Deriving Access Control Policies from Requirements Specifications and Database Designs

Qingfeng He and Annie I. Antón
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8207 USA
{qhe2, aianton}@eos.ncsu.edu

Abstract

Access control is a mechanism for achieving confidentiality and integrity in software systems. Specifying access control policies (ACPs) is a complex process that can benefit from requirements engineering techniques. In this paper, we present a method for deriving access control policies from software requirements specifications (SRS) and database designs. The approach provides prescriptive guidance for how to derive and specify ACPs. It also improves the quality of requirements specifications and the database designs by clarifying ambiguities and resolving conflicts across both artifacts. The approach provides traceability support between requirements, access control policies and design decisions, ensuring consistency among these artifacts. Examples from two projects are employed to demonstrate how the approach helps bridge the gap between requirements and design.

1. Introduction

Access control (AC) mechanisms ensure every access to a system and its resources is controlled according to a set of predefined policies [39]. It is one of the major security mechanisms used to achieve confidentiality, integrity and data privacy in software systems [16]. We use these terms as follows: *Confidentiality* means that information is not disclosed to unauthorized persons, processes or devices. *Integrity* means that unauthorized persons, processes or devices cannot modify information. *Privacy* implies that data is protected so that it is used only for authorized business purposes, based on legal requirements, corporate policies and end-user choices.

In the design of software applications, access control analysis entails analyzing business tasks and organizational structures to specify access control policies (ACPs). Defining and deploying ACPs is both a conceptually and practically complex process because a software system can have many users performing various tasks and many resources that need to be protected via access control [37, 38]. Another challenge is the complexity of the organization for which a system is designed — it is difficult to identify and agree upon a common set of roles and associated permissions within an organization that may have hundreds of roles to be considered.

Ideally, security-related software requirements are analyzed and specified before system design rather than as an afterthought. These security requirements should drive ACP specification activities. However, current policy specification efforts often occur after systems are deployed [7]. Because policy specification efforts are often isolated from the software requirements, the resulting ACPs and requirements may not be compliant with one another. Our goal is to develop techniques that help bring policies and system requirements into better alignment.

Requirements engineering (RE) entails discovering the real-world goals for which a software system is intended [29]. During RE, analysts produce a software requirements specification (SRS) that details the envisioned system's functions and constraints. ACPs may be derived from system requirements, but prescriptive guidance is needed to aid in this specification process.

Researchers are recognizing the need to bridge the gap between requirements analysis and complex ACP specification [7]. Existing RE approaches (e.g., KAOS [13], *i** [40] and the analytical role modeling framework [7]) provide limited support as we discuss herein. In this paper, we present a method for deriving ACPs from a system's SRS and database design. This RE activity provides prescriptive guidance for deriving access control policies and improves the quality of the SRS as well as the database design.

Field studies have shown the importance of maintaining traceability during the requirements process [20, 21]. Requirements and policy management are a significant challenge, however. One reason for these difficulties is that both requirements and policies change during the system development process and it is challenging to maintain traceability in the face of inevitable evolution. The approach presented herein features traceability support as a prominent design principle.

The rest of the paper is structured as follows. Section 2 summarizes the most relevant work. Section 3 overviews our approach for deriving ACPs from requirements

specifications and database designs. Section 4 introduces the two systems that serve as the basis for our case studies: the Security and Privacy Requirements Analysis Tool (SPRAT) and the Transnational Digital Government (TDG) Project. Section 5 details the ACP specification process and associated heuristics, using concrete examples from both case studies. Finally, Section 6 provides a discussion and summarizes our plans for future work.

2. Background and related work

This section discusses the key elements of ACPs and summarizes prior related work.

2.1. Access control policies in security

An access control system is typically described in three ways: access control policies, access control models and access control mechanisms [39]. *Access control policies* are security requirements that describe how access is managed, what information can be accessed by whom, and under what conditions that information can be accessed [16]. These policies are enforced via a mechanism that mediates access requests and makes grant/deny decisions. The *access control mechanism* defines the low-level functions that implement the controls imposed by the policies. It must work as a reference monitor [39], a trusted component intercepting each and every request to the system. *Access control models* provide a formal representation of an access control system. They provide ways to reason about the policies they support and prove the security properties of the access control system. Access control models provide a level of abstraction between policies and mechanisms, enabling the design of implementation mechanisms to enforce multiple policies in various computing environments.

ACPs can be broadly grouped into three main policy categories: Discretionary Access Control (DAC), Mandatory Access Control (MAC), and Role-Based Access Control (RBAC). DAC policies enforce access control based on the identity of the requestor and the explicit rules specifying who can or cannot perform specific actions on specific objects. Early discretionary access control models, such as the access control matrix model [27, 19] and the HRU model [23], provide a basic framework for describing DAC policies. It is the users' discretion to pass their privileges on to other users. Thus, DAC policies are vulnerable to Trojan Horse attacks [39].

MAC policies enforce access control based on the security classifications of subjects and objects. For example, the lattice-based multilevel security policy [12], policies represented by the Bell-LaPadula model [5, 6] and the Biba model [3] are MAC policies. MAC policies protect indirect information leakages (e.g., Trojan Horse attacks), but are still vulnerable to covert channel attacks [39, 32].

RBAC policies employ roles to simplify authorization management for enforcing enterprise-specific security policies [16, 36]. The RBAC model is an alternative to traditional DAC and MAC models and has received increased attention in commercial applications, such as the Oracle 9i DBMS [9]. RBAC is now an American National Standard[1].

