r))$
- $\Box(auth(u, op, obj) \Rightarrow \exists s : Sessions;\ r : Roles\,.\ user(s) = u \wedge r\ active\_in\ s \wedge PA(op, obj, r))$

end

spec RichTemporalRBAC = TemporalRBAC **then**
flexible preds $\_active\_for\_\_ : Roles \times Users$;
$exec : Users \times Operations \times Objects$
forall $r : Roles$; $u : Users$; $op : Operations$; $obj : Objects$
- $\Box(r\ active\_for\ u \Leftrightarrow \exists s : Sessions\,.\ user(s) = u \wedge r\ active\_in\ s)$
- $\Box(exec(u, op, obj) \Leftrightarrow \exists s : Sessions\,.\ user(s) = u \wedge exec(s, op, obj))$

end

Figure 1: Temporal-logic RBAC, formalized within temporal first-order logic.

A temporal first-order signature consists of a set of sorts, a set of function symbols and a set of predicate symbols (each symbol coming with a string of argument sorts and, for function symbols, a result sort). Function and predicate symbols are partitioned into *rigid* and *flexible* symbols: the former do not change over time, while the latter may vary.

Sentences are the usual first-order sentences built from equations, predicate applications and logical connectives and quantifiers $\forall$, $\exists$. Additionally, we have the modalities $\Box$ (always in the future), $\Diamond$ (sometimes in the future) and $\bigcirc$ (in the next step). The corresponding past modalities are $\boxminus$, $\blacklozenge$ and $\ominus$.

We now specify RBAC in first-order LTL (see Figure 1). The function *user* is rigid (i.e., do not depend on the state), while the predicates UA, PA, *active_in* and *exec* are flexible (i.e., do depend on the state). Hence, *exec* traces the operations performed: $exec(s, op, obj)$ means that session s executes operation *op* on object *obj* in the present (implicit) state. UA has been defined as a flexible predicate in order to express delegation constraints (cf. section 3.2). *auth* means that a user u may be permitted to execute operation *op* on object o.

3. AUTHORISATION CONSTRAINTS FOR CLINICAL INFORMATION SYSTEMS

In the following, we discuss and then formally specify authorisation constraints that can be seen as the foundation of a security policy for clinical information systems such as context constraints, constraints for delegation, grant constraints, and order-based workflow constraints. The focus

of this section lies on delegation concepts which make possible possible information sharing as often required in the healthcare domain.

3.1 Context Constraints

Role-based systems usually treat roles as static attributes or at least attributes that change infrequently. That is, we might not allow the following scenario: A hospital employee works as a doctor in the morning, as a billing clerk in the afternoon, and then as a doctor again the next day. As a result, roles are usually defined with a fixed set of permissions. Unfortunately, in a healthcare setting, permissions assigned to a role are not always static. Sometimes the permissions assigned to a role should be given depending on what the member of the role is currently doing, or the security-related contexts. For example, suppose a hospitals privacy policy grants access to sensitive patient information only to the patient's Primary Care Physician (PCP). What permissions should be assigned to the role of PCP? It is inappropriate to grant the permission to all patients' records to PCP. A doctor should be granted the permissions assigned to the PCP of a patient only when the patient has designated him as the PCP. Therefore, we need to dynamically assign a set of permissions to a role at run-time. This dynamic permission assignment is achieved by applying anchored context constraints during the role activation process in a session. Traditional RBAC permission activation has three steps. At first, a user presents suitable credentials to complete the identification and authentication procedure; then the user has to select a subset of roles from the assigned role set for activating in current session; finally, a particular set of permissions assigned to the subset of roles is granted to the user. In order to apply context constraints, we propose to change the traditional permission activation process as follows. After successful authentication, the user selects a subset of roles for activating in current session. The anchored context constraints from the user are retrieved and applied at role activation; after successful role activation, the anchored context constraints further are applied to permissions of the activated roles; finally a particular set of permissions is granted to the user.

