Towards an Intelligent Domain-Specific Traceability Solution

Jin Guo, Natawut Monaikul, Cody Plepel, and Jane Cleland-Huang
Systems and Requirements Engineering Center
School of Computing, DePaul University
Chicago, IL USA
jguo9@cdm.depaul.edu; natawutmo@gmail.com; plepel.cody@gmail.com; jhuang@cs.depaul.edu

ABSTRACT

State-of-the-art software trace retrieval techniques are unable to perform the complex reasoning that a human analyst follows in order to create accurate trace links between artifacts such as regulatory codes and requirements. As a result, current algorithms often generate imprecise links. To address this problem, we present the Domain-Contextualized Intelligent Traceability Solution (DoCIT), designed to mimic some of the higher level reasoning that a human trace analyst performs. We focus our efforts on the complex domain of communication and control in a transportation system. DoCIT includes rules for extracting "action units" from software artifacts, a domain-specific knowledge base for relating semantically similar concepts across action units, and a set of link-creation heuristics which utilize the action units to establish meaningful trace links across pairs of artifacts. Our approach significantly improves the quality of the generated trace links. We illustrate and evaluate DoCIT with examples and experiments from the control and communication sector of a transportation domain.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/ Specifications; I.2.7 [**Computing Methodologies**]: Natural Language Processing —*Text Analysis*

Keywords

Traceability; Intelligent System; Domain Ontology

1. INTRODUCTION

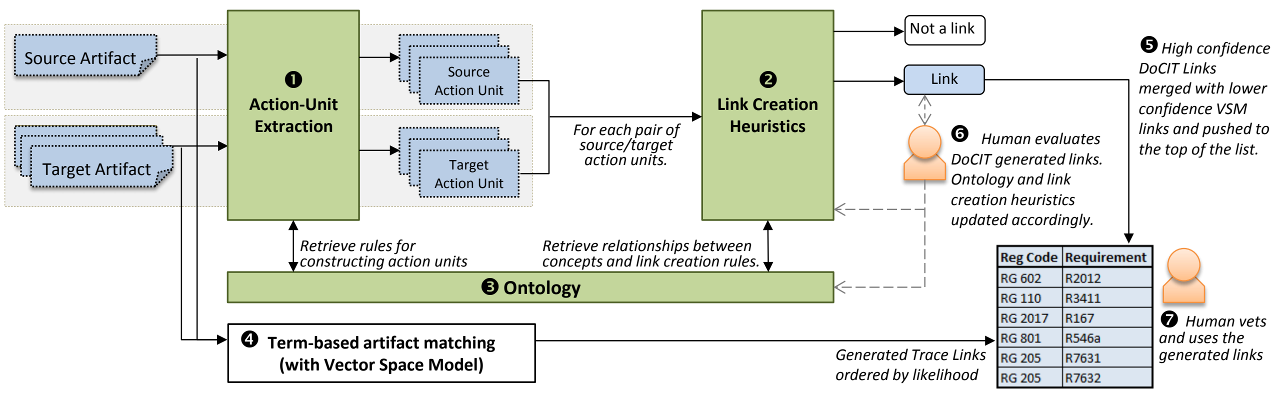
Software traceability is an essential component in the development of any non-trivial software system. It is defined by the Center of Excellence for Software Traceability (CoEST) as "the ability to interrelate any uniquely identifiable software engineering artifact to any other, maintain required links over time, and use the resulting network to answer questions of both the software product and its development process" [1]. In safety-critical systems regulated by bodies such as the US Food and Drug Administration or the Federal Aviation Authority [28], it is specifically required as part of the assessment, approval, and certification process [19]. In all domains, software traceability can provide useful support for a variety of software engineering activities including compliance verification, impact analysis, and test-case regression analysis. Despite its importance, numerous studies have shown that achieving effective traceability is challenging, primarily because creating and maintaining accurate trace links for large and/or complex projects is effort-intensive, arduous, and error-prone [7, 22, 27, 23].