Instead of considering ACP specification from a holistic, real-systems perspective as we advocate in this paper, current ACP specification research has a much narrower focus (e.g., uniform or flexible ways to specify ACPs [25], specifying ACPs for XML documents [17], etc.). There are few reported methods and experiences relating ACP specification in real software systems. In the RBAC literature[2], researchers are investigating role engineering, the process of defining roles and privileges as well as assigning privileges to roles [8]. Several role-engineering approaches employ RE concepts. For example, Fernandez and Hawkins suggest deriving the needed rights for roles from use cases [14]. Neumann and Strembeck propose a scenario-driven approach for engineering functional roles in RBAC [30]. Role engineering is specific to RBAC, whereas our method is a more general approach for specifying ACPs.

2.2. Elements of access control policies

An access control *policy* is comprised of a set of access control rules. A *rule* can have various modes (e.g., permit/deny/oblige/refrain). This paper is focused on allow and deny rules. *Allow* rules authorize a subject to access a particular object. *Deny* rules explicitly prohibit a subject from accessing a particular object. When a subject requests to perform an action on an object, the corresponding rules will be evaluated by the enforcement engine for that request. A typical access control *rule* is expressed as a 3-tuple <*subject*, *object*, *action*>, such that a *subject* can perform some *action* on an *object* [11]. A *subject* is a user or a program agent, or any entity that may access objects. An *object* is a data field, a table, a procedure, an application or any entity to which access is restricted. An *action* is a simple operation (e.g. read or write) or an abstract operation (e.g. deposit or withdraw). In this paper, we extend the typical AC rule 3-tuple to include conditions and obligations as we now discuss.

An ACP may express additional *conditions* that must be satisfied before an access request can be granted. For example, in healthcare applications, the location from which the access request originates might affect the grant/deny decision [2]. If an access request is from the emergency room, then the request may be granted. In this case, we can specify *the location of the request is emergency room* as a condition for the AC rule. Additionally, in the context of privacy protection, we may specify additional conditions to restrict access to personal data. For example, purpose is a standard entity in most privacy policies as recognized in P3P (The Platform for Privacy Preferences Project) [31]. When a subject (e.g., a nurse) requests to perform an action on an object (in the context of privacy, the object is often personal data, e.g., medical records), the purpose of the operation should be

[1] American National Standard: ANSI INCITS 359-2004.

[2] See ACM RBAC and SACMAT workshop series, http://www.sacmat.org/.

bound to the purposes consented to by the data subjects. This is the purpose binding principle [15], which can be enforced by specifying conditions for ACPs. Karjoth et al. treat purpose the same as subjects, objects, actions, etc. for a privacy authorization rule because purpose tends to be the primary focus of privacy protection [26]. In this paper, we do not treat purpose as an independent element because it is mainly useful for protecting data privacy.

Obligations [4] are actions that must be fulfilled if a request to access an object is granted. For example, consider: *require affiliates to destroy customer data after service is completed*. In this case, "destroy customer data" is an obligation that must be satisfied by affiliates. Obligation-based security policies can be enforced if they can be completely resolved within an atomic execution [34]. If the obligation is not an immediate action (e.g., it is a task that will be executed in the future), monitoring and auditing its execution might be sufficient for enforcement [4].

In requirements specification, we are concerned with the *actions* for which each actor (*subject*) is responsible, the conditions under which each action can occur (constraints and pre-*conditions*) and the post-conditions (*obligations*) that must be satisfied. Each of the five access control elements can be mapped to a requirements specification element. Because we seek to ensure that ACPs comply with the requirements, we have mapped these five ACP elements <*subject, object, action, condition, obligation*> to requirements elements. This mapping helps guide analysts as they derive ACPs from requirements.

2.3 Access control analysis in RE

Requirements engineering researchers are investigating methods and tools to help analyze and specify access-related security requirements. Fontaine [18] employs KAOS, a goal-based requirements acquisition and elaboration method [13], to refine security requirements into specific authorization rules and ACPs expressed in Ponder — a language for specifying management and security policies for distributed systems [10]. Fontaine's work is an important step towards requirements-level access control analysis for security policy specification. However, the method for mapping KAOS specifications to Ponder policies is not general in the sense that not all types of Ponder policies can be generated from KAOS specifications. Fontaine has thus far only shown how to specify authorization and obligation policies [18], not refrain and delegation policies.

Liu et al. applied the *i** framework [40], a goal-based requirements analysis method, to support access control analysis by modeling the dependencies among actors, tasks and a system's resources [28]. However this approach is limited in that it assumes the roles and privileges have been previously derived. Additionally, it provides no guidance as to how roles and privileges are identified, from where they originate, or how privileges are assigned to these roles. Moreover, it is difficult to model context and constraint information in the *i** framework. These topics remain major challenges in access control analysis during RE.

Crook et al. proposed an analytical role-modeling framework to model ACPs [7]. This approach offers two contributions. First, the framework clarifies the need to model ACPs during requirements analysis. Second, the rationale for deriving roles based on organizational structures is very useful. Job positions in an organization can be mapped to *roles* in RBAC. Organizational and seniority hierarchies can be mapped to RBAC *role hierarchies*. This is common in identity management products in which individuals in a particular department and/or division are classified into a specific role and that role grants them the access rights to specific (e.g., bank) accounts. Deriving roles from organizational structures facilitates the user assignment and authorization management processes in access control.

3. Access control analysis & ACP specification

This section overviews our approach and introduces the general principles for this kind of analysis.

