RC25128 (W1103-095) March 16, 2011
Computer Science

Automated Extraction and Validation of Security Policies from Natural-Language Documents

Xusheng Xiao[1], Amit Paradkar[2], Tao Xie[1]

[1]Department of Computer Science
North Carolina State University
Raleigh, NC USA

[2]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598 USA

Automated Extraction and Validation of Security Policies from Natural-Language Documents

Xusheng Xiao[1] Amit Paradkar[2] Tao Xie[1]
[1]Dept. of Computer Science, North Carolina State University, Raleigh, NC, USA
[2]I.B.M. T. J. Watson Research Center, Hawthorne, NY, USA
[1]{xxiao2,txie}@ncsu.edu, [2]paradkar@us.ibm.com

ABSTRACT

As one of the most fundamental security mechanisms of resources, Access Control Policies (ACP) specify which principals such as users or processes have access to which resources. Ensuring the correct specification and enforcement 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 directly checkable for correctness. It is very tedious and error-prone to manually identify and 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 a novel approach, called Text2Policy, that automatically extracts ACPs from NL documents and extracts action steps from NL scenario-based functional requirements (such as use cases). From the extracted ACPs, Text2Policy automatically generates checkable ACPs in specification languages such as XACML. From the extracted action steps, Text2Policy automatically derives access control requests that can be validated against specified or extracted ACPs to detect inconsistencies. To assess the effectiveness of Text2Policy, we conduct three evaluations on the collected ACP sentences from 18 sources and 37 use cases from an open source project called iTrust (including 448 use-case sentences). The results show that Text2Policy effectively extracts ACPs from NL documents and action steps from use cases for detecting issues in the use cases.

1. INTRODUCTION

Access control is one of the most fundamental and widely used privacy and security mechanisms. Access control is governed by an access control policy (ACP) [26], which includes a set of rules to specify which principals such as users or processes have access to which resources. Since access decisions on requests are based on ACPs, ACPs that are not correctly specified can result in consequences such as allowing an unauthorized user to access protected resources. Moreover, given specified ACPs, the system implementation needs to correctly enforce these ACPs, otherwise could cause similar consequences as incorrect specification of ACPs. Thus, ensuring the correct specification and enforcement of ACPs is crucial to prevent security vulnerabilities.

Problems. Correctly specifying ACPs is an important and yet challenging task, since ACPs may contain a large number of rules and be very complex in order to meet various security and privacy requirements. To ensure the correct specification of ACPs, policy authors can apply approaches of systematic testing and verification [17,23] on ACPs, which require ACPs being formally specified. However, in practice, ACPs are commonly written in Natural Language (NL) and buried in NL documents such as requirements documents, e.g., *"The Health Care Personnel (HCP) does not have the ability to edit the patient's security question and password"* in iTrust requirements [5, 30]. These ACP sentences (i.e., sentences describing ACP rules) are not directly enforceable or checkable for correctness, requiring manual inspection of the NL documents for identifying ACP sentences and extracting ACPs from these sentences into enforceable formats, such as XACML (eXtensible Access Control Markup Language) [4]. In general, these NL documents could be large in size, often consisting of hundreds or even thousands of sentences (iTrust consists of 37 use cases with 448 use-case sentences), where only a small portion describes ACPs (10 sentences in iTrust). Thus, it is very tedious and error-prone to manually inspect these NL for identifying and extracting ACPs for policy modelling and specification.

Similarly, correctly enforcing ACPs is also an important and yet challenging task due to the gap between ACPs specified using domain concepts and system implementation developed using programming concepts. Functional requirements, such as scenario-based functional requirements (use cases [18]) that specify sequences of action steps[1], bridges the gap since it describes 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. As a result, we can validate such action steps against provided ACPs to detect inconsistencies of resource access specified in action steps and ACPs. Such inconsistency detection can help the policy authors or developer to address the problems of correct enforcement of ACPs. In large-size function requirements, there may be only a few action steps that could cause inconsistencies. Manually inspecting functional requirements to identify inconsistencies is also labor-intensive and tedious.

[1]To differentiate *action* in the access control model described later, we here use *action step* rather than *action*.

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

Figure 1: Example ACP sentences written in NL.

Proposed Approach. To reduce manual efforts in addressing the problems of correct ACP specification and enforcement, we propose a novel approach, called Text2Policy, which includes novel Natural Language Processing (NLP) techniques designed around model (such as the ACP model and action-step model) to automatically extract model instances from NL documents and produces formal specifications. Our general approach consists three main steps: (1) apply linguistic analysis to parse NL documents and annotate words and phrases in sentences from NL documents with semantic meaning; (2) construct model instances using the annotated words and phrases in the sentences; (3) transform these model instances into formal specifications. In this paper, we provide techniques to concretize our general approach to extract ACPs from NL documents and extract action steps from functional requirements. From the extracted ACPs, our approach automatically generates machine-enforceable ACPs in specification languages such as XACML, which can be used by automatic verification and testing approaches [17, 23] 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 that a principal requests to access a resource with the expected permit decisions. Such derived requests with the expected permit decisions can be used for automatic validation against specified or extracted ACPs for detecting inconsistencies. We next describe the technical challenges faced by ACP extraction and action-step extraction by using example ACP sentences in Figure 1 and a sequence of example action steps in Figure 2 and our proposed techniques to address these challenges.