In order to conveniently formulate context constraints for clinical information systems, we introduce additional predicates and sorts (cf. Figure 2). The meaning of these sorts and predicates is as follows:

- $Patients$ – a sort that represents patients
- $Locations$ – a sort that represents locations (like a department of a hospital)
- $location(r, l)$ – true if l is the current location of role r (e.g., role $Clinician$ belongs to department $Radiology$)
- $patient(r, p)$ – true if p is the patient of role r (e.g., role $ReadEPR$ refers to patient ($Mary$))
- $patient(u, p)$ – true if p is the patient of clinician u
- $location(o, l)$ – true if l is the current location of object o (e.g., EPR currently belongs to department $Radiology$)
- $patient(o, p)$ – true if p is the patient of object o (e.g., EPR is the electronic patient record belonging to $Mary$)

spec ContextTemporalRBAC=RichTemporalRBAC
sorts $Patients, Locations$;
flexible preds $location : Objects \times Locations$;
$location : Roles \times Locations$;
$location : Users \times Locations$;
$patient : Users \times Locations$;
rigid pred $patient : Objects \times Patients$;
$patient : Roles \times Patients$;
$isEPR : Objects$;
$isClinician : Roles$;
end

Figure 2: Sorts and predicates for context constraints.

- $location(u, l)$ – true if l is the location of user u

We could also have replaced the predicate $location(u, l)$ by defining a location-specific role that indicates that user u is currently on department l. The user could then activate that role when she is on this department. However, for reasons of simplicity we use the predicate $location(u, l)$ extensively in the subsequent example security policies. The auxiliary predicates $isClinician()$ and $isEPR()$ in Figure 2 are only introduced due to technical reasons, i.e., to indicate that some roles and objects are of certain additional types (in this case, $Clinician$ and EPR).

The actual context constraints can be expressed by the *location* and *patient* predicates. Note the *location* predicates and the $patient(u, p)$ predicate are defined as flexible, whereas the other *patient* predicates are not. The reason for this decision is that the former predicates can change over time (for example, if a user moves to a different department or if a different clinician is now responsible for the patient). Since patient data (like those of the EPR) always belong to the *same* patient, the other patient predicates are defined rigid.

With help of the aforementioned predicates we can formulate context constraints relevant to clinical information systems. Consider, for example, the dynamic permission assignment described above. The formal specification of this rule is as follows:

forall $u : Users, o : Objects, op : Operations,$
$r : Roles, p : Patients;$
$\Box[UA(u, r) \Rightarrow (PA(o, op, r) \wedge patient(o, p)$
$\Rightarrow r\ active\_for\ u \wedge patient(u, p))]$[1]

A further security policy rule, which could be implemented in a clinical information system, is "Clinicians may only read EPRs which belong to their department." This rule can be formally specified in a way similar to the aforementioned example, using the dynamic $auth$ predicate and the context predicates:

forall $u : Users, dep : Locations, epr : Objects;$
$auth(u, read, epr) \wedge isEPR(epr) \Rightarrow (location(epr, dep)$
$\Rightarrow location(u, dep))$

3.2 Delegation

Delegation is an important factor to accommodate aforementioned requirements for secure distributed computing environment. There are many definitions of delegation in the literature [2, 12, 10]. In general, it is referred to as one active entity in a system delegates its authority to another

entity to carry out some functions on behalf of the former. Over the years, researchers have proposed a variety of distributed access control mechanisms. Delegation has been recognised as one of mechanisms to support access management in a distributed computing environment [1].

Zhang et al. [25] introduced RDM2000 (role-based delegation model 2000) for user-to-user delegation in role-based systems. We use RDM2000 model to specify role-based delegation policies in clinical information systems. It formalises the relationship between two user assignments that form a delegation relation (DLGT). It also includes sets of three elements: original user assignments UAO, delegated user assignment UAD, and constraints. The motivation behind this relation is to address the relationships among different components involved in a delegation. In a user-to-user delegation, there are four components: a delegating user, a delegating role, a delegated user, and a delegated role. A delegation relation is one-to-many relationship on user assignments. It consists of original user delegation (ODLGT) and delegated user delegation (DDLGT). We assume each delegation relation may have a duration constraint associated with it. If the duration is not explicitly specified, we consider the delegation as permanent unless another user revokes it.

In some cases, we may need to define whether or not each delegation can be further delegated and for how many times, or up to the maximum delegation depth. We introduce two types of delegation: single-step delegation and multi-step delegation. Single-step delegation does not allow the delegated role to be further delegated; multi-step delegation allows multiple delegations until it reaches the maximum delegation depth. The maximum delegation depth is a natural number defined to impose restriction on the delegation.