3.1. Overview of our approach

Figure 1 portrays our approach using a traditional ICOM (Inputs-Constraints-Outputs-Mechanisms) model. The required source documents for the approach are: the initial SRS and database design (e.g., E/R diagram). The two source documents are complementary in that the requirements specification justifies the rationale for the ACPs (e.g., why a user is given a particular privilege to perform a task or access an object), whereas the database design details the objects to which any access should be controlled. If the requirements specification is unclear, analysts might also consult the database design to clarify these ambiguities. One of these source documents alone cannot achieve this result – both are necessary.

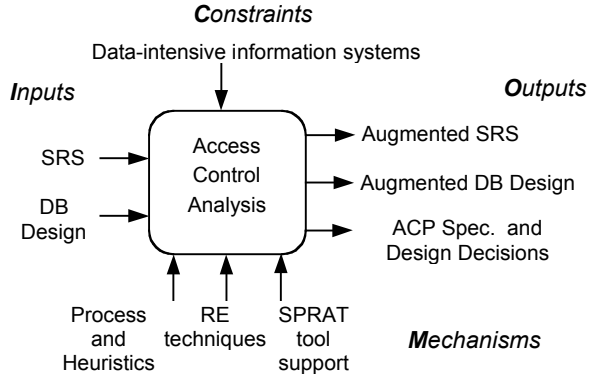

Figure 1: ACP Specification ICOM Model

The approach guides analysts with a detailed process description and heuristics as they examine both types of documents and derive ACPs based on this information. During the process, analysts employ RE techniques, such as goal [13] and scenario analysis techniques [33], to help

identify access control elements. During the process, ambiguities in the requirements specification are clarified, and inconsistencies between the SRS and database design are identified and resolved. A tool is currently under development to support the process.

The overall objective of our approach is to produce a comprehensive set of ACP specifications. Additionally, the source documents inevitably benefit from the process as the SRS and database design are augmented, which results in more complete, correct and less ambiguous project documentation. The approach also yields a documented list of design decisions made during the analysis process.

Our approach is based on the following three assumptions that serve to narrow the scope of our efforts to data intensive information systems:

Assumption #1: The existence of a system's database design and requirements specification are required. Both are pre-requisite source documents for this approach. ACPs, in a sense, bridge the gap between an information system's requirements specification and its database design.

Assumption #2: *Data* in software systems must be protected. There are various objects in a system to which access may be restricted. For example, an employee may be allowed to print jobs to a local printer, but not a central printer in the neighboring office. This kind of resource access control is beyond the scope of this paper. Instead, we focus on access to data within a database.

Assumption #3: Restricting access to data may be implemented at different levels, e.g., function, application, etc. For example, we can implement the *System Administrator*'s capabilities in one or more functions and restrict users' access to these functions. Sometimes, restricting access to functions and applications is important for other security reasons beyond simply protecting the data. However, function-level and application-level access controls are beyond the scope of this paper.

Assumption #4: ACPs are specified for data-intensive *information systems*. These information systems are typically supported by a database containing sensitive data. We have not investigated ACP specifications for security kernels, such as file access in operating systems.

Figure 2 portrays the activities an analyst undertakes to derive ACPs from requirements specifications and database designs. The process is detailed in Section 4, employing concrete examples from two case studies.

3.2 Access Control Analysis Principles

The high-level activities described above provide an overview of our method. In practice, we adopt several principles to guide this process.

Principle #1: Software requirements and the operational functioning of policy enforcing systems are often misaligned. Policies and system requirements must be brought into better alignment to ensure that unauthorized accesses to sensitive data and security breaches are prevented. The underlying principle of our approach is thus to ensure compliance between access control policies, system requirements and database design.

Principle #2: Traceability between ACPs and requirements must be maintained. Both policies and requirements may change throughout system development and even after the system is deployed. They are interdependent with respect to change. When one of them changes, it is important to make appropriate changes to the other. Traceability helps analysts track changes and maintain consistency between requirements and policies.

Principle #3: In our approach, ACPs are data-centric rather than function-specific (e.g., logging into a system is not codified as an ACP). If a requirement does not describe *access* to some *data* (e.g., the system shall support language translation between English and Spanish), then we cannot derive access control rules from it.