To address these problems, researchers have developed and evaluated techniques that use standard information retrieval methods to automate, or semi-automate, the process of trace link creation. Techniques include the Vector Space Model (VSM), Latent Semantic Indexing (LSI), and probabilistic networks [3, 14]. Other techniques involve instrumenting the project environment and configuration management tools to prospectively capture trace links as a byproduct of the development process [5]. Unfortunately, while such techniques are often able to return 85-90% of the targeted links, precision rates are often quite low, ranging from 10-50% depending on the characteristics of the project and its software artifacts [14]. Industry demands more precise results before they are willing to integrate trace retrieval techniques into their software development tools and processes [6], and therefore these low accuracy problems have hindered the adoption of trace retrieval practices in industry.

1.1 Term Mismatches

The major barrier is caused by the mismatch of terms that occurs frequently across software artifacts to be traced [10]. For example, consider the following system requirement for a Driver-Optional Highway System (DOHS) that states: *the Highway Wayside Segment shall monitor signal, road work directive, and hazard detector information from field devices* and the related system design artifact that states: *during lamp-out conditions the WIU shall send the current state of the highway signal.* Even though the two artifacts exhibit almost no shared terms, the fact that the WIU is an acronym of *Highway Wayside Interface Unit*, that WIUs are located in the Highway Wayside Segment and are connected to field devices, and that monitoring of field devices is supported by data, mean that these two artifacts are related and a

ASE'14, September 15-19, 2014, Vasteras, Sweden.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3013-8/14/09 ...$15.00.
http://dx.doi.org/10.1145/2642937.2642970 .


Source Artifact
Target Artifact
❶ Action-Unit Extraction
Source Action Unit
Target Action Unit
For each pair of source/target action units.
❷ Link Creation Heuristics
Not a link
Link
❻ Human evaluates DoCIT generated links. Ontology and link creation heuristics updated accordingly.
❺ High confidence DoCIT Links merged with lower confidence VSM links and pushed to the top of the list.
Retrieve rules for constructing action units
Retrieve relationships between concepts and link creation rules.
❸ Ontology
❹ Term-based artifact matching (with Vector Space Model)
Generated Trace Links ordered by likelihood
❼ Human vets and uses the generated links

Reg Code	Requirement
RG 602	R2012
RG 110	R3411
RG 2017	R167
RG 801	R546a
RG 205	R7631
RG 205	R7632

Figure 1: The DoCIT process showing its primary components and execution context.

trace link should be created. Current term-based trace retrieval algorithms are incapable of understanding these concepts and would therefore be unlikely to precisely create the corresponding link [10]. On the other hand, most human domain experts performing the tracing task manually would be able to look beyond the mismatch of terms, reason about the concepts in the domain, and correctly establish a trace link between a source and target artifact.

Approaches that attempt to solve this problem through the use of ontologies alone fail to take into consideration specific roles played by different concepts [20, 18]. They therefore do not handle the nuances of why two artifacts with similar terms, or even terms related through the ontology, should or should not be linked. Approaches that rely purely on ontology therefore may improve recall but still fail to adequately address the precision problem [20].

1.2 DoCIT Solution

Our work delivers a Domain-Contextualized Intelligent Traceability (DoCIT) system, capable of performing human-like reasoning for the highly focused area of communication and control within a transportation domain. While there are non-trivial costs involved in creating a DoCIT system for even a single regulatory domain, this initial outlay must be balanced against the far greater costs of establishing a sufficient set of trace links to demonstrate compliance and safety for such systems. Our experiences of working in this domain have shown that traceability costs for a single project can easily exceed \$1 Million (US) [6]. Fortunately, as regulations apply across multiple projects, the DoCIT knowledge base can be reused in projects that need to comply to the same regulatory codes.

