Automated Extraction of Security Policies from Natural-Language Software Documents

Xusheng Xiao[1] Amit Paradkar[2] Suresh Thummalapenta[3] Tao Xie[1]
[1]Dept. of Computer Science, North Carolina State University, Raleigh, NC, USA
[2]IBM T. J. Watson Research Center, Hawthorne, NY, USA
[3]IBM Research, Bangalore, India
[1]xxiao2@ncsu.edu, [2]paradkar@us.ibm.com, [3]surthumm@in.ibm.com, [1]xie@csc.ncsu.edu

ABSTRACT

Access Control Policies (ACP) specify which principals such as users have access to which resources. Ensuring the correctness and consistency of ACPs is crucial to prevent security vulnerabilities. However, in practice, ACPs are commonly written in Natural Language (NL) and buried in large documents such as requirements documents, not amenable for automated techniques to check for correctness and consistency. It is tedious to manually extract ACPs from these NL documents and validate NL functional requirements such as use cases against ACPs for detecting inconsistencies. To address these issues, we propose an approach, called Text2Policy, to automatically extract ACPs from NL software documents and resource-access information from NL scenario-based functional requirements. We conducted three evaluations on the collected ACP sentences from publicly available sources along with use cases from both open source and proprietary projects. The results show that Text2Policy effectively identifies ACP sentences with the precision of 88.7% and the recall of 89.4%, extracts ACP rules with the accuracy of 86.3%, and extracts action steps with the accuracy of 81.9%.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection — Access Control; D.2.1 [**Software Engineering**]: Requirements/Specifications

Keywords

Access control, natural language processing, requirements analysis

1. INTRODUCTION

Access control is one of the most fundamental and widely used privacy and security mechanisms. Access control is often governed by an Access Control Policy (ACP) [36] that includes a set of rules specifying which principals (such as users or processes) have access to which resources. ACPs are crucial in preventing security vulnerabilities, since decisions (such as *accept* or *deny*) on user requests are based on ACPs. In ACP practice, there exist two major issues that can result in serious consequences such as allowing an unauthorized user to access protected resources: incorrect specification of ACPs and incorrect enforcement of ACP specifications in the system implementation.

The first issue of incorrect specification of ACPs is primarily due to two reasons. First, ACPs contain a large number of complex rules to meet various security and privacy requirements. One way to ensure the correctness of such complex rules is to leverage systematic testing and verification approaches [20, 29] that accept ACPs in a form of formal specification. In practice, ACPs are commonly written in Natural Language (NL) and are supposed to be written in security requirements, a type of non-functional requirements. However, often ACPs are buried in NL documents such as requirement documents. For example, consider the following ACP sentence (i.e., sentence describing ACP rules) for iTrust [5, 41], an open source health-care application: *"The Health Care Personnel (HCP) does not have the ability to edit the patient's security question and password"*. This ACP sentence is not amenable for automated verification, requiring manual effort in extracting the ACP from this sentence into an enforceable format such as the eXtensible Access Control Markup Language (XACML) [3]. Second, NL software documents could be large in size, often consisting of hundreds or even thousands of sentences (iTrust consists of 37 use cases [23] with 448 use-case sentences), where a portion of the sentences describing ACPs (117 sentences in iTrust) are buried among other sentences. Thus, it is very tedious and error-prone to manually inspect these NL documents for identifying and extracting ACPs for policy modeling and specification.

The second issue of incorrect enforcement of ACP specifications is primarily due to the inherent gap between ACPs specified using domain concepts and the actual system implementation developed using programming concepts. Functional requirements, such as scenario-based requirements (e.g., use cases) that specify sequences of action steps[1], bridge the gap, since they describe functionalities to be implemented by developers using domain concepts. For example, an action step *"The patient chooses to view his or her access log."* in Use Case 8 of iTrust implies that the system shall have the functionality for patient (domain concepts) to view his or her access log. These action steps typically describe that actors (principals) access different resources for achieving some functionalities and help developers determine what system functionalities to implement. Therefore, policy authors can validate action steps against provided ACPs to detect inconsistencies of resource access, also helping the policy authors construct consistent ACPs for the system. In practice, manually inspecting large functional requirements to extract resource-access information is also labor-intensive and tedious. For example, a proprietary IBM enterprise application

[1]We use the term of an *action step* rather than *action* to distinguish the term from an *action* in the access control model described later.

(that we used in our evaluations) includes 659 use cases with 8,817 sentences.

In general, like other types of NL documents, NL requirements written in English are unstructured and can be ambiguous or include implicit information, posing significant challenges for Natural Language Processing (NLP). However, in software documents such as functional and non-functional requirements, ACP sentences (i.e., NL security requirements for describing ACP rules) tend to follow specific styles such as: *[subject] [can/cannot/is allowed to] [action] [resource]* for role-based ACPs [16]. For example, based on our manual inspection of 217 ACP sentences collected from the iTrust requirements and various security requirements in published articles and web sites [6], about 85% of the ACP sentences follow this style. Similarly, to provide communication values, functional requirements such as use cases are usually written in a relatively simple, consistent, and straightforward style [12, 24].

To tackle the problem, in this paper, we propose a novel approach, called Text2Policy, which adapts NLP techniques designed around a model (such as the ACP model and the action-step model) to automatically extract model instances from NL software documents and produce formal specifications. Our general approach consists of three main steps: (1) apply linguistic analysis to parse NL documents and annotate words and phrases in sentences from NL documents with semantic meanings, (2) construct model instances using annotated words and phrases in the sentences, and (3) transform these model instances into formal specifications.

Specifically, we provide techniques to concretize our general approach for extracting role-based ACPs and action steps from NL software documents and functional requirements, respectively. From the extracted ACPs, our approach automatically generates machine-enforceable ACPs in specification languages such as XACML. These ACPs can be used by automatic testing and verification approaches [20, 29] for checking policy correctness or serve as an initial version of ACPs for policy authors to improve. From each extracted action step, our approach automatically derives an access control request. An example request could be that a principal requests access to a resource with the expected permit or deny decision. Such derived requests can be used for automatic validation against specified or extracted ACPs for detecting inconsistencies.

This paper makes the following main contributions:

- A novel approach, called Text2Policy, which provides a general framework that incorporates syntactic and semantic NL analyses to extract model instances from NL software documents and produces formal specifications. Our work demonstrates that manual effort can be reduced with automated extraction of security policies from NL documents in a specific domain such as security requirements written in specific styles.

- Analysis techniques that concretize our general approach for extracting role-based ACP rules and action steps from NL documents and functional requirements (such as use cases), respectively.

- Three evaluations of Text2Policy on 37 iTrust [5, 41] use cases, 25 use cases from a module in a proprietary IBM enterprise application (in the financial domain), and the collected 115 ACP sentences from 18 publicly available sources (published papers and public web sites). The results show that (1) Text2Policy effectively identifies ACP sentences from 927 use-case sentences with the precision of 88.7% and the recall of 89.4%, (2) Text2Policy effectively extracts ACP rules from 241 ACP sentences with the accuracy of 86.3%, (3) Text2Policy effectively extracts action steps from 767 action-step sentences with the accuracy of 81.9%,[2] and (4) Text2Policy helps us identify a name inconsistency of iTrust use cases using the extracted ACP rules and action steps.

```
ACP-1: An HCP should not change a patient's account.
ACP-2: An HCP is disallowed to change a patient's
       account.
```

Figure 1: Example ACP sentences written in NL.

2. BACKGROUND AND EXAMPLES

In this section, we first introduce the background of the ACP model used for representing ACPs in our approach, and then describe the background of the action-step model used for representing action steps in our approach.

2.1 ACP Model and XACML

This section provides the background information about our ACP model and XACML.