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 Action Step 2 shown in Figure 2 needs to be replaced with *HCP* from Action Step 1. For ACP extraction, there are two unique technique challenges: (1) *TC2-Semantic Structure Variance.* ACP1 and ACP2 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 ACP1. Additionally, the verb in the sentence may have negative meaning, such as *disallow* in ACP2. For action-step extraction, there are two unique challenges: (1) *TC4-Transitive Actor.* Action Step 3 implies that *HCP* (actor from Action Step 2) is the initiating actor of Action Step 3; (2) *TC5-Perspective Variance.* Action Step 4 implies that *HCP views* the updated *account*, requiring a conversion to replace the actor and action of Action Step 4.

Proposed Techniques. To address *TC1-Anaphora*, we propose a new technique, called *Anaphora Resolution*, which adapts the anaphora algorithm introduced by Kennedy et. al [22] to identify and replace pronouns with noun phrases based on the context. To address *TC2-Semantic Structure Variance*, we propose a new technique, called *Semantic Pattern Matching*, which provides 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 a new 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 a new technique, called *Actor Flow Tracking*, which tracks non-system actors of action steps and replace system actors with tracked actors for action steps that have only system actors. To address *TC5-Perspective Variance*, we propose a new technique, called *Perspective Conversion*, which tracks non-system actors of action steps similar to *Actor Flow Tracking* and converts action steps that have only system actors and output information from system by replacing actors and actions of the action steps.

```
Action Step 1: A HCP creates an account.
Action Step 2: He edits the account.
Action Step 3: The system updates the account.
Action Step 4: The system displays the updated account.
```

Figure 2: An example use case.

This paper makes the following major contributions:

- A novel approach, called Text2Policy, which provides a general framework that incorporates syntactic and semantic NL analysis to extract model instances and produces formal specifications. Our approach is the first attempt to automatically extract ACP rules from NL documents and extract action-steps from functional requirements to assist correct specification and enforcement of ACPs.
- New techniques that concretize our general approach to extract ACP rules from NL documents.
- New techniques that concretize our general approach to extract action steps from functional requirements such as use cases.
- Three evaluations of Text2Policy on the iTrust [5, 30] use cases and the collected 115 ACP sentences from 18 sources. The results show that (1) Text2Policy effectively identifies 8 ACP sentences with no false positives and 2 false negatives from 37 use cases (448 sentences) of iTrust requirements; (2) Text2Policy effectively extracts ACP rules from 115 ACP sentences with the accuracy of 92.17%; (3) Text2Policy effectively extracts action steps from 438 action-step sentences in iTrust use cases with the accuracy of 84.47%. The evaluation artifacts and detailed results are publicly available on our project web site [6].

2. BACKGROUND

In this section, we first introduce the background of ACP model used for representing ACPs in our approach, and then describe the background of the action-step model (adapted from the use case meta-model [27, 28]) used for representing action steps in our approach.

2.1 ACP Model

An access control policy consists of a set of access control rules. A rule can have one of various effects (i.e., permit, deny, oblige, or refrain). In this paper, we focus on the permit and deny rules (i.e., rules with permit or deny effects). *Permit* rules allows a principal, such as a user or a process, to access a particular resource, while *deny* rules prevent a principal from accessing a particular resource.

A typical access control rule consists of four elements: subject, action, resource, and effect, as shown in Figure 3.

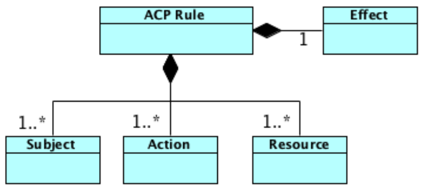

Figure 3: ACP model.

The subject element describes the principals such as users or processes that may access resources. The action element describes a simple action (e.g., view or udpate) or an abstract action (e.g., assign or approve) that the principals can perform. The resource element describes the resource (e.g., patient's password) to which access is restricted.

The eXtensible Access Control Markup Language (XACML) [4] is an XML-based general-purpose language used to describe policies, requests, and responses for access control policies, recognized as a standard by Organization for the Advancement of Structured Information Standards (OASIS).

To enforce ACP rules, before a principal can perform an action on a particular resource, a Policy Enforcement Point (PEP) sends a requests to the Policy Decision Point (PDP). The PDP makes the decision on whether the access can be granted by evaluating the ACP rules whose subject, action, and resource elements match the request. Based on the decision sent back by the PDP, the PEP allows or denies the access. Thus, to correctly enforce ACP rules in a system, PEPs need to be correctly deployed for sending access requests before accesses to protected resources.

2.2 Action-Step Model

Use cases [19] 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 (Action Step 2 in Figure 2) or receiving output from the systems (Action Step 4 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.

Functionally, use cases serve as requirements documents that helps developers determine which features of software systems to implement. A use case action step is usually implemented as a method or multiple methods among different modules in the code portions. For example, action 3 in Figure 2 may be mapped to a method named `updateAccount` that updates the account information. When we validate access control requests with expected permit decisions derived from action steps against specified or extracted ACPs, this mapping from action steps to system implementation can be used to locate PEPs in the system implementation, assisting the correct enforcement of ACPs. For example, if we find that a request (derived from an action step) has subject, action, and resource matched with an specified or extracted ACP rule, we then report that a PEP should be deployed for the action step. Using the reported action steps, developers can use the mapping to locate the portion of code in the system implementation to deploy PEPs.