1.3 Overview

The remainder of the paper is laid out as follows. Sections 2 and 3 provide an overview of DoCIT and the targeted domain. Sections 4, 5, and 6 describe action units, link heuristics, and the knowledge base respectively. Section 7 describes the process we used for constructing the initial knowledge base and the way we implemented DoCIT in conjunction with a term-based approach. Section 8 describes a series of experiments we conducted to evaluate DoCIT. Finally, Section 9 discusses threats to validity, Section 10 presents related work, and Section 11 summarizes our findings and lays out directions for future work.

2. DOCIT

The various elements of DoCIT are depicted in Figure 1. DoCIT takes as input a *source* artifact (e.g. a regulatory code) and a set of *target* artifacts (e.g. system level requirements) to be traced. Trace links are built around *action units* extracted from source and target artifacts ①. Link heuristics define the conditions under which the properties of two action units determine that a link should be created between their associated source and target artifacts ②. Both the action unit extraction process and the application of link heuristics are supported by a domain-specific ontology ③.

While DoCIT tends to deliver more precise trace links than term-based algorithms, it depends on a relatively complete and correct set of action unit definitions, action unit extraction rules, link heuristics, and supporting ontology. We refer to these collectively as the DoCIT *knowledge base* from now on. Given that we cannot guarantee completeness of this knowledge base, especially when tracing new datasets, we deploy DoCIT within the 'safety-net' of the popular, term-based Vector Space Model (VSM) ④. By merging DoCIT generated links with term-based generated ones, we push the more precise DoCIT links towards the top of the merged list of candidate links. Furthermore, we leverage the VSM similarity scores to create an ordered ranking of DoCIT links ⑤.

Finally, we incorporate human feedback into our process. This facilitates ongoing refinements in the DoCIT knowledge base ⑥, and assigns the human analyst the final task of vetting the generated trace links ⑦.

The work we present in this paper builds on our previous proof-of-concept [13]. However, our earlier work relied upon a human analyst to manually map artifacts onto a simple set of seven trace creation heuristics, and utilized a rudimentary ontology. In contrast, the work in this paper makes very significant advances in moving toward our longer-range goal of complete automation. Knowledge is now represented in a formal ontology language. The novel concept of action units and their associated extraction rules are introduced, allowing us to automatically parse and analyze non-trivial industrial requirements. Furthermore, the previous high-level link heuristics are discarded and replaced with 33 heuristics centered around the new notion of action units. Given an underlying knowledge base, these link heuristics allow us to automate the task of generating a relatively precise set of trace links.

3. TRANSPORTATION DOMAIN

The purpose of this paper is to demonstrate that a DoCIT solution can be used effectively in a complex regulated domain. We focus our efforts on the communication and control aspects of the transportation domain for several reasons. First, it represents a regulated safety-critical industry for which traceability is required. Second, its projects tend to grow very large with tens of thousands of trace links and therefore the tracing task can be particularly costly and arduous. Finally, we have access to two large industrial datasets from our collaborators. Due to the proprietary nature of the project, we are not able to disclose its exact domain, and therefore all of our examples in this paper have been disguised under the domain of communication and control for the Driver-Optional Highway System (DOHS). We established a one-to-one mapping between terms in our domain and terms in the DOHS. The DOHS provides environmental controls used by both manned and unmanned vehicles to propose routing, to prevent vehicles from entering accident zones, and to provide a wide range of directives.

The dataset includes 242 System Requirement Specification artifacts (SRS), 963 System Design Description artifacts (SDD), and 583 SubSystem Requirement Specification artifacts (SSRS). Our industrial collaborators provided two validated trace matrices which included 583 trace links from SRS to SDD, and 700 trace links from SSRS to SDD.

4. ACTION UNITS

The *action unit* represents a novel and fundamental building block of our approach. In this section we explain what an action unit is, how it is defined, and finally how it is instantiated through a natural language extraction process.

4.1 Action Unit Definition

Each action unit is defined in terms of a *verb syntactic* group, a set of *properties*, and a set of *mapping rules* which identify a part of speech to be mapped to each property.