2.1.1 ACP Model

An ACP consists of a set of ACP rules. A typical role-based ACP rule consists of four elements: subject, action, resource, and effect [3, 16]. Figure 1 shows two example ACP rules. The subject element describes a principal such as a user or process that may request to access resources (e.g., an *HCP* in ACP-1). The action element describes an action (e.g., *change* in ACP-1) that the principal may request to perform. The resource element describes the resource (e.g., *a patient's account* in ACP-1) to which access is restricted. A rule can have one of various effects (i.e., permit, deny, oblige, or refrain). In this paper, we focus on permit rules and deny rules (i.e., rules with permit or deny effects), which are commonly used in various software systems for granting or blocking accesses to protected resources. A *permit* rule allows a principal to access a resource, whereas a *deny* rule, such as ACP-1 and ACP-2, prevents a principal from accessing a resource.

2.1.2 XACML

The eXtensible Access Control Markup Language (XACML) [3] is an XML-based general-purpose language used to describe policies, requests, and responses for ACPs, recognized as a standard by the Organization for the Advancement of Structured Information Standards (OASIS). XACML is designed to replace application-specific and proprietary ACP languages, thus enabling communication among applications created by different application vendors.

In an application deployed with XACML-based access control, before a principal can perform an action on a particular resource, a Policy Enforcement Point (PEP) sends a request formulated in XACML to the Policy Decision Point (PDP) that stores principal-specific XACML ACP rules. The PDP determines whether the request should be permitted or denied by evaluating the policies whose subject, action, and resource elements match the request. Finally, the PDP formulates its decision in the XACML response language and sends it to the PEP, which enforces the decision.

Currently, XACML has been widely supported by all the main platform vendors and extensively used in a variety of applications [28]. Recent research also provides systematic testing and verification approaches [20,29] for ensuring the correct specification of XACML rules. There also exist XACML-based research tools used in vari-

[2]The evaluation artifacts and detailed results of the iTrust use cases and the collected ACP sentences are publicly available on our project web site [6].

```
AS-1: An HCP creates an account.
AS-2: He edits the account.
AS-3: The system updates the account.
AS-4: The system displays the updated account.
```

Figure 2: An example use case.

ous agencies/labs and companies [21]. Thus, we choose XACML as the formal specification to model ACP.

2.2 Action-Step Model

Use cases [22] are scenario-based requirements specifications that consist of sequences of action steps for illustrating behaviors of software systems. These action steps describe how actors interact with software systems for exchanging information. Actors are entities outside software systems (such as users) that interact with the systems by providing input to the systems (e.g., in Action Step AS-2 shown in Figure 2) or receiving output from the systems (e.g., in AS-4 shown in Figure 2). Since action steps describe how actors access or update information (resources) of the systems, each action step can be considered to encode an access control request that an actor requests to access the resources and expect the request to be permitted. Using the access control requests with expected permit decisions derived from action steps, we can automatically validate such requests with expected decisions against specified or extracted ACPs to detect inconsistencies.

We represent the contents of use cases (sequences of action steps) in a formal representation. The content of a NL use case contains a list of sentences, each of which in turn contains one or more action steps initiated by some actor (e.g., an *HCP* in AS-1 shown in Figure 2). Each action step has an action associated with a classification, such as the INPUT classification for the action of providing information (e.g., *edits* in AS-2 shown in Figure 2) and the OUTPUT classification for the action of receiving information (e.g., *display* in AS-4 shown in Figure 2). An action step is also associated to one or more actors and has a set of parameters. These parameters represent the resources created, modified, or used by the actions. In Figure 2, AS-2 shows a resource *account* that is modified.

3. CHALLENGES AND EXAMPLES

In this section, we first describe the technical challenges faced by ACP extraction and action-step extraction. We next use examples to illustrate how Text2Policy extracts ACPs and action steps from NL documents and NL use cases, respectively.

3.1 Technical Challenges

As a common technical challenge for both ACP extraction and action-step extraction, *TC1-Anaphora* refers to identifying and replacing pronouns with noun phrases based on the context. For example, the pronoun *he* in AS-2 shown in Figure 2 needs to be replaced with the *HCP* from AS-1. For ACP extraction, there are two unique technical challenges: (1) *TC2-Semantic-Structure Variance.* ACP-1 and ACP-2 in Figure 1 use different ways (semantic structures) to describe the same ACP rule; (2) *TC3-Negative-Meaning Implicitness*. An ACP sentence may contain negative expressions, such as ACP-1. Additionally, the verb in the sentence may have negative meaning, such as *disallow* in ACP-2. For action-step extraction, there are two unique challenges: (1) *TC4-Transitive Actor*. AS-3 implies that an *HCP* (the actor from AS-2) is the initiating actor of AS-3; (2) *TC5-Perspective Variance*. AS-4 implies that an *HCP views* the updated *account*, requiring a conversion to replace the actor and action of AS-4.

To address *TC1-Anaphora*, we adapt the technique *Anaphora Resolution*, specializing the anaphora algorithm introduced by Kennedy et al. [27] to identify and replace pronouns with noun phrases based on the context. To address *TC2-Semantic-Structure Variance*, we propose a technique, called *Semantic-Pattern Matching*, which uses different semantic patterns based on the grammatical functions (subject, main verb, and object) to match different semantic structures of ACP sentences. To address *TC3-Negative-Meaning Implicitness*, we propose an inference technique, called *Negative-Meaning Inference*, which infers negative meaning by using patterns to identify negative expressions and a domain dictionary to identify negative meaning of verbs. To address *TC4-Transitive Actor*, we propose an analysis technique, called *Actor-Flow Tracking*. This technique first tracks non-system actors in action steps. Later, when the analysis encounters action steps that have only system actors, it replaces system actors with tracked non-system actors. To address *TC5-Perspective Variance*, we propose an analysis technique, *Perspective Conversion*. This technique tracks non-system actors of action steps. Later when the analysis encounters action steps that have only system actors and output information from the system, it replaces the system actors with tracked non-system actors and replaces output actions with read actions (such as *view*).

```
<Policy PolicyId="2" RuleCombAlgId="...">
  <Target/>
  <Rule Effect="Deny" RuleId="rule-1">
    <Target>
      <Subjects><Subject>
        <SubjectMatch MatchId="string-equal">
          <AttrValue>HCP</AttrValue>
          <SubjectAttrDesignator AttrId="subject:role"/>
        </SubjectMatch></Subject></Subjects>
      <Resources><Resource>
        <ResourceMatch MatchId="string-equal">
          <AttrValue>patient.account</AttrValue>
          <ResourceAttrDesignator AttrId="resource-id"/>
        </ResourceMatch></Resource></Resources>
      <Actions><Action>
        <ActionMatch MatchId="string-equal">
          <AttrValue>UPDATE</AttrValue>
          <ActionAttriDesignator AttrId="action-id"/>
        </ActionMatch></Action></Actions></Target>
  </Rule></Policy>
```

Figure 3: Generated XACML ACP for ACP-2 in Figure 1.

3.2 Example of ACP Extraction

Text2Policy adapts NLP techniques that incorporate syntactic and semantic analyses to parse NL software documents, constructs ACP model instances, and produces formal specifications.

In particular, Text2Policy first applies shallow parsing [32] that annotates sentences with phrases, clauses, and grammatical functions of phrases, such as subject, main verb, and object. For example, the shallow-parsing component parses ACP-1 in Figure 1 as [subject: *An HCP*] [main verb group: *should not change*] [object: *a patient's account.*]. Text2Policy then uses the domain dictionary to associate verbs with pre-defined semantic classes. For example, in ACP-2, the domain dictionary is used to associate *change* with the UPDATE semantic class, and *disallow* with the NEGATIVE semantic class.