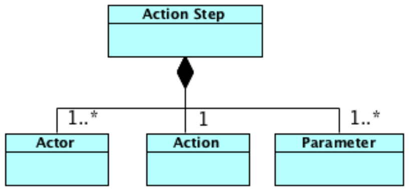

Figure 4: Action-Step model.

We represent the contents of use cases (sequences of action steps) in a formal representation, i.e., a structured model shown in Figure 4. 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., *HCP* in action step 1 shown in Figure 2). Each action step has an action associated with a classification, such as INPUT classification for the act of providing information (e.g., *edits* in action step 2 shown in Figure 2) and OUTPUT classification for the act of receiving information (e.g., *display* in action 4 shown in Figure 2). An action step is also associated to one or more actors and has a set of parameters. An parameter represents the resources created, modified, or used by the actions, such as *account* in action step 2 shown in Figure 2.

3. EXAMPLES

In this section, we present how Text2Policy extracts ACPs from NL documents and how Text2Policy extracts action steps from NL use cases.

3.1 Example of ACP Extraction

Text2Policy includes novel NLP techniques that incorporate syntatic analysis by using shallow parsing [24] and semantic analysis by using semantic pattern matching and domain dictionary to extract elements of subject, action, and resource, and infer policy effect.

The shallow parsing component in Text2Policy first uses a lexical processor to associate words with contextually appropriate part-of-speech (POS) information [7]. The shallow parsing component then uses a cascade of several finite-state transducers (FSTs) to identifies phrases, clauses, and grammatical functions of phrases by recognizing patterns of POS of tokens and already identified phrases and clauses in the text. Consider example ACPs shown in Figure 1. Through multiple levels of FSTs, the shallow parsing component parses ACP2 as [object: *A HCP*] [main verb group: *is disallowed*] [infinitive phrase: *to change patient's account.*]. The verb group phrase *is disallowed* is also identified by the shallow parsing component as passive voice.

To determine whether a sentence is describing 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, ACP2 can be matched by the semantic pattern *passive voice followed by to-infinitive phrase.* Based on this semantic pattern, Text2Policy extracts *HCP* as the subject element, *change* as the action element, *patient's account* as the resource element for an ACP rule. The domain dictionary used in our approach further associates the verb *change* in the action element with the UPDATE semantic

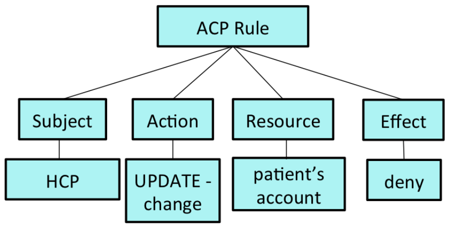

Figure 5: Example instance of ACP Model for ACP2 in Figure 1.

class.

To infer the effect for an ACP rule, Text2Policy first uses a domain dictionary to associate verbs with pre-defined semantic classes from an ACP sentence. For example, Text2Policy associates *is disallowed* in ACP2 with NEGATIVE semantic class and considers the effect of ACP2 as *deny*. Text2Policy then checks whether the ACP sentence contains any negative expression. For example, Text2Policy identifies the negative expression of *should not change* in ACP1 and considers the effect of ACP1 as *deny*.

Using the subject, action, and resource elements extracted by using the semantic patterns and the effect element inferred by checking semantic classes and negative expression, Text2Policy constructs an ACP model instance for each ACP sentence. Figure 5 shows the example model instance for ACP2.

3.2 Example of Action-Step Extraction

Text2Policy includes novel NLP techniques to extract action steps in the format of the model shown in Figure 4. Consider the example use case shown in Figure 2. Text2Policy first uses NLP techniques to parse and represent a use case as a sequence of action steps associated with actors (*system*, *HCP*), action types representing the classification of the actions (e.g., the classification of *display* in action step 4 as OUTPUT), and parameters (*account*). As a complete example, action step 1 is shown in Figure 6.

During the parsing, our new NLP techniques apply the anaphora resolution algorithm [22] to identify and replace pronouns with the noun phrases they refer to. For example, the anaphora resolution algorithm replaces *he* in Action Step 2 is replaced by *HCP* by the anaphora resolution technique.

As we discussed in the introduction, the actors of both Action Steps 3 and 4 are *system*. However, by inspecting the use case, we know that *HCP* would be the initiating actor of Action Step 3 and the receiving actor of Action Step 4, since *HCP* updates *account* and *the system* displays *account* for *HCP* to view. To address the challenges of transitive subject (e.g., in Action Step 3) and perspective variance (e.g., in Action Step 4), Text2Policy applies data flow analysis on actors in use case actions. Since action step 3 has *system* as its only initiating actor and action 1 has *non-system* (i.e., *HCP*) as its initiating actor, Text2Policy considers *HCP* as the actor for action step 3. Since Action Step 4 has *system* as its only initiating actor and the classification of its action type is OUTPUT, Text2Policy converts Action Step 4 as *HCP views the updated account*.