A verb syntactic group represents a group of verbs which belong together because they tend to be *used* in similar ways. For example, the verbs *transmit*, *broadcast*, *indicate*, *upload*, and *deliver* do not mean exactly the same thing, but they do exhibit the same syntactical usage pattern. As a result they also exhibit similar part-of-speech dependencies. For example, we can say that when one of these verbs is found in a sentence, the recipient of the action is normally identified as the noun phrase which is the object of the preposition *to* associated with the verb.

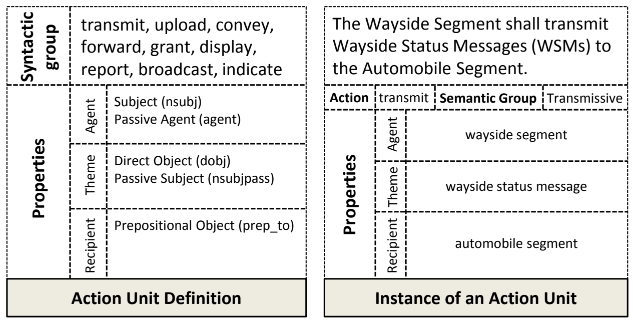

Syntactic group
transmit, upload, convey, forward, grant, display, report, broadcast, indicate
Properties
Agent
Subject (nsubj)
Passive Agent (agent)
Theme
Direct Object (dobj)
Passive Subject (nsubjpass)
Recipient
Prepositional Object (prep_to)
Action Unit Definition
The Wayside Segment shall transmit Wayside Status Messages (WSMs) to the Automobile Segment.
Action
transmit
Semantic Group
Transmissive
Properties
Agent
wayside segment
Theme
wayside status message
Recipient
automobile segment
Instance of an Action Unit

Figure 2: A definition of an action unit showing sample properties and an example instantiation

The action unit's properties are then defined in terms of thematic roles [26], which represent a standard set of roles that a noun phrase can play with respect to a verb. While linguistic theories have recognized over 1000 different types of thematic roles, we observed six of them in the data set from which the knowledge base was constructed. These are:
• **Agent:** The initiator of some action, capable of acting with volition, as in "The **DOHS system** shall send..."
• **Theme**: The entity which is moved by an action, as in "...shall send a **message**"
• **Recipient**: The object which receives or gains something as a result of the action, as in "...send a message to a **subsystem**"
• **Instrument**: The object with which or by which the action is performed, as in "...send a message through an **interface**"
• **Location**: The place in which the action is completed, as in "...send a message within the **limits**"
• **Initial Location**: The place in which the action is initiated, as in "...leave the **area**". This role was created to account for sentences containing two locations, as in "transition from A to B".
The work in this paper is based on these six observed roles, however DoCIT algorithms are extensible to accommodate additional roles as needed.

Finally, the action unit definition provides a set of mapping rules which describe the part-of-speech dependencies that must be mapped to each property. Different definitions use various combinations of properties and dependencies.

For example, in Figure 2, an action unit definition is given for a syntactic group related to the set of actions *transmit*, *upload*, *convey*, etc. The definition prescribes the need to identify an *agent*, *theme*, and *recipient* and specifies that these roles are played by the subject (nsubj), direct object (dobj), and prepositional object of *to* (prep_to) respectively.

The same figure shows an instantiated action unit populated through parsing the artifact "The Wayside Segment shall transmit Wayside Status Messages (WSMs) to the Automobile Segment." The action *transmit* was recognized as a member of the *Transmissive* semantic group, and the noun phrases *wayside segment*, *wayside status message*, and *automobile segment* were mapped to the agent, theme, and recipient properties respectively. We discuss the purpose of the semantic group in the next section.