Another characteristic of delegation according to Barka and Sandhu [2] is *monotonicity*. Monotonicity refers to the state of authorisation that the delegating role member possesses after delegation. Monotone delegation means that upon delegation the role member maintains the power of his or her role. On the other hand, with a non-monotone delegation, the delegating role member loses the power of the delegated role (at least for the duration of the delegation).

As can be concluded from the above discussion, delegation is a central concept for information sharing. However, due to the strict privacy rules to be enforced in clinical information systems, information sharing should only be possible under certain conditions and should hence be restricted by appropriate authorisation constraints.

Subsequently, we formalise several basic variants of delegation relevant to clinical information systems in first-order LTL. In particular, we use the UAO, UAD, and UA predicates following the terminology mentioned above. The relationship between UAO, UAD, and UA is presented in Figure 3. Authorisation constraints are represented by additional predicates (conditions).

First, we formally specify *non-monotone delegation* in LTL, i.e., user u delegates role r to user $u1$ and loses at the same time the power of the delegated role r:

forall $u, u1 : Users, r, r1 : Roles$;
$delegateNonMon(u, r, u1, r1, ...) \Leftrightarrow$
$[UAO(u, r) \wedge \bigcirc UA(u1, r1) \wedge \neg UA(u1, r) \wedge cond$
$\Rightarrow \bigcirc(UAD(u1, r) \wedge \neg UA(u, r))]$

$\neg UA(u1, r)$ is required here because it is not useful to delegate role r to a user $u1$ who has already been assigned to this role. Moreover, u must obviously belong to r before/during delegation. However, often $u1$ should hold the power of a certain role $r1$ on the delegation process [2], and there sometimes also exist additional conditions (e.g., temporal constraints, constraints concerning the delegation depth) which are to be satisfied to enable the delegation process. These conditions are represented by the *cond* statement[3], which is a first-order LTL formula. The ellipsis ... in the parameter list denotes further parameters which depend on *cond*. Several examples of additional conditions can be found below (cf. sections 4.1 and 4.2).

Following the RDM2000 model, we have specified the ODLGT variant of non-monotone delegation. The DDLGT variant can easily be defined by using the UAD relation instead of UAO for the delegating user. Of course, similar remarks apply to the delegation predicates defined below.

Non-monotone delegation is required for clinical information systems when a patient is transferred from one hospital department to another. In this case, the EPR must be passed on to a different clinician while the former clinician loses this responsibility at the same time.

However, *monotone delegation* is another relevant concept for clinical security policies, for example, when a physician consults a specialist. Here, the former physician still holds the power of the PCP role, but simultaneously delegates at least a part of her PCP access rights to the consulting specialist.

Monotone delegation can now formally be specified in the following way:

forall $u, u1 : Users, r, r1 : Roles$;
$delegateMon(u, r, u1, r1, ...) \Leftrightarrow$
$[UAO(u, r) \wedge \bigcirc UA(u1, r1) \wedge \neg UA(u1, r) \wedge cond$
$\Rightarrow \bigcirc(UAD(u1, r) \wedge UAO(u, r))]$

In the sections above we have specified delegation policies. Strictly speaking, delegation as defined above holds only for one step. Hence, it is also necessary to formulate a rule for maintaining delegation. This can be done as follows:

forall $u : Users, r : Roles$;
$maintain(u, r, ...) \Leftrightarrow [UAD(u, r) \wedge cond \Rightarrow \bigcirc UAD(u, r)]$

Delegation is maintained as long as some condition *cond* is satisfied. For example, a user must be assigned to a certain prerequisite role, otherwise the power of the delegated role would be lost. *cond* could also be a duration restriction constraint, and in case of permanent delegation (no duration restriction) *cond* could clearly be set to *true*.

3.3 Revocation

Several different semantics are possible for user revocation. Hagström and others [15] categorised revocations into three dimensions in the context of owner-based approach: global and local (propagation), strong and weak (dominance), and deletion or negative (resilience). Barka and Sandhu [2] further identified user grant-dependent and grant-independent revocation (grant-dependency). We articulate user revoca-

[2]More precisely, "on the delegation process" means here "immediately after the delegation step" as expressed by $\bigcirc UA(u1, r1)$. This formulation will be used several times in the following specifications and examples.