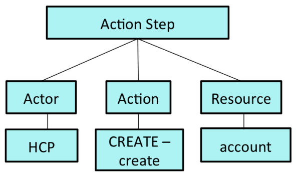

Figure 6: An example action step.

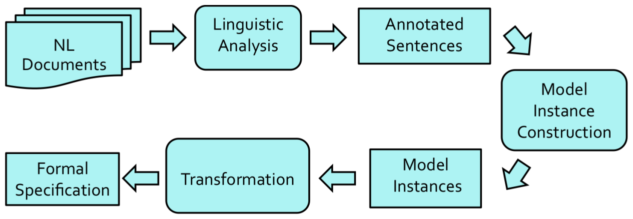

Figure 7: Overview of our approach.

4. APPROACH

In this section, we describe how our general approach automatically extracts model instances from NL documents and produces formal specification. In this paper, we concretize our approach by providing techniques to extract model instances of ACP rules from NL documents and extract model instances of action steps from use cases. Our approach consists of three main steps: Linguistic Analysis, Model Instance Construction, and Transformation.

4.1 Overview of Our Approach

Figure 7 shows the overview of our approach. Our approach accepts NL documents as input and applies linguistic analysis to parse the NL documents and annotate the sentences from the NL documents with semantic meaning for words and phrases. Using the annotated sentences, our approach construct model instances. Based on transformation rules, our approach transforms the model instances into formal specifications, which can be automatically checked for correctness and enforced in the deployed system.

4.2 Linguistic Analysis

The linguistic analysis component includes novel NLP techniques that incorporate syntactic and semantic NL analyses to parse the NL 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 techniques proposed for ACP extraction and action-step extraction, respectively.

4.2.1 *Common Linguistic Analysis Techniques*

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

Shallow Parsing. Shallow parsing determines the syntactic structures of sentences in NL documents. Research [15, 29] has shown the efficiency of shallow parsing based on finite-state techniques and the effectiveness of using finite-state methods for lexical lookup, morphological analysis, part-of-speech (POS) determination, and phrase identification. Our previous approach [28] 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 [9].

Semantic Pattern	Examples
Modal Verb in Main Verb Group	A HCP **can view** the patient's account. An admin **should not update** patient's password.
Passive Voice followed by To-infinitive Phrase	A HCP **is disallowed to update patient's password**. A HCP **is allowed to view patient's account**.
Access Expression	A HCP **has read access to** patient's account. An patient's account **is accessible to** A HCP.
Ability Expression	A HCP **is able to** read patient's account. A HCP **has the ability to** read patient's account.

Table 1: Semantic patterns for ACP sentences

In the shallow parser, an FST identifies phrases, clauses, and grammatical functions of phrases by recognizing patterns of POS of tokens 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, ACP1 is parsed as [NP: *A HCP*] [VG: *should not change*] [NP: *patient's account.*]. Later stages of the cascade try to build complex phrases, identify clause boundaries based on patterns of already identified tokens and phrases. For example, *to change patient's account* in ACP2 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 ACP1 as [subject: *A HCP*] [main verb group: *should not change*] [object: *patient's account.*].

Domain Dictionary. The domain dictionary associates 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 ACP2 shown in Figure 1. Without the semantic class of 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*. our approach uses verb synonym during validation of action-step information against ACPs, since our approach needs to match an ACP rule with the access requests transformed from action steps and the verbs used in the ACP rule and the action steps may be synonyms.

The domain dictionary associates 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 [27, 28], e.g., OUTPUT (view, displays) and UPDATE (change, edit). We populated the domain dictionary with an initial set of commonly used verb entries and their respective semantic classes. We then use WordNet [14], 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 base domain dictionary and the content is populated manually. One major limitation of static XML-Blob is that the domain dictionary can only assign the UNCLASSIFIED semantic class to unknown verbs. In future work, we plan to extend the domain dictionary to query WordNet dynamically when unknown verbs or adjectives are encountered. By querying WordNet for synonyms or antonyms of the currently known verbs, the domain dictionary can assign semantic classes to the unknown verbs using their most similar verbs' semantic classes. Alternatively, the domain dictionary can assign semantic classes of those already known verbs that belong to the k-Nearest neighbors of unknown verbs.

Anaphora Resolution. To address *TC1-Anaphora*, we provide the technique of anaphora resolution to identify and replace pronouns with the noun phrases that they refer to. Our approach uses this technique as part of the approach to identify actors for an action step. To resolve anaphora encountered during use-case parsing, we adapt the anaphora algorithm introduced by Kennedy et. al [22] with an additional rule: a pronoun in the position of an actor is replaceable only by noun phrases that also appear as actors of the previous action step. As an example, *he* in Action Step 2 shown in Figure 2 is replaced by *HCP*, the actor of Action Step 1.

4.2.2 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. Our approach uses this technique as part of the approach to identify subject, action, and resource elements for an ACP rule. To identify different semantic structures that describe ACP rules, semantic pattern matching uses their corresponding semantic patterns. These patterns are composed based on grammatical functions identified by shallow parsing. Thus, it is more general than patterns based on POS tags [13].