4.2 Instantiated Action Units

An action unit needs to be identified and extracted automatically from the text in an artifact. The challenging aspect of this task is that artifacts, such as requirements and regulatory codes, are written in natural language, often describe multiple features, carry extraneous information such as rationales, definitions, and other forms of description, often describe complex constraints and conditions, and are not always grammatically well-formed.

Action unit extraction is a natural language processing (NLP) task. We used the Stanford parser for this task because its unlexicalized PCFG Parser [17] can achieve relatively accurate parsing results with fast processing speed. We now provide an overview of the parsing process.

4.2.1 Preprocessing

The text from the artifact is first preprocessed by replacing terms such as (/) with *or*, and removing superflu-

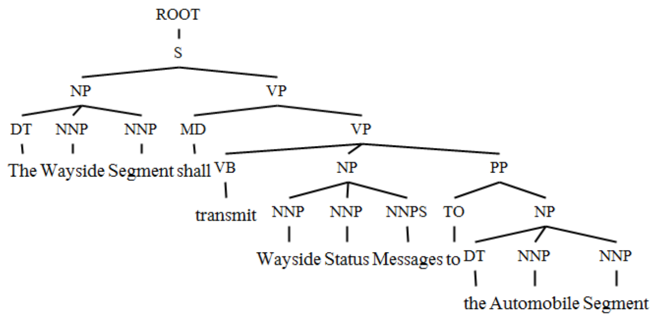


Figure 3: Parsed Tree of Sample Artifact

Table 1: Dependencies extracted from the parse tree

det(Segment-3, The-1)	nn(Messages-8, Status-7)
nn(Segment-3, Wayside-2)	*dobj(transmit-5, Messages-8)*
nsubj(transmit-5, Segment-3)	det(Segment-12, the-10)
aux(transmit-5, shall-4)	nn(Segment-12, Automobile-11)
root(ROOT-0, transmit-5)	*prep_to(transmit-5, Segment-12)*
nn(Messages-8, Wayside-6)	

ous characters such as (-) which confuse the parsing process. In addition, unhelpful, frequently occurring parts of the sentence, such as *provide [a mechanism] for* were removed through a string pattern matching process. Such patterns have been added to our knowledge base.

4.2.2 Parse Tree

The Stanford parser was used to produce a POS (Part-of-Speech) tree as illustrated in Figure 3. Each node is labeled with a POS tag. We limited our approach to support only noun phrases (NP) as they have been shown to play the most crucial role in the tracing process [34]. NPs such as *all critical data* can be identified by extracting nodes labeled *NP* in the parsed tree. Phrases which have similar meanings but different structures, such as *signal state* and *state of signal*, are transformed into a standard form using tree pattern analysis and matching techniques.

Overall, we processed basic noun phrases, lists of noun phrases, and also prepositional phrases, verb phrases, and clauses found within a noun phrase. From this tree, the Stanford parser generated a dependency set, as depicted in Table 1, which provides the information needed to automatically map POS to the properties defined in each applicable action unit definition. Thus, a dependency such as *nsubj(transmit-5, Segment-3)* relates *(Automobile) Segment* as the subject of *transmit*. In general, the NP in the *nsubj* dependency with the verb *transmit*, along with any other verb in its syntactic group, should be considered the agent of the verb. From the dependency set, we may then extract an instance of the action unit from an artifact in an automated fashion as each relevant dependency links a verb to nouns that get mapped onto action unit properties.

4.2.3 Semantic Verb Groups

There is one final step in the action unit instantiation process. While properties and their mappings are driven by the *usage* patterns of the verb, the link creation process is driven by their *semantics*. The final step in the process therefore involves looking up the semantic group of the verb. Our current knowledge base includes 24 distinct semantic groups such as *Affirmative*, *Calculative*, *Evasive*, and *Receptive*. While standard semantic groupings do exist (i.e. in WordNet), they tend to focus on quite commonly used verbs and do not include many of the verbs found in our transportation domain. Furthermore, in many cases, no appropriate grouping exists in which a new verb can be added. We therefore developed our own semantic groupings following the process described in Section 7.1 of this paper.