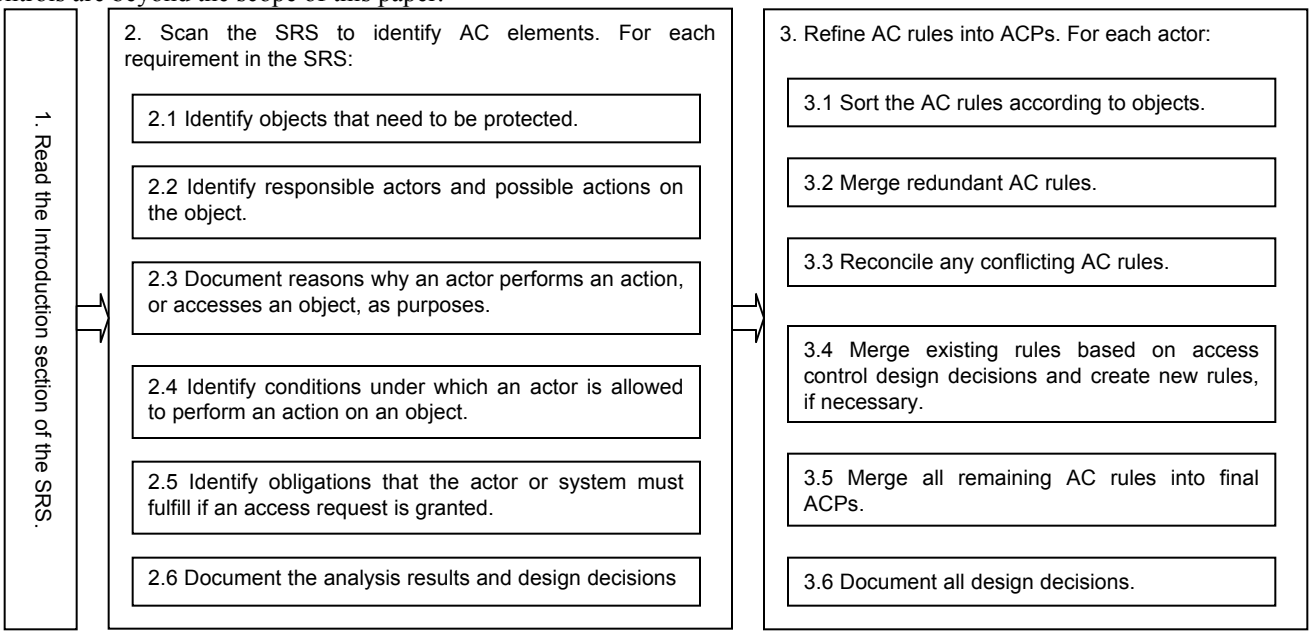

Figure 2. The process of access control analysis

Principle #4: Access control analysis and requirements analysis is an iterative process. Both require one to maintain documents containing information with different levels of formality and describing different kinds of information. Therefore, it is helpful to employ an inquiry-driven approach [33] to document important or recurring questions, design decisions as well as identified inconsistencies that are pending resolution.

4. Case studies

Two case studies involving two real systems have enabled us to preliminarily evaluate the method introduced in Section 3 as we now discuss.

4.1. The Security and Privacy Requirements Analysis Tool (SPRAT)

The Security and Privacy Requirements Analysis Tool (SPRAT) [24] is a research tool under development at North Carolina State University with funding from the U.S. National Science Foundation (NSF). The tool supports goal-based and scenario-based requirements analysis and provides support for analyzing and specifying security and privacy requirements as well as ACPs. It builds upon and extends two existing tools, the Privacy Goal Management Tool (PGMT) [1], and the Scenario Management and Requirements Tool (SMaRT) [35].

With the help of SPRAT, analysts can seamlessly integrate goals, scenarios, requirements, policies and documents in a project repository and conveniently trace from one element (e.g., a goal) to another (e.g., a scenario) during requirements analysis. The SPRAT architecture calls for centralized data storage with distributed client access, with an option that allows local data storage. The envisioned users include requirements analysts, policy makers and security engineers. The information stored in the centralized database is proprietary and needs to be protected. Thus, access control is critical For example, we have a rich privacy goal repository in the PGMT, which is the result of four case studies conducted over the course of three years. We treat these goals as valuable assets that need to be protected via access control. SPRAT analysts will have certain permissions to access these goals. Additionally, industry has expressed interest in having access to these goals to help them analyze and specify their own privacy policies. We may give Chief Privacy Officers or industry collaborators permissions to view some attributes of the data (e.g., only the goal descriptions but not the classifications and statistical data). In short, there are a variety of access control requirements in the system, making it a sufficiently sophisticated system to analyze using our approach.

At the time of writing, we have completed the SRS document [24], the database design and most of the user interface design. We are currently implementing the tool's functionality. The SRS and the database design are the inputs for our first case study.

4.2. The Transnational Digital Government (TDG) project

The Transnational Digital Government (TDG) project, funded by the U.S. National Science Foundation (NSF)[3], is a collaborative research project involving researchers at seven universities as well as government agencies in three participating countries: U.S., Belize and Dominican Republic. The project's objective is to research advanced information technologies useful for rapid collection, dissemination, and exchange of information related to transnational border control. The system will collect and share immigration information. The prototype system is the source of this case study.

The security and privacy of immigration data is critical in this project. Secure remote sharing of sensitive data requires reliable access control mechanisms. RBAC and distributed trust management have been chosen to implement this functionality. Many organizations are involved in the transnational border control process, including remote border stations, governments, police departments, immigration departments, customs, etc. and information must flow among these organizations as well as across national borders. Different countries have different security and privacy policies and laws regulating the corresponding activities. Policy enforcement was important throughout the system's development to ensure compliance with the corresponding policies yet flexible enough to enforce changing policies in the future. Using our approach, we specified a set of ACPs for the TDG project in collaboration with the TDG database team at the University of Florida

4.3. Efforts and results

Table 1 summarizes the efforts devoted to both case studies as well as the resulting artifacts. This paper's authors conducted both case studies. The first case study took a total of 15.33 person-hours, whereas the second case study took a total of 5.83 person-hours. The first case study took longer than the second one for two reasons. First, it served as a formative case study. At the time of the analysis, the methodology was still under development and our experiences helped us refine the approach throughout the first case study Second, the SPRAT SRS is more concrete and detailed than the TDG SRS, which is relatively high-level and stable, with minimal need for change; thus, no new requirements were created during the second case study and few requirements changed.

During both case studies, the analysts devoted a modest amount of time to training to ensure a common understanding of the analysis activities. The total effort devoted to training for each case study was 6% for the SPRAT and 17% for the TDG project. The steps that required the largest proportion of analysis effort were Steps 2 (67% for SPRAT and 74% for the TDG project) and Step 3 (26% for SPRAT and 8% for the TDG project).

[3] The TDG Project URL: http://www.acis.ufl.edu/transdg/

5. Analysis process and heuristics

We now describe the steps an analyst takes to conduct this kind of analysis using examples from both systems, demonstrating how the heuristics are applied.