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. These semantic patterns identify ACP sentences. The first pattern, *Modal Verb in Main Verb Group*, identifies sentences whose main verb contains a modal verb. This pattern can identify ACP1 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 ACP2 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 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, *HCP can never edit patient's account.* has *never* in the main verb group. ACP1 in Figure 1 contains a negative expression in the main verb

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

Table 2: Identified subject, action, resource elements in sentences matched with semantic patterns for ACP sentences

group. Our approach uses the negative-expression identification as part of the approach to infer policy effect for an ACP rule.

4.2.3 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), we provide the technique of syntactic pattern matching that identifies sentences that have the syntactic elements (subject, main verb group, and object) required for constructing an action step. The sentences with missing subject or object are not considered as action-step sentences. Our approach also uses the technique of negative meaning inference (described later in Section 4.3.1) to filer out sentences that contain negative meaning, since these negative-meaning sentences tend not to describe action steps.

4.3 Model Instance Construction

After our approach applies linguistic analysis techniques to parse the input NL documents, our approach annotates words and phrases in the sentences of the NL documents are 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.

4.3.1 ACP Model Instance Construction

To construct model instances for ACP rules described in sentences, 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 the negative meaning of the sentences.

Subject, Action, and Resource Identification. Based on the matched semantic patterns, our approach identifies subject, action, resource elements from different syntactic structures in the sentences.

Table 2 shows the identified subject, action, resource elements 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. Our approach provides the technique of negative meaning inference to address *TC3-Negative-Meaning Implicitness* and infers policy effect for an ACP rule: if an ACP sentence contains negative meaning, we infer the policy effect to be deny (permit otherwise). To infer whether a sentence contains negative meaning, the technique of negative meaning inference considers two factors: negative expression and negative-meaning words in the main verb group. Our approach uses the technique of negative expression identification in the linguistic analysis component to identify negative expressions in a sentence. ACP1 in Figure 1 contains a negative expression in the main verb group. To determine whether there are negative meaning words in main verb group, our approach checks the semantic class associated with the verb in the main verb group. If the semantic class of the verb in the main verb group is NEGATIVE, we consider the sentence has negative meaning. ACP2 has a negative meaning word, *disallow*, in the main verb group, and therefore its inferred policy effect is deny.

ACP Model Instance Construction. Using the identified elements (subject, action, and resource) and inferred policy effect, our approach constructs an ACP model instance an ACP sentence. Figure 5 shows an example instance of the ACP model for ACP2. When our approach extracts ACP rules from functional requirements, our approach keeps only the constructed ACP model instances whose effect is deny, since negative-meaning sentences tend to reflect real ACPs.

4.3.2 Action-Step Model Instance 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. We further develop two additional new techniques to address *TC4-Transitive Actor* and *TC5-Perspective Variance*.

Actor, Action, and Parameter Identification. Our approach uses known patterns of use case action steps compiled in our previous approach [28] to identify action, actor, and parameter elements for action steps. We devise these patterns based on the subject use cases used in our previous approach and iTrust use cases. 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. For an example sentence *An patient views access log*, our approach identifies *patient* as an actor element, *view* as an action element, and *access log* 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.

Action-Step 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. Figure 6 shows an action-step model instance for the example sentence *An patient views access log.*

Actor Flow Tracking. To address *TC4-Transitive Actor*, we apply data flow tracking on non-system actors of an action step. Algorithm 4.1 shows the actor flow tracking (AFT) algorithm.

We next illustrate the algorithm using the example shown in Figure 1. AFT first checks Action Step 1 and tracks the actor of Action Step 1 since its actor is a non-system actor (*HCP*) (satisfying the condition at Line 11). AFT then checks Action Step 2 and tracks the actor of Action Step 2 (*HCP*, replaced by anaphora resolution) since its actor is also *HCP*. When AFT checks Action Step 3, AFT finds that Action Step 3 has only *system* as its actor (satisfying the condition at Line 15) and replaces *System* with *HCP* as the actor of Action Step 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 use $convertPerspective(AS, trackActor)$ as the replacement statement at Line 16. Using the same example shown in Figure 1. When the algorithm reaches Action Step 4, the tracked actor is *HCP*. Since Action Step 4 has *system* as its only subject and its action type is OUTPUT (*displays*), our approach converts action step 4 into *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*.

4.4 Transformation

With the formal model of ACPs, our approach can use different transformation rules to transform model instances into formal specifications, such as XACML [4].