The end result of the extraction process is that each artifact is represented by one or more instantiated action units. It is these action units which are then used in the trace link generation process.

5. LINK HEURISTICS

Trace links are created by comparing two action units extracted from a source and target artifact respectively. Currently, a trace link is established between the source and target artifact whenever a relationship is established between a pair of their action units. However, in future implementations we plan to take all pairs of action units for a source and target artifact into account simultaneously in order to reason more effectively about conditions and constraints.

5.1 Link Heuristic Definition

Each link heuristic provides a set of guidelines for evaluating whether two action units of specified semantic groups should be linked or not. For this reason, most heuristics are named according to a semantic pair, for example: motive-permissive, transmissive-administrative or transmissive- receptive. The transmissive-receptive heuristic can only be applied when one action unit is of type transmissive, and the other is of type receptive, and so on. Currently, the only exception is the *basic* heuristic, which can be applied whenever two action units share the same semantic group.

Link heuristics also specify rules for comparing the action unit properties. For example, a heuristic might specify a rule that the *themes* of two action units must be the same, or that the *agent* of one action unit must match the *recipient* of another. Such matches can be either *exact* or *hierarchical*. An exact match means that two elements are the same or equivalent, while a hierarchical match means that two elements are related through the is-a or compositional hierarchy of the ontology. For example, a *Wayside Interface Unit* is part of the *Wayside Segment*. Such hierarchical and compositional relationships are captured in the domain ontology which we describe in Section 6.

5.2 Sample Link Heuristics

We illustrate our approach with three different link heuristics. Each is defined in terms of its semantic groups, as well as the rules by which the elements must match, where an element is defined as the terms assigned to a property in the action unit. For example, *WIU* might be the element assigned to a theme property.

There are currently four types of matching criteria (MC) defined as follows for a pair of elements:
•**MC 0** : If both elements are present, then they must match (exact or hierarchical). However, if one or both of the elements are missing, the MC can be ignored.
•**MC 1** : If one element is present, the other must be present too and must match (exact or hierarchical). If neither element is present, the MC can be ignored.

•**MC 2** : Both elements must be present and must match (exact or hierarchical).
•**MC 3** : Both elements must be present and must match exactly.

5.2.1 Basic Link

The basic heuristic is applicable when the semantic group in both action units are the same. A link is established if and only if *agents*, *recipients*, and *instruments* are matched at MC level 0, and *themes* and *locations* are matched at MC level 1. We provide the following illustrative example:

A1: Any critical failure during the Disengaged Mode will force the OBM to enter the Failed Mode.

Action	enter
Semantic Group	Motive
Themes	obm
Location	fail mode

A2: The VehicleGuard DOH System shall have a Failed Mode where the Automobile segment transitions if a vital hardware or software failure is detected.

Action	transition
Semantic Group	Motive
Themes	automobile segment
Location	fail mode

In this case all the criteria for the heuristic are met. Both action units belong to the *Motive* semantic group. The OBM (i.e. Onboard Module) is part of the automobile segment; therefore, the themes match at the hierarchical level. Finally, the two locations exhibit an exact match. Therefore a trace link is established.

5.2.2 Transmissive-Receptive

This heuristic applies when the semantic group in one action unit is *Transmissive* and the other is *Receptive.* A link is established if and only if *agents*, *recipients*, *instruments*, and *locations* match at MC level 0, and *themes* match at MC level 2. We provide the following example.

A1: The OBM shall support reception and decomposition of Wayside Status Messages (WSM).

Action	reception
Semantic Group	Receptive
Recipients	obm
Themes	wayside status message

A2: The Wayside Segment shall transmit information to the Automobile Segment in the form of Wayside Status Messages (WSMs).