Table 1. Summary of the two case studies

	SPRAT	TDG
No. of analysts	2	2
Total time efforts (person-hour)	15.33	5.83
No. of functional requirements	56	25
No. (percentage) of new requirements created	6 (10.7%)	0 (0%)
No. (percentage) of changed requirements	27 (48.2%)	4 (16%)
No. of access control rules derived	73	17
No. of inconsistencies between the SRS and the DB design	2	17

Step 1: Read the Introduction section of the SRS.

The SRS for both projects contains introductory material, which yielded a general understanding of the envisioned system, the stakeholders and the end users. It is important for system designers to understand the concerns of both end-users and stakeholders so they can control end-users' access to data accordingly.

Step 2: Scan the SRS to identify access control elements

Actors, actions, objects, etc. can be identified using traditional inquiry-driven analysis [33]. For each requirement in the SRS, we followed steps 2.1 through 2.6 (see Figure 2) to identify these elements. It is not necessary to follow these steps in sequential order, they merely serve as a checklist; an experienced analyst may perform some of these steps in parallel.

Step 2.1 *Identify objects that need to be protected.*

Objects are data to which access needs to be restricted. We present three general heuristics (H1, H2 and H3) for identifying these objects:

H1: Objects are nouns that can typically be identified by looking at the nouns that follow verbs.

Consider the following SPRAT requirement:

`FR-GSM-3: The system shall allow analysts to classify goals.`

The noun "goals" follows the verb "classify", thus "goal" is tagged as an object. Note that because not every noun is an object, the following heuristic is used to distinguish access-related objects from other objects.

H2: Objects are system resources that should only be accessed by authorized actors.

In the case of FR-GSM-3 above, heuristic H2 ensures "goals" is the identified object to be protected instead of "analyst".

H3: Every object identified in the SRS should also appear in the database design.

Heuristic H3 forces analysts to ensure that the requirements and database design are consistent with one another. In both case studies, H3 was instrumental in helping us identify missing data fields in the database design. Consider the following TDG requirement:

`2.3.1: The system shall allow border immigration agents to determine if the traveler is on the "watch list".`

This requirement was annotated with stakeholder comments on what data is contained in the "watch list". Three of these items were missing in the database schema: gender, the reasons for a person's name being on the watch list, and actions to be taken if person whose name appears on the watch list is encountered at a border station. As shown in Table 1, we identified 17 inconsistencies between the TDG SRS and the corresponding database designs. Clearly H3 allowed us to correct the database designs early on, preventing possible costly changes that may not have been identified until well into the development lifecycle.

Step 2.2 *Identify responsible actors and possible actions on the object.*

Although these heuristics are not comprehensive, we discuss some of the most helpful ones below.

H4: If a requirement is stated as "The system shall allow <someone> to <do something>", then the actor is <someone> and the action is <do>.

Consider the following TDG requirement:

`2.5.2: The system shall allow border immigration agents to create a new record in the "watch list".`

The actor is *border immigration agent* and the action is *create*.

H7: When requirements are vague or when it is impossible to directly identify any actor/action, scenario analysis can be used to elaborate the requirements.

Consider the following SPRAT requirement:

`FR-PM-3: The system shall support multi-user analyst results comparison.`

This requirement was so ambiguous that without clarification it is impossible to understand what it means. Scenario analysis enabled us to identify the main events in this scenario as follows [22]:

`Event 1: Analysts classify goals independently according to predefined categories.`

`Event 2: Project managers select analysts whose classification results they wish to compare.`

`Event 3: The system shall display those goals that are classified differently and how they are different (e.g., by showing the different categories)`

This analysis yields the following actors and actions, which are of interest for specifying ACPs:

Actor: Analyst *Action:* classify
Actor: Project Manager *Action:* request
Actor: Project Manager *Action:* view

Step 2.3 *Document reasons why an actor performs an action, or accesses an object, as purposes.*

This step is necessary only when the purpose of an access request affects grant/deny decisions. Purpose is often used in the context of privacy protection. Example purposes include telemarketing, payment, research and development. If the purpose of an access needs to be considered when making grant/deny decisions, we specify the purpose in the *condition* part of an AC rule as follows [22]:

```
action.BusinessPurpose << object.DataPurpose
```

The symbol << means business purpose is contained in data purpose. Consider the following example. A particular piece of data (e.g., credit card information) is supposed to be used only for payment (data purpose). Given the condition in the ACP rule above, a data access request will be evaluated by an enforcement engine (to either grant or deny access); the business purpose of this access will be checked against the requested object's data purpose — payment.

Step 2.4 *Identify conditions under which an actor is allowed to perform an action on an object.*

Again, consider SPRAT requirement FR-PM-3 mentioned previously. If a user assumes the role of *Project Manager* and *Analyst* at the same time, as an analyst, he can classify goals; as a project manager, he can view other analysts' classification results. However, access to the information (classification results) is withheld until he is finished with his own classification of the goals. This condition must be satisfied before a Project Manager can view the classification results. We document this rule as follows:

```
IF Role (user, Project Manager) = TRUE AND
        Role (user, Analyst) = TRUE AND
        user.scheduledToClassify = TRUE AND
        user.classifyingFinished = FALSE

THEN viewClassificationResults = DENY
```

Step 2.5 *Identify obligations that the actor or system must fulfill if an access request is granted.*

Although no obligations were identified for either system, this step is still necessary because in other systems obligations may be identified.