Algorithm 4.1 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
```

ACP Model Instance Transformation. Currently, our approach supports the transformation of each ACP rule into an XACML policy rule [4]. Our approach 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. With more transformation rules, our approach can easily transform the ACP model instances into other specification languages, such as EPAL [8].

Action-Step Model Instance Transformation. Currently, our approach supports the transformation of each action step into an XACML request [4] with the expected permit decision. For each action step, our approach transforms the actor elements as subject elements of the request, the action elements as action elements of the request, and the parameter elements as resource elements of the request.

5. EXAMPLE APPLICATIONS

In this section, we describe several applications of extracted ACPs and extracted action steps in our approach.

Assisting Construction of Complete ACPs. From the extracted ACPs, our approach automatically generate formal specifications of ACPs. These formal ACPs can be used to validate manually specified ACPs for checking correctness and completeness. Additionally, these ACPs can serve as an initial version of ACPs for policy authors to improve, greatly reducing manual efforts in extracting ACPs from NL documents.

Validating Action Steps against Specified or Extracted ACPs. From the action steps extracted from functional requirements, our approach automatically derives access control requests (describing actors request to access resources) with the expected permit decisions. These access control requests can be automatically validated with the specified or extracted ACPs to detect inconsistencies. By inspecting the inconsistencies, policy authors and requirement analysts can fix either functional requirements or security requirements to resolve the inconsistencies.

Locating Policy Enforcement Points (PEP). In general, action steps can be mapped to one or more methods in the code portions of the system implementation. This mapping from action steps to the system implementation can be used to locate PEPs in the system implementation, assisting the correct enforcement of ACPs. For example, during validation of access control requests derived from action steps against specified or extracted ACP rules, we can identify the access control requests whose subject, action, and resource matched with one or more specified or extracted ACP rules. Developers can use these identified action steps to locate the portions of code in the system implementation to deploy PEPs.

Assisting 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 control requests from action steps, we can also derive a permit ACP rule from each action step. With the extracted ACPs and the permit ACPs, policy authors have two ways to model ACPs: (1) policy authors can apply the extracted deny ACPs and add a policy rule to permit all other accesses; (2) policy authors can combine the extracted deny ACPs and the derived permit ACPs, and add a policy rule to deny all other accesses.

6. EVALUATIONS

In this section, we discuss the three evaluations conducted to assess the effectiveness of Text2Policy. In our evaluations, we use use cases from an open source project iTrust [5, 30] and 115 ACP sentences from 18 sources (published papers, public websites, and iTrust), and 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)?

We next provide details of the metrics that we use in our evaluations. To address RQ1, we applied Text2Policy to identify ACP sentences (i.e., sentences describing ACP rules) in the use cases of iTrust and used the standard metrics of precision ($Prec$) and recall (Rec) to measure the accuracy of Text2Policy in identifying ACP sentences. The metrics of precision and recall are computed as $Prec = \frac{TP}{TP+FP}, Rec = \frac{TP}{TP+FN}$, where TP represents true positives, i.e., the number of correct ACP rules identified by Text2Policy, FP represents False Positives (FP), i.e., the number of incorrect ACP sentences identified by Text2Policy, and FN represents False Negative (FN), i.e., the number of real ACP sentences that are missed by Text2Policy. To address RQ2, we applied Text2Policy to extract ACP rules from ACP sentences. To measure the effectiveness, we measure the number of ACP rules correctly extracted by Text2Policy and compute the accuracy as $Accu = \frac{C}{T}$, where C represents the number of ACP rules correctly extracted by Text2Policy and T represents the total number of subject ACP rules. To address RQ3, we applied Text2Policy to extract action steps from action-step sentences (i.e., sentences describing action steps) of the iTrust use cases. To measure the effectiveness, we measure the number of sentences from which Text2Policy correctly extracts action steps and compute the accuracy as: $Accu = \frac{C}{T}$, where C represents the number of sentences from which Text2Policy correctly extracts action steps and T represents the total number of action-step sentences in the use cases of iTrust.

6.1 Subjects and Evaluation Setup

We use the use cases in iTrust [5,30] as the subject for RQ1 and RQ3. iTrust is an open source medical application that provides patients with a means to keep up with their medical history and records as well as communicate with their doctors, including selecting which doctors to be their primary caregiver, seeing and sharing satisfaction results, and other tasks. The requirements documents and source codes of iTrust are publicly available at its website. 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 (users) that interact with the system. The total lines of code (LOC) of iTrust implementation is 28,514, including 13,528 LOC for production code, 11,445 LOC for unit tests, and 3,541 LOC for httptests.

We preprocessed the iTrust use cases so that the format of the use cases can be processed by Text2Policy. In particular, we remove symbols (e.g., [E1] and [S1]) that cannot be parsed by our approach. We replace some names with comments quoted in parenthesis. For example, when we see *A user (an LHCP or patient)*, we replace *A user* with *an LHCP or patient*. We break down sentences by replacing / with *or*. We break down 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 website [6].

To evaluate the effectiveness of ACP extraction, we further collected 115 ACP sentences from 18 sources (published papers and public websites). These ACP sentences, including 10 NL ACP rules from the iTrust use cases, are the subjects for our evaluation to address RQ 2. The document that contains the collected ACP sentences and their original sources can be downloaded from our project website [6].

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.