To determine whether a sentence describes an ACP rule (i.e., is an ACP sentence) and extract elements of subject, action, and resource, Text2Policy composes semantic patterns using the identified grammatical functions of phrases and clauses extracted by the shallow-parsing component. For example, ACP-1 can be matched by the semantic pattern *Modal Verb in Main Verb Group*, and the constructed model instance of ACP-1 is [Subject: *HCP*] [Action: *change - UPDATE*] [Resource: *patient.account*].

To infer the effect for an ACP rule, Text2Policy checks whether the corresponding sentence contains any negative expression and whether the main verb group is associated with the NEGATIVE

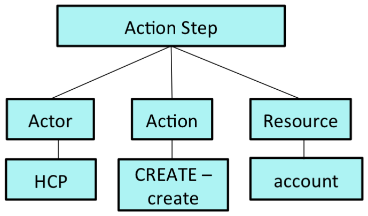

Figure 4: An example action step.

semantic class. For example, Text2Policy identifies the negative expression of *should not change* in ACP-1 and infers the effect of ACP-1 as *deny*.

Using the extracted ACP-model elements and the inferred effect, Text2Policy constructs an ACP model instance for each ACP sentence and generates ACP rules in XACML. Figure 3 shows the generated XACML ACP for ACP-2.

3.3 Example of Action-Step Extraction

Action-step extraction uses similar linguistic analyses as ACP extraction. First, the techniques of shallow parsing and domain dictionary are used to parse and annotate each sentence in use cases. Next, the technique of anaphora resolution is used to identify and replace pronouns (from the sentence) with the noun phrases based on the context. For example, *He* in AS-2 is replaced with *HCP*. Text2policy then uses a syntactic pattern to check whether the sentence has required elements (subject, main verb group, and object) for constructing an action step, and constructs a model instance if all the elements are found.

Consider the example use case shown in Figure 2. Since all sentences include the required elements, Text2policy constructs model instances of these action steps associated with actors (the *system*, *HCP*), action types representing the classification of the actions (e.g., the classification of *display* in AS-4 as OUTPUT), and parameters (the *account*). For example, the model instance of AS-1 is shown in Figure 4. In addition, since AS-3 and AS-4 have the *system* as the actor, Text2policy further applies the techniques for *TC4-Transitive Actor* and *TC5-Perspective Variance* on AS-3 and AS-4 to replace the actors and actions.

4. APPROACH

In this section, we describe our general approach for extracting model instances from NL documents and producing formal specification. Our approach consists of three main steps: Linguistic Analysis, Model-Instance Construction, and Transformation.

Figure 5 shows the overview of our approach. Our approach accepts NL software documents as input and applies linguistic analysis to parse the NL software documents and annotates their sentences with semantic meanings for words and phrases. Using the annotated sentences, our approach constructs model instances. Based on provided transformation rules, our approach transforms the model instances to formal specifications, which can be automatically checked for correctness and consistencies.

4.1 Linguistic Analysis

The linguistic-analysis component includes adapted NLP techniques that incorporate syntactic and semantic NL analyses to parse the NL software documents and annotate the words and phrases in the document sentences with semantic meaning. We next describe the common linguistic-analysis techniques used for both ACP extraction and action-step extraction, and describe the unique analysis techniques proposed for ACP extraction and action-step extraction, respectively.

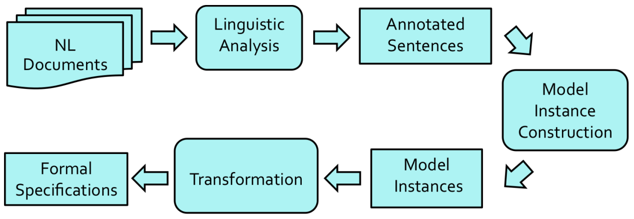

Figure 5: Overview of our approach.

4.1.1 *Common Linguistic-Analysis Techniques*

In this section, we describe the common linguistic-analysis techniques used in our general approach: shallow parsing and domain dictionary.

Shallow Parsing. Shallow parsing determines the syntactic structures of sentences in NL documents. Research [18, 40] has shown the efficiency of shallow parsing based on finite-state techniques and the effectiveness of using finite-state techniques for lexical lookup, morphological analysis, Part-Of-Speech (POS) determination, and phrase identification. Sinha et al.'s work [39] also shows that the shallow-parsing analysis is effective and efficient for semantic and discourse processing. Therefore, our approach chooses a shallow parser that is fully implemented as a cascade of several Finite-State Transducers (FSTs), described in detail by Boguraev [8].

In the shallow parser, an FST identifies phrases, clauses, and grammatical functions of phrases by recognizing patterns of POS tags and already identified phrases and clauses in the text. The lowest level of the cascade recognizes simple Noun Group (NP) and Verb Group (VG) grammars. For example, ACP-1 is parsed as [NP: *An HCP*] [VG: *should not change*] [NP: *patient's account.*]. Later stages of the cascade try to build complex phrases and identify clause boundaries based on patterns of already identified tokens and phrases. For example, *to change patient's account* in ACP-2 is recognized as a to-infinitive clause. The final set of FSTs marks grammatical functions such as subjects, main verb group, and objects. As an example, the shallow parser finally parses and annotates ACP-1 as [subject: *An HCP*] [main verb group: *should not change*] [object: *patient's account.*].

Domain Dictionary. The domain dictionary is used to associate verbs with pre-defined semantic classes. There are two benefits of associating verbs with semantic classes. The first benefit is to help address *TC3-Negative-Meaning Implicitness*. Consider ACP-2 shown in Figure 1. Without the NEGATIVE semantic class associated with the main verb group (*is disallowed*), our analysis would incorrectly infer the effect as *permit* instead of *deny*. The second benefit is to identify verb synonyms, such as *change* and *update*. During validation of action-step information against ACPs, our approach uses verb synonyms to match access requests (transformed from action steps) with an applicable ACP rule.

The domain dictionary is used to associate each verb entry with a semantic class. Besides the NEGATIVE class that we mentioned earlier, a verb entry can be associated with a semantic class that is a kind of operation [38, 39], e.g., OUTPUT (*view* or *display*) and UPDATE (*change* or *edit*). To achieve so, we populate the domain dictionary with an initial set of commonly used verb entries and their respective semantic classes. We then use WordNet [15], a large lexical database of English, to further expand the entries with their synonyms.

Currently, we implement the domain dictionary as an extensible and externalizable XML Blob and the content is populated manually. One major limitation of using an XML Blob is that unmatched verbs (i.e., ones without matched entries in the dictionary) are as-

Table 1: Identified subject, action, and resource elements in sentences matched with semantic patterns for ACP sentences.

Semantic Pattern	Examples
Modal Verb in Main Verb Group	An HCP $_{[subject]}$ **can view** $_{[action]}$ the patient's account $_{[resource]}$.
	An admin $_{[subject]}$ **should not update** $_{[action]}$ patient's account $_{[resource]}$.
Passive Voice followed by To-infinitive Phrase	An HCP $_{[subject]}$ **is disallowed to update** $_{[action]}$ patient's account $_{[resource]}$.
	An HCP $_{[subject]}$ **is allowed to view** $_{[action]}$ patient's account $_{[resource]}$.
Access Expression	An HCP $_{[subject]}$ **has read** $_{[action]}$ **access to** patient's account $_{[resource]}$.
	A patient's account $_{[resouce]}$ **is accessible** $_{[action]}$ **to** an HCP $_{[subject]}$.
Ability Expression	An HCP $_{[subject]}$ **is able to read** $_{[action]}$ patient's account $_{[resource]}$.
	An HCP $_{[subject]}$ **has the ability to read** $_{[action]}$ patient's account $_{[resource]}$.

signed with the UNCLASSIFIED semantic class. In future work, we plan to extend our technique to query WordNet dynamically when an unmatched verb or adjective is encountered. For example, by querying WordNet for synonyms, we can assign to an unmatched verb the semantic class of its most similar verb among its matched synonyms. Alternatively, we can assign to an unmatched verb the semantic class that is most common among the unmatched verb's k-nearest neighbors.