[3]Clearly, the *cond* statement is somewhat semi-formal and has been introduced for the sake of readability.

spec DelegateTemporalRBAC=ContextTemporalRBAC

flexible preds $location : Objects \times Locations$;
$UA : Users \times Roles$;
$UAD : Users \times Roles$;
$UAO : Users \times Roles$;
... further predicates such as delegateNonMon

forall $u : Users$; $r : Roles$
- $UAD(u, r) \vee UAO(u, r) \Leftrightarrow UA(u, r)$
- $\neg(UAD(u, r) \wedge UAO(u, r))$
- *... see text*

end

Figure 3: Predicates for delegation.

tion in the following dimensions: grant-dependency, propagation, and dominance. Grant-dependency refers to the legitimacy of a user who can revoke a delegated role. Grant-dependent revocation means only the delegating user can revoke the delegated user from the delegated role membership. Grant-independent revocation means any original user of the delegating role can revoke the user from the delegated role. Dominance refers to the effect of a revocation on implicit/explicit role memberships of a user. A strong revocation of a user from a role requires that the user be removed not only from the explicit membership but also from the implicit memberships of the delegated role. A weak revocation only removes the user from the delegated role (explicit membership) and leaves other roles intact. Strong revocation is theoretically equivalent to a series of weak revocations. To perform strong revocation, the implied weak revocations are authorised based on revocation policies. However, a strong revocation may have no effect if any upward weak revocation in the role hierarchy fails [22]. Propagation refers to the extent of the revocation to other delegated users. A cascading revocation directly revokes a delegated user assignment in a delegation relation and also indirectly revokes a set of subsequent propagated user assignments. A non-cascading revocation only revokes a delegated user assignment.

Revocation is an essential element of security policies in the healthcare domain. For example, a specialist should be revoked from the role *ReadERP* after the consultation has been done. When a patient leaves the hospital, we also need revocation such that all the clinicians who treated the patient should lose the power to read patient data (at least, after a certain while).

A formal specification of a simple revocation rule is given below:

forall $u : Users, r : Roles$;
$revoke(u, r, ...) \Leftrightarrow [UAD(u, r) \wedge cond \Rightarrow \bigcirc\neg UAD(u, r)]$

The abbreviation *cond* means in this context if an appropriate condition holds, the role r is revoked from user u. For example, in order to read a patient's EPR, the clinician must belong to the same department as that of the patient, otherwise the power of this role is lost.

3.4 Grant Constraints

In clinical information systems there often arises the situation that a user u grants a role r to a user $u1$, while u *does not necessarily hold* the role herself. Compare this with the aforementioned delegation policies where it is a prerequisite that the delegator holds the role to be delegated.

One example of this situation is the admission of a patient at the hospital reception. The receptionist passes the patient's EPR to the department where the patient should be treated. On the other hand, the receptionist is certainly allowed to enter clerical patient data, but cannot see any medical data. Another example is the health card ("Gesundheitskarte"), which will be introduced by the German government: The patient may grant the right to read certain patient data to a doctor he trusts. On the other hand, the patient is not permitted to read his own patient data without authorisation by a doctor (this shall, for example, prevent the patient from doing any harm to himself in case he misunderstands his patient/diagnosis data).

The formal specification of this grant constraint is given below:

forall $u, u1 : Users, r, r1, ar : Roles$;
$grant(u, ar, u1, r1, r, ...) \Leftrightarrow$
$[UA(u, ar) \wedge \bigcirc UA(u1, r1) \wedge \neg UA(u1, r) \wedge u \neq u1 \wedge cond$
$\Rightarrow \bigcirc UA(u1, r)]$

In this case, u does not necessarily belong to r, but to an administrative role ar. Of course, other security policies based upon administrative models (e.g., ARBAC97 [22], URA02 and PRA02 [21]) can also be elegantly formulated by first-order LTL because this calculus is well-suited to describing dynamic behaviour like that inherent in the administrative security policies. Roles obtained via the granting mechanism can also be revoked in a way similar to the revocation of delegated roles (refer to section 3.3). For reasons of brevity, the formal specification of this kind of revocation is left out here.

3.5 Order-based workflow Constraints

First-order LTL was employed in [18] in order to formally specify separation of duty (SoD) security policies where access history matters, e.g., dynamic object-based SoD, as proposed by Nash and Poland [19], and history-based SoD [24]. In particular, an *exec* predicate was introduced which holds if the operation *op* is being executed on the object o at present (cf. section 2.2). Certainly, the *exec* predicate can be used to express workflow security policies where tasks are to be executed by WFMSs (workflow management systems) [4] in a certain order. In the example workflow given in section 4.1 the *exec* predicate is used several times to specify a security policy for the workflow of a diagnostic finding.