6.2 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 measure the number of identified ACP sentences by Text2Policy, the number of false positives, and the number of false negatives generated by Text2Policy. We then compute the standard precision and recall using these values. To measure values for these metrics, we manually inspected the use cases of iTrust to identify ACP sentences, and applied Text2Policy to identify ACP sentences. We then manually classified these the ACP sentences identified by Text2Policy as correct sentences and false positives, and manually identified false negatives.

Among 448 use-case sentences in the iTrust use cases, we manually identified 10 ACP sentences. We applied Text2Policy on the iTrust use cases and Text2Policy identified 8 sentences with no false positives and 2 false negatives. Based on these numbers, the computed precision is 100% and the recall is 80%.

We next provide some examples to describe how Text2Policy produces false negatives. The sentence that cannot be identified by Text2Policy is "*The administrator is not allowed through the system interface to delete an existing entry or modify the appointment type name in an existing entry.*" [5, 30]. Since the prepositional phrase *through the system interface* appears just after the main verb group *is not allowed*, the underlying shallow parser that we use does not successfully identify the grammatical functions of the sentence, resulting in a false negative. The other sentence causing a false negative is "*The administrator is not allowed through the system interface to delete an existing entry or modify the reason ID number in an existing entry.*" [5,30], sharing the similar reason to the other false-negative sentence. In our future work, we plan to improve the precision of the underlying shallow parser by incorporating more general patterns.

6.3 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 measure the number of ACP sentences from which Text2Policy correctly extracts ACP rules. We manually extracted ACP rules from these ACP sentences and compared the manually extracted ACP rules with the ACP rules extracted by Text2Policy to determine whether the ACP rules extracted by Text2Policy are correct. Using the number of ACP sentences from which Text2Policy correctly extract ACPs and the total number of ACP sentences, we compute the accuracy of the ACP extraction.

Among 115 ACP sentences (including 10 from iTrust use cases), Text2Policy successfully extracted ACP rules from 106 ACP sentences. Based on these numbers, the accuracy of ACP extraction is 92.17%.

We first provide an example to describe how Text2Policy correctly extracts some ACP rules. One of the sentences from which Text2Policy correctly extract ACP rules is "The administrator is not allowed to delete an existing entry." [5, 30]. Our semantic pattern *Passive Voice followed by To-infinitive Phrase* helped 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 inferred 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.*" [3]. 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 of our approach. In our future work, we plan to provide techniques to analyze the effects of prepositional phrases for improving the accuracy of ACP extraction.

Another example sentence is "*A reviewer of a paper can resign the review of the paper, unless he has already appointed a sub-reviewer for the paper.*" [31]. This sentence includes a conditional clause starting with *unless*, which is not handled by the current implementation of our approach. In our future work, we plan to introduce new techniques to deal with such conditional expressions.

6.4 RQ3: Accuracy of Action-Step Extraction

In this section, we address the research question RQ3 of how effectively Text2Policy extracted action steps from action-step sentences . To address this question, we measure the number of action-step sentences from which Text2Policy correctly extract action steps. We manually extracted action steps from these action-step sentences and compare the manually extracted action steps with the action steps extracted by Text2Policy to determine whether the action steps extracted by Text2Policy are correct. Using the number of action-step sentences from which Text2Policy correctly extract action steps and the total number of action-step sentences, we compute the accuracy of the ACP extraction.

Among 412 action-step sentences, Text2Policy successfully extracted action steps from 348 action-step sentences. Based on these numbers, the accuracy of ACP extraction is 84.47%.

We next provide examples to describe how Text2Policy fails to extract action steps. One of the action-step sentences from which our approach failed to extract action steps is "*The HCP must provide instructions, or else they cannot add the prescription.*" [5, 30], since 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, 30]. 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 cannot correctly identify the grammatical functions. We plan to study more use cases on medical care applications, so that we can improve the underlying shallow parser with more patterns to identify grammatical functions of action-step sentences.

Using the specification of action steps, we applied union on the specifications of action steps to collect the information of what users perform what actions on what resources. From this information, we found that *editor*, one of the system users, were actually not described in the glossary of the requirements. We further checked the use-case diagram and confirmed that *editor* in Use Case 1 in fact refers to HCP, *editor* in Use Case 2 in fact refers to *admin*, and *editor* in Use Case 4 in fact refers to *all users*. Such name inconsistencies have been easily identified by using the union information of extracted action steps.

7. 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 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 of the iTrust use cases make the iTrust use cases suitable for our subjects, we evaluated our approach only on these limited use cases. To reduce the threat, for the evaluation of ACP extraction, we further collected 105 ACP sentences from other 17 sources. The threat of using only the iTrust use cases as subjects could be further reduced by using more use cases from other domains as evaluation subjects. The threats to internal validity include human factors for determining correct identification of ACP sentences from NL 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 evaluation comparison basics. 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 for our evaluation comparison base.

8. DISCUSSION AND FUTURE WORK

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

Conditions in ACP rules and Action Steps. 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.*" [12] 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. We already started the work on extracting such conditions using the infrastructure that we built for our current approach and had some preliminary results. We plan to continue this work and evaluate the effectiveness of this technique.