Anaphora Resolution. To address *TC1-Anaphora*, our approach includes the anaphora-resolution technique to identify and replace pronouns with the noun phrases that they refer to. To resolve anaphora encountered during use-case parsing, we adapt the anaphora algorithm introduced by Kennedy et al. [27] with an additional rule: a pronoun in the position of a subject is replaceable only by noun phrases that also appear as subjects of a previous sentence. As an example, *he* in AS-2 shown in Figure 2 is replaced by the *HCP*, the actor of AS-1.

4.1.2 ACP Linguistic Analysis

In this section, we describe unique linguistic-analysis techniques proposed for ACP extraction.

Semantic-Pattern Matching. To address *TC2-Semantic-Structure Variance*, we provide the technique of semantic-pattern matching to identify whether a sentence is an ACP sentence. We compose different semantic patterns based on the grammatical function of phrases identified by shallow parsing. These semantic patterns are more general and more accurate than templates written using low-level syntactical structures, such as POS tags [14]. Our approach uses this technique while identifying subject, action, and resource elements for an ACP rule.

Table 1 shows the semantic patterns used in our approach. The text in bold shows the part of a sentence that matches a given semantic pattern. The first pattern, *Modal Verb in Main Verb Group*, identifies sentences whose main verb contains a modal verb. This pattern can identify ACP-1 shown in Figure 1. The second pattern, *Passive Voice followed by To-infinitive Phrase*, identifies sentences whose main verb group is passive voice and is followed by a to-infinitive phrase. This pattern can identify ACP-2 shown in Figure 1. The third pattern, *Access Expression*, captures different ways of expressing that a principal can have access to a particular resource. The fourth pattern, *Ability Expression*, captures different ways of expressing that a principal has the ability to access a particular resource. Using the semantic patterns, our approach filters out NL-document sentences that do not match with any of these provided patterns.

Negative-Expression Identification. Negative expressions in sentences can be used to determine whether the sentences have negative meaning. To identify negative expressions in a sentence, our approach composes patterns to identify negative expressions in a subject and main verb group. For example, "*No HCP can edit patient's account.*" has *no* in the subject. As another example, "*An HCP can never edit patient's account.*" has *never* in the main verb group. ACP-1 in Figure 1 contains a negative expression in the main verb group. Our approach uses the negative-expression identification while inferring policy effect for an ACP rule.

4.1.3 Use-Case Linguistic Analysis

In this section, we describe a unique linguistic-analysis technique proposed for action-step extraction.

Syntactic-Pattern Matching. To identify whether a sentence is an action-step sentence (i.e., describing an action step), our approach includes the technique of syntactic-pattern matching that identifies sentences with syntactic elements (subject, main verb group, and object) required for constructing an action step. To improve precision in identifying sentences describing users accessing resources, our approach further checks whether the subject is a user of the system and whether the object is a resource defined in the system. For example, our approach ignores the sentence "The prescription list should include medication, the name of the doctor..." [5, 41], since its subject *prescription list* is not a user of the system. Moreover, our approach also uses the technique of negative-meaning inference (described later in Section 4.3.1) to filter out sentences that contain negative meaning, since these negative-meaning sentences tend not to describe action steps.

4.2 Model-Instance Construction

After our approach uses linguistic-analysis techniques to parse the input NL documents, words and phrases in the sentences of the NL documents are annotated with semantic meaning. For example, shallow parsing annotates phrases as subjects, main verb groups, and objects. To construct model instances from these sentences, our approach uses the annotated information of words and phrases to identify necessary elements for a given model.

4.2.1 ACP-Model Construction

To construct model instances for ACP rules, our approach identifies subject, action, resource elements based on the matched semantic patterns and infers the policy effect based on the presence or absence of negative expressions in sentences.

Model-Element Identification. Based on the matched semantic patterns, our approach identifies subject, action, resource elements from different syntactic structures in sentences.

Table 1 shows the identified subject, action, and resource elements (underlined words) in the sentences matched with semantic patterns. For a sentence that matches the first pattern, *Modal Verb in Main Verb Group*, our approach identifies the subject of the sentence as a subject element, the verb (not the modal verb) in the main verb group as an action element, and the object of the sentence as a resource element. For a sentence that matches the second pattern, *Passive Voice followed by To-infinitive Phrase*, our approach identifies the subject of the sentence as a subject element and identifies action and resource elements from the verb and object in the to-infinitive phrase, respectively. For the first example of the third pattern, *Access Expression*, our approach identifies the subject of the sentence as a subject element, the noun *read* in the main verb group as an action element, and the noun phrase *patient's account* in the prepositional phrase *to patient's account* as a resource element. For the second example of the third pattern, our approach identifies the subject *patient's account* as the resource element, the adjective *accessible* as an action, and the object *HCP* as the subject element. For the sentences that match the fourth pattern, our approach identifies the subject of the sentence as a subject element and identifies action and resource elements from the verb and object in the to-infinitive phrase, respectively.

Policy-Effect Inference. To address *TC3-Negative-Meaning Implicitness*, our approach includes the technique of negative-meaning inference. If an ACP sentence contains negative meaning, we infer the policy effect to be deny (permit otherwise). To infer whether a sentence has negative meaning, the technique of negative-meaning inference considers two factors: negative expression and negative-meaning words in the main verb group. Recall that negative expressions is identified using the technique of negative-expression identification in Section 4.1.2. ACP-1 in Figure 1 contains a negative expression in the main verb group. To determine whether there are negative meaning words in the main verb group, our approach checks the semantic class associated with the verb in the main verb group. If the semantic class is NEGATIVE, we consider the sentence has negative meaning. ACP-2 has a negative meaning word, *disallow*, in the main verb group, and therefore its inferred policy effect is deny.

Model-Instance Construction. Using the identified elements (subject, action, and resource) and inferred policy effect, our approach constructs an ACP-model instance for an ACP sentence. Moreover, our approach provides techniques to deal with a possessive noun phrase, such as *patient's account* or *the account of patient*. Our approach extracts the possessor as an entity and the possessed item as its property. As a complete example, the constructed model instance of ACP-2 is [Subject: *HCP*] [Action: *change - UPDATE*] [Resource: *patient.account.*] [Effect: *deny*]. Here the technique of domain dictionary associates the verb *change* with the semantic class UPDATE.

4.2.2 Action-Step-Model Construction

To construct model instances for action steps described in sentences, our approach identifies actor, action, and parameter elements based on the use-case patterns. Our approach includes two additional new techniques to address *TC4-Transitive Actor* and *TC5-Perspective Variance*.

Model-Element Identification. Our approach uses known patterns of use-case action steps to identify action, actor, and parameter elements for action steps. We devise these patterns based on industry use cases [39], iTrust use cases, and use cases collected from published articles [35]. One of the most used patterns is to identify the subject of a sentence as an actor element, the verb in the main verb group as an action element, and the object of the sentence as a parameter element. These patterns could be easily updated or extended based on the domain characteristics of the use cases for improving the precision of extracting actor, action, and parameter elements.

Model-Instance Construction. Using the identified actor, action, and parameter elements in a sentence, our approach constructs action-step model instances for action steps described in the sentence. For example, the model instance for *A patient views access log* is [Actor: *patient*] [Action: *view - READ*] [Parameter: *access log*]. Here the technique of domain dictionary associates the verb *view* with the semantic class READ.