4. CASE STUDIES

Having enumerated authorisation constraints' characteristics of clinical information systems, we now discuss two examples of security policies for typical clinical business processes.[4]

The first example is a workflow for a diagnostic finding, and the second deals with passing on the EPR between different hospital departments.

4.1 Diagnostic Finding

The workflow for the diagnostic finding is presented in Figure 4. The tasks the workflow consists of, and the authorisation constraints of the single steps are informally defined in the following way:

[4]However, it should be stressed that there might be different security policies in other hospitals.

1. A clinician responsible for the patient (PCP) requests a diagnostic finding from a clinician of a different department (in our case here, radiology).

2. After the patient has been examined, the clinician responsible for the finding writes the finding down. In addition, having the right to write down the finding also implies the right to read the finding.

3. The chief physician of the radiology department must sign this finding.

4. The finding is sent back to the requesting PCP (cf. step 1) and at the same time the clinician responsible for the finding loses the power of writing the finding down (since this should be done when the requesting clinician wants to read the finding).

Notice that the tasks have to be performed in the order given in the aforementioned enumeration (order-based constraint).

The security policy (consisting of several rules with authorisation constraints) for this workflow is depicted in Figure 5. As explained above, the auxiliary predicates $isEPR()$, $isFinding()$, etc. are only introduced to indicate that some roles and objects are of certain additional types (like PCPs, findings, EPRs). In addition, the first rule expresses the permission assignment for the finding-related roles. The

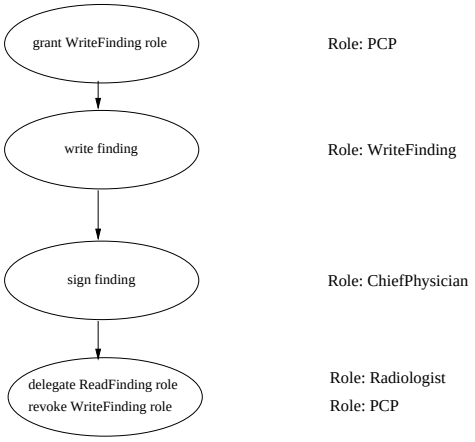

Figure 4: The diagnostic finding workflow.

other rules are the formal pendants of the authorisation constraints which have been informally described above. Thus, the second rule states that a user who possesses the power of writing down the diagnostic finding at the same time is permitted to read this finding. In the third rule we rather employ the concept of granting a role than that of delegation, since the requesting clinician (PCP) is in most cases not competent to perform the diagnostic finding herself. The radiologist may only write down the finding if she has obtained the power for the appropriate role ($WriteFinding$) from the patient's PCP before. Furthermore, several context constraints are required for this granting process which serve as preconditions. These context constraints are in particular used to indicate that it is patient p's PCP and that p resides actually on the same department as the PCP [5]. Finally, $\ominus false$ means that there are no preceding worlds, i.e., we consider here the start of the workflow (cf. step 1).

The fourth rule expresses the fact that the chief physician (of the radiology department) can only sign the finding if there exists a user who must have written down the finding one step before. The fifth rule means that a radiologist can only delegate the role $ReadFinding$ to the requesting clinician (cf. exist part of this rule) after the finding has been signed by the chief physician of the radiology department. We have decided to apply monotone delegation here because the radiologist should also have the right to read the finding after the delegation process (at least for a certain while). The last rule states that the finding physician loses the power of the $WriteFinding$ role when the $ReadFinding$ role is being delegated back to the requesting clinician.

To sum up, the security policy emphasises the relevance of compositional constraints within the healthcare domain: The grant predicate is only true if the granting user is patient p's PCP and the PCP is on the same department as p (context constraints). In addition, the $WriteFinding$ role may only be delegated back to a user u if u is p's PCP, u is on the same department as p and the finding has been signed one step before (context and order-based constraints). As a consequence, we must cope with a quite complex dynamic policy where the UA relation changes over time and where the policy is composed of different types of constraints.

In the next section, we give another example for a business process relevant to clinical information systems, namely, passing on the EPR from one hospital department to another.