Action	transmit
SemanticGroups	Transmissive
Agents	wayside segment
Recipients	automobile segment
Themes	wayside status message

In this case the semantic groups are *Transmissive* and *Receptive* and so the heuristic is applicable. Checking the property pairs, we see that *agents, instruments*, and *locations* are missing and therefore ignored. Furthermore, the *recipients* property is present in both cases, and *OBM* stands for *Onboard Module*, which is a part of the *Automobile Segment* and therefore hierarchically linked through the ontology. Finally, both action units contain the same theme of *Wayside Status Message*. As a result, a link is created.

5.2.3 Motive-Permissive:

This heuristic applies when the semantic group in one action unit is *Motive* and the other is *Permissive.* A link is established if and only if *locations* match at MC level 3, and *themes* match at MC level 0. We illustrate this with a final example.

A1: The Cut-Out mode is entered automatically from the self-test mode.

Action	enter
SemanticGroups	Motive
Location	Cut-Out Mode
InitialLocation	Self Test Mode

A2: The VehicleGuard DOH System shall be forced into cut-out mode by selecting the cut-out mode setting on the sealed mode switch on the OBM unit.

Action	force
SemanticGroups	Permissive
Themes	VehicleGuard DOH System
Location	Cut-Out Mode

In this example, the semantic groups are *Motive* and *Permissive*; *location* properties are an exact match, and as the *theme* property is present on only one side, it is ignored. A link is therefore created.

Our current knowledge base includes 33 link heuristics, each of which is summarized in Figure 4.

6. ONTOLOGY

Both the action units and link heuristics are supported by an underlying ontology. Without the ontology we would be unable to understand conceptual relationships between different terms and would therefore be constrained by the same problems as more basic term-matching approaches.

A knowledge base is an information repository used to collect, organize, and search data [16, 29]. Many KBs are built in a manner that supports automated deductive reasoning. Such KBs are composed of a set of data that includes basic terms that define the vocabulary of the domain, and a set of sentences that describe the relationships between those terms. The data is often represented in the form of an ontology in which knowledge is constructed using logical operators such as *AND* and *OR*, as well as *implication* and *negation* operators to build complex ideas from more primitive concepts [16].

Our KB performs two primary functions for tracing purposes: using the knowledge in the KB to *compare* two artifacts and *mapping* elements of software artifacts to concepts in the ontology. We also store syntactic group information and semantic group information in the knowledge base so that given a verb, we can determine which groups it belongs to.

The DoCIT KB was created using OWL 2 Web Ontology Language, a formally defined ontology language developed by W3C for the Semantic Web [2]. OWL 2 is capable of representing complex knowledge about things, groups of things, and relationships between those things. OWL 2 ontologies provide classes, properties, individuals, and data values, all

Semantic Groups	Agent	Theme	Recipient	Instrument	Location	Init Loc.	Agent$_1$/Recip.$_2$	Recip.$_1$/Agent$_2$	Instrum.$_1$/Theme$_2$	Init Loc.$_1$/Loc.$_2$	Loc.$_1$/Theme$_2$	Agent$_1$/Theme$_2$
Basic (Groups must match)	⓿	❶	⓿	⓿	❶	⓿						
Affirmative-Inclusive	⓿	❷										
Calculative-Inclusive	⓿	❷										
Descriptive-Transmissive		❷										
Enforcive-Calculative (1)	❷								❷			
Enforcive-Calculative (2)	⓿	❷	⓿	⓿	⓿							
Enforcive-Enforcive	❷								❷			
Enforcive-Receptive		❷					⓿					
Enforcive-Regulative (1)	❶										❷	
Enforcive-Regulative (2)	⓿	❷										
Inceptive-Inceptive	⓿	❷		⓿	⓿							
Inceptive-Receptive		❷					⓿					
Inceptive-Transgressive (1)		❷			❸							
Inceptive-Transgressive (2)					❸							❷
Inspective-Respective							⓿		❷			
Motive-Motive					❸							
Necessitative-Receptive							⓿		❷			
Permissive-Motive		⓿			❸							
Permissive-Permissive					❸							
Preservative-Inceptive	❷	❷										
Preservative-Preservative	⓿	❷	⓿	⓿	⓿	⓿						
Receptive-Receptive		❷						❸				
Receptive-Transmissive (1)		❷					❸					
Receptive-Transmissive (2)	⓿	❷	⓿	⓿	⓿							
Regulative-Inclusive	⓿	❸	⓿	⓿	⓿							
Regulative-Transmissive		❷					❶					
Transgressive-Motive		⓿			❷							
Transgressive-Permissive					❸							
Transmissive-Administrative		⓿					❷					
Transmissive-Enforcive		⓿					❷					
Transmissive-Inceptive	❷	❷										
Transmissive-Permissive		❷						❸				
Transmissive-Transmissive	⓿	❷										
⓿ = If both elements are present, they must match (exact or hierarchical).												
❶ = If one element is present, the other must also be, & match (exact or hier.)												
❷ = Both elements must be present and must match (exact or hierarchical)												
❸ = Both elements must be present and must exhibit an exact match.												