Algorithm 1 Actor-Flow Tracking

```
Require: $ASs$ for action steps in a use case
 1: $trackedActor = NULL$
 2: for $AS$ in $ASs$ do
 3:    $Actors = getActors(AS)$
 4:    $onlySystemActor = TRUE$
 5:    for $actor$ in $Actors$ do
 6:       if $!isSystemActor(actor)$ then
 7:          $onlySystemActor = FALSE$
 8:          break
 9:       end if
10:    end for
11:    if $!onlySystemActor$ then
12:       $trackedActor = getNonSystemActor(Actors)$
13:       continue
14:    end if
15:    if $trackedActor\ != NULL$ then
16:       $replaceActors(AS, trackedActor)$
17:    end if
18: end for
```

Actor-Flow Tracking. To address *TC4-Transitive Actor*, we apply data-flow tracking on non-system actors of an action step. We consider subjects (such as the *system* in AS-3) with some specific names as system actors. Non-system actors can usually be obtained from the glossary of requirements documents. Algorithm 1 shows the Actor-Flow Tracking (AFT) algorithm.

We next illustrate the algorithm using the example shown in Figure 1. AFT first checks AS-1 and tracks the actor of AS-1 since its actor is a non-system actor (*HCP*) (satisfying the condition at Line 11). AFT then checks AS-2 and tracks the actor of AS-2 (*HCP*, replaced by anaphora resolution) since its actor is also *HCP*. When AFT checks AS-3, AFT finds that AS-3 has only the *system* as its actor (satisfying the condition at Line 15) and replaces the *system* with *HCP* as the actor of AS-3.

Perspective Conversion. To address *TC5-Perspective Variance*, we use a similar algorithm as AFT. The only difference is to replace the condition at Line 15 as $trackActor\ != NULL\ AND\ getActionType(AS) == OUTPUT$, and to replace the statement at Line 16 as $convertPerspective(AS, trackActor)$. Consider the same example shown in Figure 1. When the algorithm reaches AS-4, the tracked actor is *HCP*. Since AS-4 has *system* as its only subject and its action type is OUTPUT (*displays*), our approach converts AS-4 into *An HCP views the updated account* by replacing its actor elements with the tracked actors and its action element with a verb entry whose classification is READ in the domain dictionary, such as *view*. Such conversion helps our approach to correctly extract access requests from action steps.

4.3 Transformation

With the formal model of ACPs, our approach can use different transformation rules to transform model instances into formal specifications, such as XACML [3].

ACP Model. Currently, our approach supports the transformation of each ACP rule into an XACML policy rule. Our approach

Table 2: Metrics for addressing research questions.

RQ	Metrics
RQ1	$Precision = \frac{TP}{TP+FP}$, $Recall = \frac{TP}{TP+FN}$, $F_1\text{-}Score = \frac{2*Precision*Recall}{Precision+Recall}$ TP: True positives, FP: False positives, FN: False negatives
RQ2	$Accuracy = \frac{C}{T}$ C: Number of correct ACP rules extracted by Text2Policy T: Total number of ACP rules
RQ3	$Accuracy = \frac{C}{T}$ C: Number of correct action-step sentences extracted by Text2Policy T: Total number of action-step sentences

transforms subject, action, and resource elements as the corresponding subject, action, and resource sub-elements of the target element for an XACML policy rule. Our approach then assigns the value of the effect element to the value of the effect attribute of the XACML policy rule to complete the construction of an XACML policy rule. Figure 3 shows the extracted XACML rule of ACP-2. More examples can be found on our project web site [6]. With more transformation rules, our approach can easily transform the ACP model instances into other specification languages, such as EPAL [7].

Action-Step Model. Currently, our approach supports the transformation of each action step into an XACML request [3] with the expected permit decision. For each action step, our approach transforms actor, action, and parameter elements as subject, action, and resource elements of the request, respectively.

5. EVALUATIONS

In this section, we present three evaluations conducted to assess the effectiveness of Text2Policy. For our evaluations, we collected use cases from an open source project iTrust [5,41], 115 ACP sentences from 18 sources (published papers, public web sites, and iTrust), and 25 use cases from a module in a proprietary IBM enterprise application. We specifically seek to answer the following research questions:

- **RQ1**: How effectively does Text2Policy identify ACP sentences in NL documents?
- **RQ2**: How effectively does Text2Policy extract ACP rules from ACP sentences?
- **RQ3**: How effectively does Text2Policy extract action steps from action-step sentences (i.e., sentences describing action steps)?

Table 2 shows metrics used to address our research questions. To address RQ1, we used three metrics: precision, recall, and F$_1$-Score. The first row in Table 2 shows formulas for computing these metrics. In these formulas, TP represents the number of correct ACP rules identified by Text2Policy, whereas FP and FN represent the number of incorrect and missing ACP rules, respectively, identified by Text2Policy. To address RQ2 and RQ3, we used the accuracy metric shown in the second and third rows of Table 2, respectively.

5.1 Subjects and Evaluation Setup

In our evaluations, we used three categories of subjects for addressing the three research questions. First, we used 37 use cases from iTrust [5,41]. iTrust is an open source health-care application that provides various features such as maintaining medical history of patients, storing communications with doctors, identifying primary caregivers, and sharing satisfaction results. The requirements documents and source code of iTrust are publicly available on its web site. iTrust requirements specification has 37 use cases, 448 use-case sentences, 10 non-functional-requirement sentences, and 8 constraint sentences. The iTrust requirements specification also has a section, called Glossary, that describes the roles of users who interact with the system.

We preprocessed the iTrust use cases so that the format of the use cases can be processed by Text2Policy. In particular, we removed symbols (e.g., [E1] and [S1]) that cannot be parsed by our approach. We replaced some names with comments quoted in parenthesis. For example, when we see *A user (an LHCP or patient)*, we replaced *A user* with *an LHCP or patient*. We separated sentences by replacing / with *or*. We also separated long sentences that span more than 2 or 3 lines, since such style affects the precision of shallow parsing. The preprocessed documents of the iTrust use cases are available on our project web site [6].

Second, we collected 100 ACP sentences from 17 sources (published articles and public web sites). These ACP sentences and 117 NL ACP rules from the iTrust use cases are the subjects for our evaluation to address RQ2. The document that contains the collected ACP sentences and their original sources can be downloaded from our project web site [6].

Third, we used 25 use cases from a module in a proprietary IBM enterprise application. Due to confidentiality, we refer to this application as *IBMApp*. This module belongs to the financial domain.

We next discuss the results of our evaluations in terms of the effectiveness of Text2Policy in identifying ACP sentences and extracting ACP rules from NL documents and in extracting action steps from use cases.

5.2 RQ1: ACP-Sentence Identification

In this section, we address the research question RQ1 of how effectively Text2Policy identifies ACP sentences in NL documents. To address this question, we first manually inspected the use cases of iTrust to identify ACP sentences. We then applied Text2Policy to identify ACP sentences and compared those results with our results of manual inspection to identify the numbers of true positives, false positives, and false negatives. We further computed precision and recall values based on these numbers.

Among 448 use-case sentences in the iTrust use cases, we manually identified 117 ACP sentences. Among 479 use-case sentences in the *IBMApp* use cases, we manually identified 24 ACP sentences. We then manually classified these ACP sentences identified by Text2Policy as correct sentences and false positives, and manually identified false negatives.

Table 3 shows the results of RQ1 for both the subjects. Column "Subjects" lists the name of the subjects. Columns "# Sent." and "# ACP Sent." show the number of use-case sentences and the number of ACP sentences. Column "# Ident." shows the number of identified ACP sentences, and Columns "FP" and "FN" show the numbers of false positives and false negatives. Based on these numbers, Columns "$Prec$", "Rec", and "F_1" show the computed precision, recall, and F_1-score. For iTrust, the results show that Text2Policy identified 119 sentences with 16 false positives and 14

Table 3: Evaluation results of RQ1.