4.2 Passing on the EPR

In several clinical information systems the EPR has been or will be introduced soon. Owing to the fact that patients are often transferred from one department (say, a urological department) to another (say, intensive care unit) the EPR must also be passed on to the patient's new department. Several conditions (authorisation constraints) must be fulfilled in order to imitate the paper-based patient record. We now give the following constraints involved in the process of passing on a patient's EPR:

- A parameterised role $respEPR$ (with the patient as the parameter) is defined, which includes delegate rights for the EPR as well as read and append rights. $respEPR$ expresses the responsibility for a certain EPR, which belongs to a unique patient. Hence, the role $respEPR$ and the object epr must be parameterised by the same unique patient (context constraint).

- The receptionist of the hospital is responsible for passing on the EPR to the department where the patient shall be treated. However, the receptionist must not be assigned to the $respEPR$ role. Thus, we can formulate this with the $grant$ predicate:

  forall $u, u1 : Users, respEPR, clinician : Roles,$
  $dep : Locations;$
  $grant(u, Receptionist, u1, clinician, respEPR, dep) \Leftrightarrow$
  $[UA(u, Receptionist) \wedge \bigcirc UA(u1, clinician) \wedge u \neq u1 \wedge$
  $\neg(UA(u1, respEPR)) \wedge cond \Rightarrow \bigcirc UA(u1, respEPR)]$

  In addition, the following condition must be satisfied:

[5] Observe that "p's current department" is expressed by $location(epr, dep)$, assuming that the EPR is always on the same department as p.

spec DiagnosticFindingWorkflow = DelegateTemporalRBAC

rigid ops $read, write, sign : Operations$;
$ReadFinding, WriteFinding, ChiefPhysician, Radiologist : Roles$;

rigid preds $isFinding : Objects$; $isPCP : Roles$;

flexible preds $delegateMon : Users \times Roles \times Users \times Roles \times Objects \times Locations \times Patients$;
$grant : Users \times Roles \times Users \times Roles \times Roles \times Objects \times Locations \times Patients \times Objects$;
$revoke : Users \times Roles \times Users \times Roles \times Roles \times Objects \times Locations \times Patients$;

forall $u, u1, u2 : Users$; $p : Patients$; $pcp : Roles$; $epr, finding : Objects$; $dep : Locations$;

- $((PA(write, finding, WriteFinding) \Leftrightarrow isFinding(finding)) \wedge ((PA(read, finding, ReadFinding) \Leftrightarrow isFinding(finding))$
  $\wedge ((PA(sign, finding, ChiefPhysician) \Leftrightarrow isFinding(finding))$

- $UA(u, WriteFinding) \Rightarrow UA(u, ReadFinding)$

- $grant(u, pcp, u1, Radiologist, WriteFinding, finding, dep, p, epr) \Leftrightarrow$
  $(UA(u, pcp) \wedge \bigcirc UA(u1, Radiologist) \wedge \neg UA(u1, WriteFinding) \wedge u \neq u1 \wedge cond \Rightarrow \bigcirc UA(u1, WriteFinding))$
  with
  $cond = isEPR(epr) \wedge isPCP(pcp) \wedge isFinding(finding) \wedge patient(epr, p) \wedge patient(pcp, p) \wedge patient(finding, p) \wedge$
  $\bigcirc(location(epr, dep) \wedge location(pcp, dep)) \wedge \ominus false$

- $auth(u2, sign, finding) \Rightarrow \exists u' : Users.\ \ominus exec(u', write, finding)$

- $delegateMon(u1, ReadFinding, u, pcp, finding, epr, dep, p) \Leftrightarrow$
  $(UAO(u1, ReadFinding) \wedge \bigcirc UA(u, pcp) \wedge \neg UA(u, ReadFinding) \wedge cond$
  $\Rightarrow \bigcirc(UAD(u, ReadFinding) \wedge UAO(u1, ReadFinding)))$
  with
  $cond = isEPR(epr) \wedge isPCP(pcp) \wedge isFinding(finding) \wedge patient(epr, p) \wedge patient(pcp, p) \wedge patient(finding, p) \wedge$
  $\bigcirc(location(epr, dep) \wedge location(pcp, dep)) \wedge \exists u' : Users.\ \ominus(exec(u', sign, finding) \wedge UA(u', ChiefPhysician))$

- $revoke(u1, WriteFinding, pcp, finding, epr, dep, p) \Leftrightarrow (UA(u1, WriteFinding) \wedge cond \Rightarrow \bigcirc \neg UA(u1, WriteFinding))$
  with
  $cond = \neg(isEPR(epr) \wedge isFinding(finding) \wedge isPCP(pcp) \wedge patient(pcp, p) \wedge patient(finding, p) \wedge patient(epr, p) \wedge$
  $\bigcirc (location(epr, dep) \wedge location(pcp, dep)) \wedge \exists u' : Users.delegateMon(u1, ReadFinding, u', pcp, finding, epr, dep, p))$

end

Figure 5: Security policy for the diagnostic finding workflow, formalised within temporal first-order logic.