It is important to document the results of the above steps. In both case studies, we marked the elements on the SRS printouts using color coding and then documented this information in an access control matrix in preparation for Step 3. We are developing the SPRAT to support this process. Recall that traceability is a primary design principle for this approach, when analysts document analysis results, they must also document from which requirement the rules were derived. With tool support, the traceability between requirements and access control policies will be maintained more easily.

Analysts must make design decisions during the ACP specification process. For example, the SPRAT SRS clearly states that the system shall support four system access levels: administrator, project manager, analyst and guest. Given that access to the system is restricted to these four levels (or roles), RBAC seems suitable for implementing access control. In our analyses, we made additional design decisions. For example, instead of building a role hierarchy, we ensured that the privileges assigned to each role never overlap.

During Step 2, analysts produce a set of candidate AC rules that are used in Step 3 to specify ACPs.

Step 3: Refine AC rules into ACPs

Recall that an access control *policy* is comprised of a set of access control *rules*. The access control rules derived from different requirements could be redundant or in conflict with each other. Thus, we have to reconcile the different rules identified in Step 2 and specify the collective privileges that should be assigned to each actor. We conducted the following analysis for each actor:

Step 3.1 *Sort the AC rules according to objects (e.g., goals, scenarios, requirements).*

Grouping rules according to objects allows analysts to identify redundant and conflicting rules more easily as described in Steps 3.2 and 3.3. In the SPRAT, for example, the actions that an analyst can perform on a goal were specified as follows:

Table 2. Actions for *Analyst* on *Goals*

Actor	Action	Object
Analyst	Add	Goals
Analyst	Delete	Goals
Analyst	Update	Goals
Analyst	View	Goals
Analyst	Search	Goals
Analyst	Classify	Goals

Step 3.2 *Merge redundant AC rules.*

As common in requirements specification, synonymous words are often interchanged. In the SPRAT SRS, for example, "view elements of goals" encompasses "view contexts of goals" because context is a goal element. These two rules were merged, yielding: "view elements of goals".

Step 3.3 *Reconcile any conflicting AC rules.*

If the requirements from which AC rules were derived are in conflict with one another, the resulting AC rules may conflict as well. Thus, any conflicts must be resolved at this stage of the analysis. The requirements specifications for both systems had been thoroughly analyzed and any conflicts identified during our analysis were resolved immediately. Sorting rules according to actors and objects also helps analysts identify conflicts.

Step 3.4 *Merge existing rules based on access control design decisions and create new rules, if necessary.*

In the SPRAT, we decided to employ RBAC to control users' access to data. Based on this decision, we merged the three privileges for the *System Administrator* role (create *Project Managers*, create *Analysts* and create *Guests*) into: "create user account" and created a new privilege: "assign roles to users". In this way, the rules we specified are consistent with the design decision and more flexible than the candidate rules.

Step 3.5 *Merge all remaining AC rules into final ACPs*

The objective is to refine all the candidate rules into ACPs. The previous steps help eliminate redundant rules, reconcile conflicting rules, and refine rules according to decision decisions. Any remaining AC rules should be merged as ACPs. For example, in the SPRAT, we did not merge or create any privilege for the *Project Manager* role. Instead, we simply merged the AC rules into ACPs because these rules did not fit any of the cases described in Steps 3.2 through 3.4.

Step 3.6 *Document all design decisions.*

Recall that analysts make important design decisions during ACP specification. Flexible access control is often desirable. In the SPRAT, for example, *Project Manager* can specify what information within the system is accessible to *Guests* and how the information can be accessed. But, increasing flexibility may also increase access control implementation complexity as well as system development costs. Analysts must conscientiously make design decisions based on qualitative tradeoff analysis (e.g., between implementation complexity and access control flexibility). These design decisions must be documented.

The main artifacts produced by analysts during the ACP specification process for a given project are a set of ACPs, documented design decisions, as well as augmented requirements and database design specifications. As previously mentioned, we specify ACPs as a group of rules that contain five elements: <*subject, object, action, condition, obligation*>. Figure 3 portrays a SPRAT ACP, which contains two access control rules: a *Deny* rule and an *Allow* rule. In this example, the ACP subject is a role – Project Manager. The ACP object is GoalClassificationResults, which can be mapped to table *clOptions* in the database design. The ACP action is *View*. In the *Allow* rule, Project Manager is allowed to view GoalClassificationResults under normal circumstances.

Deny rule {

Subject:	Role (Project Manager)
Object:	GoalClassificationResults
Action:	View
Condition:	Role (user, Analyst) = TRUE AND user.scheduledToClassify = TRUE AND user.classifyingFinished = FALSE
Obligation:	Null

}

Allow rule {

Subject:	Role (Project Manager)
Object:	GoalClassificationResults
Action:	View
Condition:	Null
Obligation:	Null

}

Figure 3. An example ACP for the SPRAT

However, the ACP also specifies a Deny rule that restricts the Project Manager's ability to view GoalClassificationResults under a particular condition. Both rules will be evaluated by the access control enforcement engine to make grant/deny decisions when a data access request occurs.

6. Discussions and plans for future work

The access control policy specification approach presented in this paper is inherently inquiry-driven and iterative. As such, it helps analysts ensure that many of the ambiguities, inconsistencies, and conflicts that often plague requirements specifications, database designs and corresponding policies are eliminated to ensure consistency across all artifacts. Specifically, our approach improves the quality of the SRS and database design in several ways. First, it helps analysts clarify ambiguities in requirements specifications and maintain consistency across the SRS and DB design. Second, checking objects identified in the SRS with the database design forces analysts to ensure that those objects missing from the database design are included in the final design. In our case studies, we identified missing data fields in the initial database design, which we were able to rectify. Examining the database design also helps one gain a better understanding of the requirements. For example, the SPRAT E/R diagram provided a general understanding of how goals, scenarios, requirements, documents, etc. are linked to each other. This was not easy to glean from the SRS alone.