Subjects	# Sent.	# ACP Sent.	# Ident.	FP	FN	$Prec$	Rec	F_1
iTrust	448	117	119	16	14	86.6%	88.0%	87.3
IBMApp	479	24	23	0	1	100.0%	95.8%	97.9
Total	927	141	142	16	15	88.7%	89.4%	89.1

Table 4: Evaluation results of RQ2.

Subjects	# ACP Sent.	# Extracted	Accu.
iTrust	217	187	86.2%
IBMApp	24	21	87.5%
Total	241	208	86.3%

Table 5: Evaluation results of RQ3.

Subjects	# AS Sent.	# Extracted	Accu.
iTrust	312	258	82.7%
IBMApp	455	370	81.3%
Total	767	628	81.9%

false negatives. For *IBMApp*, Text2Policy identified 23 sentences with 0 false positive and 1 false negative. The results show that our semantic patterns help identify ACP sentences more precisely on the *IBMApp* use cases. One explanation could be that proprietary use cases are often of higher quality compared to open-source use cases and conform to simple grammatical patterns.

We first provide an example to describe how Text2Policy correctly identifies ACP sentences. One of the ACP sentences that Text2Policy correctly identifies ACP rules is "*HCPs can modify or delete the fields of the office visit information.*" [5, 41]. Our semantic pattern *Modal Verb in Main Verb Group* helps identify that the main verb contains the modal verb *can* and correctly identify the sentence as an ACP sentence.

We next provide some examples to describe how Text2Policy produces false positives and negatives. One false positive produced by Text2Policy is "*The instructions can contain numbers, characters...*" [5, 41], which matches the pattern *Modal Verb in Main Verb Group*. However, this sentence describes a requirement on password setting, instead of an ACP rule. These false positives can be reduced by expanding the domain dictionary to include commonly used nouns that are unlikely to be systems or system actors. The sentence that cannot be identified by Text2Policy is "*The LHCP can select a patient to obtain additional information about a patient.*" [5, 41]. Due to precision in parsing long phrases, the underlying shallow parser fails to identify *to obtain additional information about a patient* as a to-infinitive phrase, causing a false negative for our approach. These false negatives can be reduced by improving the underlying shallow parser using more training corpus in future work.

5.3 RQ2: Accuracy of ACP Extraction

In this section, we address the research question RQ2 of how effectively Text2Policy extracts ACP rules from ACP sentences. To address this question, we manually extracted ACP rules from these ACP sentences. We next applied Text2Policy and compared the results with our manually extracted results. We compute the accuracy of the ACP extraction using the number of ACP sentences from which Text2Policy correctly extracts ACPs and the total number of ACP sentences.

Table 4 shows the results of RQ2. Column "Subject" lists the name of the subjects. Columns "# ACP Sent." and "# Extracted" show the total number of ACP sentences and the number of ACP sentences from which Text2Policy correctly extracts ACPs. The statistics shown by these two columns are used to compute the accuracy shown in Column "Accu.". Among 217 ACP sentences of iTrust (including 117 from iTrust use cases), Text2Policy correctly extracts ACP rules from 187 ACP sentences, achieving the accuracy of 86.2%. Among 24 ACP sentences in the 25 use cases of *IBMApp*, Text2Policy correctly extracts ACP rules from 21 ACP sentences, achieving the accuracy of 87.5%.

We first provide an example to describe how Text2Policy correctly extracts some ACP rules. One of the sentences from which Text2Policy correctly extracts ACP rules is "*The administrator is not allowed through the system interface to delete an existing entry.*" [5, 41]. Our semantic pattern *Passive Voice followed by To-infinitive Phrase* helps correctly identify this ACP sentence, and correctly extract subject (*administrator*), action (*delete*), and resource (*an existing entry*) elements. Our technique of negative-meaning inference also correctly infers the policy effect to be *deny*.

We next provide examples to describe how Text2Policy fails to extract some ACP rules. One of the sentences from which Text2Policy cannot correctly extract ACP rules is "*Any subject with an e-mail name in the med.example.com domain can perform any action on any resource.*" [4]. The subject of this sentence *Any subject* is a noun phrase followed by two prepositional phrases (*with an e-mail name* and *in the med.example.com domain*). These two prepositional phrases constrain the subject *Any subject*, which is not correctly handled by our current implementation. Moreover, due to the imprecision in parsing long phrases, Text2Policy fails to extract some resources from ACP sentences. In future work, we plan to develop techniques to analyze the effects of prepositional phrases and long phrases for improving the accuracy of ACP extraction.

5.4 RQ3: Accuracy of Action-Step Extraction

In this section, we address the research question RQ3 of how effectively Text2Policy extracts action steps from action-step sentences. First, we manually extracted actions steps from these action-step sentences. We next used Text2Policy to automatically extract actions steps and compared the results with our manually extracted results. We computed the accuracy of the action-step extraction by using the number of correctly extracted action-step sentences and the total number of action-step sentences.

Table 5 shows the results of RQ3. Column "Subject" lists the name of the subjects. Columns "# AS Sent." and "# Extracted" show the total number of action-step sentences and the number of action-step sentences from which Text2Policy correctly extracts action steps. The statistics shown by these two columns are used to compute the accuracy shown in Column "Accu.". Among 312 action-step sentences in the iTrust use cases, Text2Policy correctly extracts action steps from 258 action-step sentences, resulting in an accuracy of 82.7%. Among 455 action-step sentences in the 25 use cases of *IBMApp*, Text2Policy correctly extracts action steps from 370 action-step sentences, resulting in an accuracy of 81.3%.

We next provide examples to describe how Text2Policy fails to extract action steps. One of the action-step sentences from which Text2Policy fails to extract action steps is "*The HCP must provide instructions, or else they cannot add the prescription.*" [5, 41]. The reason is that the current implementation of our approach does not handle the subordinate conjunctions *or else*. Another example sentence is "*The public health agent can send a fake email message* *to the adverse event reporter to gain more information about the report.*" [5,41]. For such long sentences with prepositional phrases *to the adverse event reporter to gain more information about the report* after the object of the sentence *a fake email message*, the underlying shallow parser of our approach cannot correctly identify the grammatical functions. We plan to study more use cases on health-care applications and improve the underlying shallow parser with more patterns to identify grammatical functions of action-step sentences.

5.5 Detected Inconsistency

Our approach validates the extracted access requests against the extracted ACPs. Although our approach does not detect violations of the extracted ACPs in our evaluations, our approach identifies a few action steps that do not match any extracted ACPs. To study why these action steps do not match any ACPs, we further apply union on the specifications of action steps to collect the information of what users perform what actions on what resources. From this information, we find that *editor*, one of the system users, is not matched with any subjects in the extracted ACPs. We then check the glossary of the iTrust requirements and the use-case diagram. We confirm that *editor* in fact refers to *HCP*, *admin*, and *all users* in use cases 1, 2, and 4, respectively. Such name inconsistencies can be easily identified by combining validation of ACP rules and using the union information of extracted action steps.

6. THREATS TO VALIDITY

The threats to external validity include the representativeness of the subjects and the underlying shallow parser used by the current implementation of our approach. To evaluate ACP extraction and action-step extraction from use cases, we applied our approach on 37 use cases of iTrust. The iTrust use cases were created based on the use cases in the U.S. Department of Health & Human Service (HHS) [2] and Office of the National Coordinator for Health Information Technology (ONC) [1], and evolved and revised by about 70 students and teaching assistants as well as instructors each semester since the iTrust requirements were initially created. Although the public availability and activeness make the iTrust use cases suitable for our subjects, we evaluated our approach only on these limited use cases. To reduce the threats, for the evaluation of ACP extraction, we further collected 100 ACP sentences from other 17 publicly available sources. Furthermore, we also applied our approach on 25 use cases of a module in a proprietary IBM enterprise application that belongs to the financial domain.