$cond = \ominus false \wedge isRespEPR(respEPR) \wedge$
$isClinician(clinician) \wedge \bigcirc location(u1, dep) \wedge$
$\forall u' : Users.\neg UA(u', respEPR)$.

Here again, $\ominus false$ means we have no further worlds before, i.e., we consider the moment of patient admission. Furthermore, $\forall u' : Users.\neg UA(u', respEPR)$ means that no user is assigned to the role $respEPR$ at the beginning. This in particular holds for the receptionist.

The constraints enumerated below could be formally specified in a similar way. However, due to space limitations the formal specifications are left out from now on.

- The receptionist may pass a patient's EPR (i.e., grant the role $respEPR$) *at most* to one clinician.
- The EPR is passed on from one department to another if the patient is transferred to a different department. At the same time, the clinician of the former department who was responsible for the EPR loses the role $respEPR$ while a clinician of the latter department gains this power (non-monotone delegation).
- Monotone delegation is allowed, but only if a clinician wants to delegate the role $respEPR$ within her own department.

The formalisation of the authorisation constraints can be helpful for the implementation of clinical information systems in two ways. Firstly, the calculus can be used for policy analysis, i.e., to deduce certain properties of the security policies (statically). For example, we could try to prove the lemma "If the role $respEPR$ is assigned or delegated to different users, they must necessarily belong to the same department". The correctness of this proof can then be assured by employing a theorem prover such as Isabelle [20]. First work in this direction has been presented in [5] where first-order LTL has been embedded into Isabelle. Verifying healthcare security policies remains future work.

Secondly, the formal specifications can be used as a recipe for the implementation of security mechanisms for clinical information systems. For example, the rules/constraints presented above give a blueprint of how a patient's EPR can be passed on between different hospital departments or between the hospital archive and the departments. Note, since the receptionist may only see cerical data or pass on the EPR, this grant rule is even more strict than the pendant for the paper-based business process. In the paper-based variant, clearly the receptionist has the opportunity to look into the patient record, violating the principle of least privilege.

5. CONCLUSION AND OUTLOOK

We formally specified authorisation constraints characteristic of clinical information systems, in particular, delegation, revocation, order-based and context constraints. We further demonstrated with the help of two example business processes how first-order LTL can be employed to elegantly specify more complex security policies which consist

of different types of authorisation constraints. First-order LTL is specifically well-suited to expressing highly dynamic policies like delegation where the UA relation changes over time. The authors hope that the specification of such security policies helps in providing a recipe for the implementation of dynamic access mechanisms required for clinical information systems.

In the future we could consider more complex and realistic security policies, e.g., how the health card, which will be introduced in Germany, might be integrated with the EPR. In addition, we could identify new types of authorisation constraints relevant to the healthcare domain in order to define an encompassing security policy for clinical information systems. Last but not least, it could be systematically analysed which kind of properties of security policies can be deduced statically and employ the theorem prover Isabelle for this verification purpose.

6. REFERENCES

[1] T. Aura, *Distributed access-rights management with delegation certificates*, Lecture Notes in Computer Science **1603** (1999), 211–236.

[2] E. Barka and R. Sandhu, *A role-based delegation model and some extensions*, Proceedings of 16th Annual Computer Security Application Conference, December 11–15 2000, pp. 125–134.

[3] E. Bertino, P.A. Bonatti, and E. Ferrari, *TRBAC: A temporal role-based access control model*, Proc. of the 5th ACM Workshop on Role-Based Access Control (N.Y.), ACM Press, July 26–27 2000, pp. 21–30.

[4] E. Bertino, E. Ferrari, and V. Atluri, *An authorization model for supporting the specification and enforcement of authorization constraints in workflow management systems*, ACM Transactions on Information and System Security **2** (1999), no. 1, 65–104.