Our approach supports access control analysis and ACP specification by providing traceability between requirements, ACPs and design decisions. In the event of a change in a policy or requirement, our approach allows analysts to quickly locate the affected requirements or policies for subsequent modification. By ensuring consistency between ACP, requirements and database designs, our approach improves the quality of ACPs and helps bridge the gap between requirements and design.

In both case studies, we examined the functional and nonfunctional requirements, but all ACPs were actually derived from functional requirements. This may be because most access controls are related to system functionalities. Access control analysis entails investigating how to control end-users' interactions with data in the system — information that is usually described in the functional requirements. These two case studies suggest that analysts may only need to derive ACPs from functional requirements, but further validation is required to be certain.

The level of detail in an SRS affects the efficiency of our approach. The SPRAT SRS provides a more detailed functional description than the TDG SRS. In contrast, the requirements description in the TDG project is high-level and stable, yet somewhat ambiguous. We have tried to clarify the requirements but the results are less than satisfactory due to the distributed environment, large research team and the communication complexities that accompany any project of this size. As a result, we admittedly spent less time on the TDG case study and also

derived fewer AC policies. We plan to conduct a face-to-face meeting with all the TDG stakeholders in the Dominican Republic this October to further disambiguate the requirements and specify the remaining ACPs.

An AC rule can have various modes (e.g. permit/deny/oblige/refrain). To date, we have found the allow/deny rule to be sufficient for the systems we have evaluated. However, we plan to explore the efficacy of other modes to possibly extend our approach accordingly.

Our approach does have limitations. It focuses on protecting sensitive data in information systems. We have not examined access control in security kernels or function- and application-level access control. Additionally, our approach assumes that a database design is available; we have not investigated projects for which one is not available.

Case study research is valuable for forming initial insights and preliminary validation. To date, we have conducted two case studies using our approach. Both the SPRAT and TDG studies, involving real systems, have offered valuable lessons learned and insights for future research. However, the scope of both studies was limited. For example, an important part of specifying ACPs for RBAC systems is to define roles. The roles were already defined in the SRS for both systems. We will continue to evaluate the approach's overall effectiveness and also plan to conduct empirical studies with software engineering students and practitioners.

Finally, we acknowledge that access control requirements not only come from the software system itself, but also from other sources, such as enterprise-wide corporate policies and legal requirements. These kinds of documents are inherently different from the kinds of documents we have examined to date. They are less specific and more ambiguous than requirements specifications. Deriving ACPs from these sources remains a challenge that we plan to explore in the future.

Acknowledgements

NSF Digital Government Grant #0131886 and NSF ITR Grant #0325269 supported this work. We thank the members of the Transnational Digital Government project for their collaboration as well as Calvin Powers, Jonathan Moffett, Ting Yu and the NCSU ThePrivacyPlace.Org reading group for their comments on this paper.

References

[1] A.I. Antón, J.B. Earp, D. Bolchini, Q. He, C. Jensen and W. Stufflebeam. Financial Privacy Policies and the Need for Standardization, *IEEE Security & Privacy*, Vol. 2 (2), pp. 36-45, 2004.

[2] K. Beznosov. Requirements for Access Control: US Healthcare Domain, *Proc. of the 3rd ACM Workshop on Role-Based Access Control*, pp. 43, 1998.

[3] K.J. Biba. Integrity considerations for secure computer systems, *Technical Report MTR-3153*, Rev. 1, MITRE Corporation, 1977.

[4] C. Bettini, S. Jajodia, S. Wang, D. Wijesekera, Provisions and obligations in policy rule management and security applications, *Proc. of the 28th Int'l Conf. on Very Large Data Bases (VLDB'02)*, pp. 502-513, 2002.

[5] D.E. Bell and L.J. LaPadula. Secure computer systems: Mathematical foundations, *Technical Report MTR-2547*, Vol. 1, MITRE Corporation, 1973.

[6] D.E. Bell and L.J. LaPadula. Secure computer system: Unified exposition and multics interpretation, *Technical Report MTR-2997*, Rev. 1, MITRE Corporation, 1976.

[7] R. Crook, D. Ince, and B. Nuseibeh. Modelling Access Policies Using Roles in Requirements Engineering, *Information and Software Technology*, 45 (14), pp. 979-991, Elsevier, 2003.

[8] E.J. Coyne. Role Engineering, *Proc. of the 1st ACM Workshop on Role-Based Access Control (RBAC'96)*, pp. 15-16, 1996.

[9] R. Chandramouli and R. Sandhu. Role Based Access Control Features in Commercial Database Management Systems, *Proc. of the 21st National Information Systems Security Conference*, 1998.

[10] N.C. Damianou. A Policy Framework for Management of Distributed Systems, *PhD Thesis*, Imperial College, London, 2002.

[11] D. E. Denning and P. J. Denning, *Cryptography and Data Security*, Addison-Wesley, 1982.

[12] D.E. Denning. A Lattice Model of Secure Information Flow, *Communications of the ACM*, 19 (5), pp. 236-243, 1976.

[13] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-Directed Requirements Acquisition, *Science of Computer Programming*, 20: 3-50, 1993.

[14] E.B. Fernandez and J.C. Hawkins. Determining Role Rights from Use Cases, *Proc. of the 2nd ACM Workshop on Role-Based Access Control*, pp. 121-125, 1997.