The threats to internal validity include human factors for determining correct identification of ACP sentences from NL software documents, correct extraction of ACP rules from these sentences, and correct extraction of action steps from use cases. In our evaluations, we inspected the whole subject documents and manually identified ACP sentences, and extracted ACPs and action steps as the comparison base in the evaluations. To reduce the human-factor threats, we did the extraction carefully and referred to existing ACPs and other use cases for determining correct identification of ACP sentences, and correct extraction of ACPs and action steps. These threats could be further reduced by involving two or more people who have experiences on ACPs to manually extract ACPs and action steps and integrating their manual-extraction results with our manual-extraction results as the comparison base.

7. DISCUSSION AND FUTURE WORK

In this section, we discuss applications and limitations of our current approach and propose directions for future work.

Construction of Complete ACPs. From the extracted ACPs, our approach automatically generates formal specifications of ACPs. These formal ACPs can assist the construction of complete ACPs in three ways: (1) these formal ACPs can be used to validate manually specified ACPs for identifying inconsistencies; (2) these formal ACPs can serve as an initial version of ACPs for policy authors to improve, greatly reducing manual effort in extracting ACPs from NL software documents; (3) combined with specified ACPs, these formal ACPs can be fed to automated ACP-verification approaches for checking correctness, such as static verification [20] and dynamic verification via access-request generation [29,30].

ACP Modelling in the Absence of Security Requirements. In the absence of security requirements, our approach can still provide a solution to assist policy authors to model ACPs for a system. Our approach first extracts deny ACPs and action steps from functional requirements. Besides deriving access requests from action steps, we can also derive a permit ACP rule from each action step. With the extracted and derived ACPs, policy authors have two ways to model ACPs: (1) the policy authors can apply the extracted deny ACPs and add a policy rule to permit all other accesses; (2) the policy authors can combine the extracted deny ACPs and the derived permit ACPs, and add a policy rule to deny all other accesses.

Cooperation Between Tool and Human. The extracted policies can serve as an initial version of ACPs for policy authors to improve, advocating cooperation between the tool and the user [42, 43]: the tool reports policies extracted with low confidence and the user can refine them to get better results. Currently, our implementation is built on an Eclipse-based IDE and can provide visual feedback of extracted policies, e.g., extracted subjects, actions, and resources. We plan to improve the IDE to better support the cooperation between the tool and the user. In addition, to improve the precision of the semantic analysis (such as anaphora resolution), we can apply ambiguity-analysis techniques [11,44] on the NL software documents to identify nocuous ambiguities, and ask the user to resolve the ambiguities before our approach is applied to extract policies.

ACP-Rule Ordering. Our current approach extracts ACP rules from sentences without considering the ordering of the rules. Doing so may cause security holes in the extracted ACP rules. We plan to study the extracted ACP rules and develop new techniques to extract ordering for the ACP rules.

Context-aware Analysis in Action-Step Extraction. A sequence of action steps may have several state transitions. The techniques of actor-flow tracking and perspective conversion in our approach partially address the context-aware analysis in action-step extraction. For example, a customer may not pay the order if he has not selected an order. We plan to develop techniques to deal with state transitions during action-step extraction.

Other Policy Models. In our evaluations, we encountered some ACP sentences that describe conditions for ACP rules. For example, the ACP sentence "*During the meeting phase, reviewer can read the scores for paper if reviewer has submitted a review for paper.*" [13] contains an if-condition to constrain the ACP rule. Without correct extraction of the condition, the produced specification of ACP rules is incomplete and requires policy authors to manually fix the incompleteness issue. Besides the issue of conditions, our current approach cannot handle multi-level models [17] or workflow models [37] for access control. We plan to extend our approach to support these new models and provide new semantic patterns for identifying new styles. In addition, our approach can be extended to support privacy policies, such as HIPAA privacy policies[3]. Supporting extraction of HIPPA policies requires more sophisticated semantic models to address new challenges, such as condition rules, rule combination, and rule ordering. We plan to investigate techniques to deal with new challenges of extracting HIPAA privacy policies.

8. RELATED WORK

Manual Extraction of ACPs from NL Documents. He and Anton [19] propose a manual approach, called Requirements-based Access Control Analysis and Policy Specification (ReCAPS), to extract ACPs from various NL documents, including requirements documents, design documents and database design, and security and privacy requirements. During the extraction, their approach also clarifies ambiguities in requirements documents and identifies inconsistencies among requirements documents and database design. Their objective is to derive a comprehensive set of ACP rules, similar to our approach. However, our approach adapts NLP techniques and provides new analysis techniques to automate the process of ACP extraction, while their approach is manual.

Template Matching. Etzioni et al. [14] propose an approach to extract lists of named entities found on the web using a set of patterns. Their approach is related to the ACP extraction of our approach, since both use patterns to extract information. However, their patterns are based on the low-level POS tags (such as NP and NPList), while our semantic patterns are based on grammatical functions of phases (such as subject, main verb group, and object). Our semantic patterns are more general and provide high precision in identifying ACP sentences as shown in our evaluations.

NLP to Analyze API Documents. Pandita et al. [34] propose an approach that analyzes the meta-data of API descriptions, programming keywords, and semantic patterns from POS tags to infer method specifications from API documents. Zhong et al. [45] propose an approach that builds action-resource pairs from API documents via NLP analysis based on machine learning, and infers automata for resources from action-resource pairs and class/interface hierarchies. Both of these approaches focus on parsing API documents, and use the specific characteristics of API documents to improve the NLP analysis. For example, different parts of API documents can be mapped to different parts of code structures, such as class/method names, return values, and parameter names. However, the contents of requirements documents usually cannot be mapped directly to code structures, thus making their approaches inappropriate on analyzing requirements documents. Moreover, these two approaches do not include techniques to address the unique challenges of ACP extraction and action-step extraction, such as *TC3-Negative-Meaning Implicitness*, *TC4-Transitive Actor*, and *TC5-Perspective Variance*.

NLP to Assist Privacy-Policy Authoring. The SPARCLE Policy Workbench [9, 10, 25, 26] employs the shallow-parsing technique [32] to parse privacy rules and extract the elements of privacy rules based on a pre-defined syntax. These elements are then used to form policies in a structured form, so that policy authors can review it and then produce policies in a machine-readable form, such as EPAL [7] and XACML [3, 33] with a privacy-policy profile. Michael et al. [31] propose an approach to map NL policy statements to an equivalent computational format suitable for further processing by a policy workbench. However, neither of these approaches can identify sentences describing a policy rule. These approaches parse all the input statements for policy extraction by assuming that the input statements are policy statements, while our approach identifies ACP sentences from requirements documents using semantic patterns. Both of these approaches provide simple templates to extract elements for constructing policy rules, while our approach provides more general semantic patterns. Additionally, their approaches cannot infer negative meaning of sentences.

Use-Case Analysis. Sinha et al. [38, 39] adapt NLP techniques to parse and represent use-case contents in use-case models. The extraction of use-case contents to formal models is similar to the action-step extraction in our approach. However, our approach focuses on extracting access requests for validation against specified and extracted ACPs and provides corresponding analysis techniques to address the *TC4-Transitive Actor* and *TC5-Perspective Variance* challenges.