Ordering in ACP rules. Our current approach extracts ACP rules from sentences without considering the ordering of the rules, which may cause security holes in the extracted ACP rules. We plan to study the extracted ACP rules and propose 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 actor flow tracking and perspective conversion techniques proposed in our approach partially address the context-aware analysis in action-step extraction. For example, customer may not pay the order if he has not selected an order. We plan to propose more techniques to even deal with state transitions during action-step extraction.

9. RELATED WORK

Manual Extraction of ACPs From Requirement Documents. He and Anton [16] 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 the ambiguities of requirements documents and identifies inconsistencies among requirement documents and database design. Their objective is to derive a comprehensive set of ACP rules, similar to our approach. However, our approach includes novel NLP techniques to automatically extract ACPs from NL documents, while their approach is a manual process.

Template Matching. Etzioni et al. [13] 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 the two approaches 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 (such as subject, main verb group, and object). Our semantic patterns are more general and provide high precision on identifying ACP sentences as shown in our evaluations.

NLP to Assist Privacy Policy Authoring. The SPARCLE Policy Workbench [10, 11, 20, 21] employs the shallow parsing technique [24] to parse the 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 [8] and XACML [4, 25] with a privacy policy profile. Although SPARCLE workbench is effective in helping authoring privacy policies in natural language, their technique provides simple templates to extract elements for constructing privacy rules while our approach provides more general semantic patterns. Additionally, their approach does not have any technique to infer negative meaning of sentences.

Use Case Analysis. Our previous approach [27, 28] provides NLP techniques to parse and represent the contents of use cases in use case models. Our previous approach then provides a suite of model analysis techniques that leverage such models to validate whether the style and content of uses cases conform to guidelines of use case style and content. The extraction of contents of use cases into formal models is similar to the action-step extraction in our approach. However, our approach provides additional techniques to address the challenges of *TC4-Transitive Actor* and *TC5-Perspective Variance* and produces extracted action steps as access control requests for validation against specified and extracted ACPs.

10. CONCLUSION

In this paper, we propose a novel approach, called Text2Policy, that incorporates syntactic and semantic NL analysis around models such as the ACP model and action-step model to extract model instances from NL documents and produces formal specifications. We provide new techniques to concretize our general approach to extract ACP rules from NL documents and extract action steps from functional requirements. From the extracted ACPs, our approach automatically generates machine-enforceable ACPs in specification languages such as XACML, which can be automatically checked for correctness. From the extracted action steps, our approach automatically derives access control requests with expected permit decisions. These access control requests can be used for automatic validation against specified or extracted ACPs for detecting inconsistencies, assisting correct enforcement of ACPs. We have evaluated Text2Policy on 37 iTrust use cases with 448 use-case sentences and the collected 115 ACP sentences from 18 sources and the results show that Text2Policy effectively identifies ACP sentences from the iTrust use cases, extracts ACP rules from ACP sentences, and extracts action steps from action-step sentences in iTrust use cases.

11. 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) Version 2.0, 2005. docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.

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

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

[6] Text2Policy, 2011. http://research.csc.ncsu.edu/ase/projects/text2policy/.

[7] R. K. Ando. Exploiting Unannotated Corpora for Tagging And Chunking. In *Proc. ACL*, 2004.

[8] 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/`.

[9] B. K. Boguraev. Towards Finite-State Analysis of Lexical Cohesion. In *Proc. International Conference on Finite-State Methods for NLP.*, 2000.

[10] 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*, 2005.

[11] 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*, 2006.

[12] D. J. Dougherty, K. Fisler, and S. Krishnamurthi. Specifying And Reasoning about Dynamic Access-Control Policies. In *Proc. IJCAR*, pages 632–646, 2006.

[13] 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.

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

[15] G. Grefenstette. *Light Parsing as Finite State Filtering.* Cambridge University Press, 1999.

[16] Q. He and A. I. Antón. Requirements-based Access Control Analysis and Policy Specification (ReCAPS). *Inf. Softw. Technol.*, 2009.

[17] V. Hu, R. Kuhn, T. Xie, and J. Hwang. Model Checking for Verification of Mandatory Access Control Models and Properties. *International Journal of Software Engineering and Knowledge Engineering*, 2010.

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

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

[20] C.-M. Karat, J. Karat, C. Brodie, and J. Feng. Evaluating Interfaces for Privacy Policy Rule Authoring. In *Proc. CHI '06*, 2006.

[21] J. Karat, C.-M. Karat, C. Brodie, and J. Feng. Designing Natural Language and Structured Entry Methods for Privacy Policy Authoring. In *INTERACT*, 2005.

[22] C. Kennedy. Anaphora For Everyone: Pronominal Anaphora Resolution Without A Parser. In *Proc. COLING*, 1996.

[23] E. Martin, J. Hwang, T. Xie, and V. Hu. Assessing Quality of Policy Properties in Verification of Access Control Policies. In *Proc. ACSAC 2008*, pages 163–172, 2008.

[24] M. S. Neff, R. J. Byrd, and B. K. Boguraev. The Talent System: TEXTRACT Architecture And Data Model. *Nat. Lang. Eng.*, 10(3-4), 2004.

[25] 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.

[26] P. Samarati and S. D. C. d. Vimercati. Access Control: Policies, Models, and Mechanisms. In *IFIP WG 1.7 Int'l School on Foundations of Security Analysis and Design (FOSAD 2000)*, pages 137–196, 2001.

[27] A. Sinha, S. M. S. Jr., and A. Paradkar. Text2Test: Automated Inspection of Natural Language Use Cases. In *Proc. ICST*, 2010.

[28] 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*, 2009.

[29] M. Stickel and M. Tyson. FASTUS: A Cascaded Finite-state Transducer for Extracting Information from Natural-language Text. In *Finite-State Language Processing.* MIT Press, 1997.

[30] 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. ASEE/IEEE Frontiers in Education Conference*, pages 30–31, 2006.

[31] N. Zhang, M. Ryan, and D. P. Guelev. Synthesising Verified Access Control Systems in XACML. In *Proc. FMSE*, pages 56–65, 2004.