[15] S. Fischer-Hübner. IT-Security and Privacy, *Lecture Notes in Computer Science 1958 (LNCS 1958)*, Springer-Verlag, 2001.

[16] D.F. Ferraiolo, D.R. Kuhn, and R. Chandramouli. *Role-Based Access Control*, Artech House computer security series, 2003.

[17] I. Fundulaki and M. Marx. Specifying access control policies for XML documents with XPath, *Proc. of the 9th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pp. 61-69, 2004.

[18] P.-J. Fontaine. Goal-Oriented Elaboration of Security Requirements, *Project Dissertation*, Université Catholique de Louvain, Belgium, 2001.

[19] G.S. Graham and P.J. Denning. Protection – principles and practice, *Proc. Spring Jt. Computer Conference*, Vol. 40, pp. 417-429, AFIPS Press, 1972.

[20] O. Gotel and A. Finkelstein. An Analysis of the Requirements Traceability Problem. *Int'l Conf. on Requirements Engineering (ICRE'95),* pp. 94-101, Colorado Springs, Colorado, USA, April 1994.

[21] O. Gotel and A. Finkelstein. Contribution Structures. In *Proc. 2nd Int'l. Symp. on Requirements Engineering (RE'95),* pp. 100-107, York, UK, March 1995.

[22] Q. He and A.I. Antón. A Framework for Modeling Privacy Requirements in Role Engineering, *Proc. of the 9th Int'l Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'03)*, pp. 137-146, 2003.

[23] M.H. Harrison, W.L. Ruzzo, and J.D. Ullman. Protection in operating systems, *Communications of the ACM*, 19 (8), pp. 461-471, 1976.

[24] N. Jain, A.I. Antón, W.H. Stufflebeam, and Q. He. Security and Privacy Requirements Analysis Tool (SPRAT) Software Requirements Specification Version 2.00, *NCSU Computer Science Technical Report TR-2004-7*, April 9, 2004.

[25] S. Jajodia, P. Samarati, M.L. Sapino, V.S. Subrahmanian. Flexible support for multiple access control policies, *ACM Transactions on Database Systems (TODS)*, 26 (2), pp. 214-260, 2001.

[26] G. Karjoth and M. Schunter. A Privacy Policy Model for Enterprises, *Proc. of the 15th IEEE Computer Security Foundations Workshop*, pp. 271-281, 2002.

[27] B.W. Lampson. Protection, *Proc. of the 5th Princeton Symposium on Information Science and Systems*, pp. 437-443, 1971. Reprinted in *ACM Operating Systems Review*, 8 (1), pp. 18-24, 1974.

[28] L. Liu, E. Yu and J. Mylopoulos. Security and Privacy Requirements Analysis within a Social Setting, *Proc. of the 11th Int'l Requirements Engineering Conference (RE'03)*, pp. 151-161, 2003.

[29] B.A. Nuseibeh and S.M. Easterbrook. Requirements Engineering: A Roadmap, In A.C.W. Finkelstein (ed.) *The Future of Software Engineering*. (Companion Volume to *Proc. of the 22nd Int'l Conf. on Software Engineering*). IEEE Computer Society Press, 2000.

[30] G. Neumann and M. Strembeck. A Scenario-driven Role Engineering Process for Functional RBAC Roles, *Proc. of the 7th ACM Symp. on Access Control Models and Technologies (SACMAT'02)*, pp. 33-42, 2002.

[31] *The Platform for Privacy Preferences 1.1 (P3P1.1) Specification*, The World Wide Web Consortium (W3C), 10 February 2004. http://www.w3.org/TR/2004/WD-P3P11-20040210/

[32] C.P. Pfleeger and C.L. Pfleeger. *Security in Computing*, 3rd ed., Prentice Hall, 2002.

[33] C. Potts, K. Takahashi and A.I. Antón. Inquiry-Based Requirements Analysis, *IEEE Software*, 11(2), pp. 21-32, March 1994.

[34] C. N. Ribeiro, A. Zuquete, P. Ferreira, and P. Guedes. SPL: An Access Control Language for Security Policies with Complex Constraints, *Proc. of Network and Distributed System Security Symposium (NDSS'01)*, pp. 89-107, 2001.

[35] W. Stufflebeam, A.I. Antón, and T.A. Alspaugh. SMaRT - Scenario Management and Requirements Tool, *Proc. of the 11th IEEE Int'l Requirements Engineering Conference*, pp. 351, 2003.

[36] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman. Role-Based Access Control Models, *IEEE Computer*, Vol. 29 (2), pp. 38–47, 1996.

[37] G. Schimpf. Role-Engineering Critical Success Factors for Enterprise Security Administration, *Proc. of the 16th Annual Computer Security Applications Conference (ACSAC'00)*, 2000.

[38] A. Schaad, J. Moffett, J. Jacob. The Role-Based Access Control System of a European Bank: A Case Study and Discussion, *Proc. of the 6th ACM Symp. on Access Control Models & Technologies (SACMAT'01)*, pp. 3-9, 2001.

[39] P. Samarati, S. De Capitani di Vimercati. Access Control: Policies, Models, and Mechanisms, *IFIP WG 1.7 Int'l School on Foundations of Security Analysis and Design (FOSAD 2000)*, LNCS 2171, pp. 137-196, 2001.

[40] E. Yu. Modeling Organizations for Information Systems Requirements Engineering, *Proc. of the 1st IEEE Int'l Symp. on Requirements Engineering*, pp. 34-41, 1993.