[3]http://crypto.stanford.edu/privacy/HIPAA/

9. CONCLUSION

In this paper, we have proposed an approach, called Text2Policy, which extracts ACPs from NL software documents and produces formal specifications. Our approach incorporates syntactic and semantic NL analyses around models such as ACP and action-step models and extracts model instances from NL software documents. From the extracted ACPs, our approach automatically generates machine-enforceable ACPs (in formal languages such as XACML) that can be automatically checked for correctness. From the extracted action steps, our approach automatically extracts resource-access information, which can be used for automatic validation against specified or extracted ACPs for detecting inconsistencies. We conducted evaluations on iTrust use cases, ACP sentences collected from 18 sources, and 25 proprietary use cases. The evaluation results show that with customized NLP techniques, automated extraction of security policies from NL documents in a specific domain helps effectively reduce manual effort and assist policy construction and understanding.

Acknowledgment

This work is supported in part by NSF grants CCF-0845272, CCF-0915400, CNS-0958235, ARO grant W911NF-08-1-0443, an NSA Science of Security Lablet grant, and a NIST grant.

10. REFERENCES

[1] Office of the National Coordinator for Health Information Technology (ONC). http://www.hhs.gov/healthit/.

[2] U.S. department of Health & Human Service (HHS). http://www.hhs.gov/.

[3] eXtensible Access Control Markup Language (XACML), 2005. http://www.oasis-open.org/committees/xacml.

[4] eXtensible Access Control Markup Language (XACML) specification, 2005. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.

[5] iTrust: Role-based healthcare, 2008. http://agile.csc.ncsu.edu/iTrust/wiki/.

[6] Text2Policy, 2012. http://research.csc.ncsu.edu/ase/projects/text2policy/.

[7] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter. Enterprise privacy architecture language (EPAL 1.2), 2003. http://www.w3.org/Submission/EPAL/.

[8] B. K. Boguraev. Towards finite-state analysis of lexical cohesion. In *Proc. INTEX-3*, 2000.

[9] C. Brodie, C.-M. Karat, J. Karat, and J. Feng. Usable security and privacy: A case study of developing privacy management tools. In *Proc. SOUPS*, pages 35–43, 2005.

[10] C. A. Brodie, C.-M. Karat, and J. Karat. An empirical study of natural language parsing of privacy policy rules using the sparcle policy workbench. In *Proc. SOUPS*, pages 8–19, 2006.

[11] F. Chantree, B. Nuseibeh, A. de Roeck, and A. Willis. Identifying nocuous ambiguities in natural language requirements. In *Proc. RE*, pages 56–65, 2006.

[12] A. Cockburn. *Writing Effective Use Cases*. Addison-Wesley Longman Publishing Co., Inc., 1st edition, 2000.

[13] D. J. Dougherty, K. Fisler, and S. Krishnamurthi. Specifying and reasoning about dynamic access-control policies. In *Proc. IJCAR*, pages 632–646, 2006.

[14] O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artif. Intell.*, pages 91–134, 2005.

[15] C. Fellbaum, editor. *WordNet An Electronic Lexical Database*. The MIT Press, 1998.

[16] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *TISSEC*, 4(3):224–274, 2001.

[17] M. I. Gofman, R. Luo, J. He, Y. Zhang, and P. Yang. Incremental information flow analysis of role based access control. In *Security and Management*, pages 397–403, 2009.

[18] G. Grefenstette. Light parsing as finite state filtering. In A. Kornai, editor, *Extended finite state models of language*, pages 86–94. Cambridge University Press, 1999.

[19] Q. He and A. I. Antón. Requirements-based access Control Analysis and Policy Specification (ReCAPS). *Inf. Softw. Technol.*, 51(6):993–1009, 2009.

[20] V. C. Hu, D. R. Kuhn, T. Xie, and J. Hwang. Model checking for verification of mandatory access control models and properties. *IJSEKE*, 21(1):103–127, 2011.

[21] J. Hwang, T. Xie, V. C. Hu, and M. Altunay. ACPT: A tool for modeling and verifying access control policies. In *Proc. POLICY*, pages 40–43, 2010.

[22] I. Jacobson. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison Wesley Longman Publishing Co., Inc., 2004.

[23] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison Wesley Longman Publishing Co., Inc., 1992.

[24] D. Jagielska, P. Wernick, M. Wood, and S. Bennett. How natural is natural language?: How well do computer science students write use cases? In *Proc. OOPSLA*, pages 914–924, 2006.

[25] C.-M. Karat, J. Karat, C. Brodie, and J. Feng. Evaluating interfaces for privacy policy rule authoring. In *Proc. CHI*, pages 83–92, 2006.

[26] J. Karat, C.-M. Karat, C. Brodie, and J. Feng. Designing natural language and structured entry methods for privacy policy authoring. In *Proc. INTERACT*, pages 671–684, 2005.

[27] C. Kennedy. Anaphora for everyone: Pronominal anaphora resolution without a parser. In *Proc. COLING*, pages 113–118, 1996.

[28] A. X. Liu, F. Chen, J. Hwang, and T. Xie. XEngine: a fast and scalable XACML policy evaluation engine. In *Proc. SIGMETRICS*, pages 265–276, 2008.

[29] E. Martin, J. Hwang, T. Xie, and V. Hu. Assessing quality of policy properties in verification of access control policies. In *Proc. ACSAC*, pages 163–172, 2008.

[30] E. Martin and T. Xie. A fault model and mutation testing of access control policies. In *Proc. WWW*, pages 667–676, 2007.

[31] J. B. Michael, V. L. Ong, and N. C. Rowe. Natural-language processing support for developing policy-governed software systems. In *Proc. TOOLS*, pages 263–274, 2001.

[32] M. S. Neff, R. J. Byrd, and B. K. Boguraev. The talent system: Textract architecture and data model. *Nat. Lang. Eng.*, 10(3-4):307–326, 2004.

[33] OASIS. Privacy policy profile of XACML v2.0., 2005. http://docs.oasis-open.org/xacml/2.0/privateprofile/access_control-xacml-2.0-privacy_profile-specos.pdf.

[34] R. Pandita, X. Xiao, H. Zhong, T. Xie, S. Oney, and A. Paradkar. Inferring method specifications from natural language API descriptions. In *Proc. ICSE*, pages 815–825, 2012.

[35] C. Rolland and C. B. Achour. Guiding the construction of textual use case specifications. *Data Knowl. Eng.*, 25(1-2):125–160, 1998.

[36] P. Samarati and S. D. C. d. Vimercati. Access control: Policies, models, and mechanisms. In *Proc. FOSAD*, pages 137–196, 2001.

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

[38] A. Sinha, S. M. S. Jr., and A. Paradkar. Text2Test: Automated inspection of natural language use cases. In *Proc. ICST*, pages 155–164, 2010.

[39] A. Sinha, A. M. Paradkar, P. Kumanan, and B. Boguraev. A linguistic analysis engine for natural language use case description and its application to dependability analysis in industrial use cases. In *Proc. DSN*, pages 327–336, 2009.

[40] M. Stickel and M. Tyson. FASTUS: A cascaded finite-state transducer for extracting information from natural-language text. In *Proc. Finite-State Language Processing*, pages 383–406, 1997.

[41] L. Williams and Y. Shin. Work in progress: Exploring security and privacy concepts through the development and testing of the iTrust medical records system. In *Proc. FIE*, pages 30–31, 2006.

[42] X. Xiao, T. Xie, N. Tillmann, and J. de Halleux. Precise identification of problems for structural test generation. In *Proc. ICSE*, pages 611–620, 2011.

[43] T. Xie. Cooperative testing and analysis: Human-tool, tool-tool, and human-human cooperations to get work done. In *Proc. SCAM*, Keynote, 2012.

[44] H. Yang, A. de Roeck, V. Gervasi, A. Willis, and B. Nuseibeh. Extending nocuous ambiguity analysis for anaphora in natural language requirements. In *Proc. RE*, pages 25–34, 2010.

[45] H. Zhong, L. Zhang, T. Xie, and H. Mei. Inferring resource specifications from natural language API documentation. In *Proc. ASE*, pages 307–318, 2009.