[5] M. Drouineaud, M. Bortin, P. Torrini, and K. Sohr, *A first step towards formal verification of security policy properties for RBAC*, Proc. of the 4th International Conference on Quality Software, 2004, pp. 60–67.

[6] EU, *Directive on the protection of individuals with regard to the processing of personal data and on the free movement of such data. Directive 95/46/EC. http://www.privacy.org/pi/intl_orgs/ec/eudp.html*, 1995.

[7] D. Ferraiolo, D. Gilbert, and N. Lynch, *An examination of federal and commercial access control policy needs*, Proc. of the NIST-NCSC Nat. (U.S.) Comp. Security Conference, 1993, pp. 107–116.

[8] D.F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramoli, *Proposed NIST standard for role-based access control*, ACM Transactions on Information and System Security **4** (2001), no. 3, 224–274.

[9] Deutsches Institut für Medizinische Dokumentation und Information, *Gesundheitskarte. http://www.dimdi.de/de/ehealth/karte/index.htm*, 2004.

[10] M. Gasser and E. McDermott, *An architecture for practical delegation in a distributed system*, Proc. IEEE Symposium on Research in Security and Privacy, 1990, pp. 20–30.

[11] C.K. Georgiadis, I. Mavridis, G. Pangalos, and R.K. Thomas, *Flexible team-based access control using contexts*, Proc. of the ACM Symposium on Access Control Models and Technologies, May 3–4 2001, pp. 21–27.

[12] H. M. Gladney, *Access control for large collections*, ACM Transactions on Information Systems **15** (1997), no. 2, 154–194.

[13] V. D. Gligor, S. I. Gavrila, and D. Ferraiolo, *On the formal definition of separation-of-duty policies and their composition*, 1998 IEEE Symposium on Security and Privacy (SSP '98), IEEE, May 1998, pp. 172–185.

[14] R. Goldblatt, *Logics of time and computation, second edition, revised and expanded*, CSLI Lecture Notes, vol. 7, CSLI, Stanford, 1992 (first edition 1987), Distributed by University of Chicago Press.

[15] A. Hagström, S. Jajodia, F. Parisi-Presicce, and D. Wijesekera, *Revocations -a classification*, 14th IEEE Computer Security Foundations Workshop (CSFW '01), IEEE, June 2001, pp. 44–58.

[16] Z. Manna, N. Bjørner, A. Browne, E.Chang, M. Colón, L. de Alfaro, H. Devarajan, A. Kapur, J. Lee, H. Sipma, and T.E. Uribe, *STeP: The stanford temporal prover*, TAPSOFT '95: Theory and Practice of Software Development (P.D. Mosses, M. Nielsen, and M.I. Schwartzbach, eds.), LNCS, vol. 915, Springer-Verlag, 1995, pp. 793–794.

[17] Z. Manna and A. Pnueli, *Temporal verification of reactive systems: Safety*, Springer-Verlag, New York, 1995.

[18] T. Mossakowski, M. Drouineaud, and K. Sohr, *A temporal-logic extension of role-based access control covering dynamic separation of duties*, Proc. of TIME-ICTL 2003), Cairns, Queensland, Australia, July 8–10 2003.

[19] M. J. Nash and K. R. Poland, *Some conundrums concerning separation of duty*, Proc. IEEE Symposium on Research in Security and Privacy, 1990, pp. 201–207.

[20] T. Nipkow, L.C. Paulson, and M. Wenzel, *Isabelle/HOL — A proof assistant for higher-order logic*, Springer Verlag, 2002.

[21] S. Oh and R. Sandhu, *A model for role organization using organizational structures*, Proc. of the 7th ACM Symposium on Access Control Models and Technologies (New York), ACM Press, June 3–4 2002, pp. 155–162.

[22] R. Sandhu, V. Bhamidipati, and Q. Munawer, *The ARBAC97 model for role-based administration of roles*, ACM Transactions on Information and System Security **2** (1999), no. 1, 105–135.

[23] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman, *Role-based access control models*, Computer **29** (1996), no. 2, 38–47.

[24] R. Simon and M. Zurko, *Separation of duty in role-based environments*, 10th IEEE Computer Security Foundations Workshop (CSFW '97), IEEE, June 1997, pp. 183–194.

[25] L. Zhang, G.-J. Ahn, and B.-T. Chu, *A rule-based framework for role-based delegation and revocation*, ACM Transactions on Information and System Security **6** (2003), no. 3.