Figure 4: Trace Link Heuristics specified in terms of semantic pairs and required matches.

of which are stored as Semantic Web documents. We created our initial KB using Protégé, an open source ontology editor [31], and saved it in an OWL (.owl) format for use during the tracing process.

6.1 Knowledge Structure

The KB contains general concepts, which are expected to be reusable across multiple domains, and also domain-specific knowledge which is designed solely for use in a specific regulatory domain, e.g., communications and control for the transportation sector of our case study.

The general section of our KB includes facts about *verbs* and *verbs used as nouns*, such as *reception* or *integration*. These are represented as individual elements in the ontology. Every identified verb belongs to one or more semantic groups representing its functionality. For example, the verb *enter* is used in two functionally distinct ways across the set of observed artifacts: first, as in "The system should enter the Failure Mode when a vital error occurs" (the **Motive** group), and second, as in "The driver should be able to enter the current date into the system from the HMI" (the **Transmissive** group).

Each verb is also categorized under one or more syntactic groups based on how it interacts with the surrounding noun phrases. Reexamining the previous example, we see that the *Failure Mode* is a location that the *system* should *enter*, while the *current date* is something the *driver* should *enter*, which gets received by the *system*. Despite having similar syntactic distributions, the arguments of *enter* in each case play different roles in determining the main function contained in each sentence.

The second part of the ontology describes domain-specific taxonomic relations between objects, such as names of systems, subsystems, operating modes, and message types used by components of the system.

6.2 Querying the KB

For an ontology to be useful, it needs to provide a mechanism for supporting queries and returning relevant information. The DoCIT ontology therefore provides a query function which can accept either a single term or a noun phrase and return a set of related concepts as well as a set of relations. For example, if we issue a simple query for the term *update*, the KB returns the information that *update* is an action and belongs to the *regulative* semantic group and syntactic group # 3 (as syntactic groups are labeled by ID only).

Similarly, a query issued for the noun phrase *Power-On Self Test* returns a rich set of hierarchically related terms, among other things, specifying that the Power-On Self Test is a kind of relay test, which in turn is a kind of self-diagnostic test. This information is critical for enabling traceability between artifacts which may use different terms that are related through a hierarchy of concepts.

7. DoCIT IN PRACTICE

Deploying DoCIT required construction of the knowledge base and development of a functioning prototype.

7.1 Building a Knowledge Base

Our current version of DoCIT requires the presence of a knowledge base composed of action units, link heuristics, and a domain ontology. In this section, we describe the systematic approach we followed to build the knowledge base needed to support automated trace link generation in our domain of study. We worked closely with a domain expert from our collaborating industry to ensure that our reasoning, and the subsequent results, were well-founded. Articulating these steps is an important precursor towards complete automation of ontology building for tracing in a given software engineering domain.

• **Step 1. Create Syntactic and Semantic Groups:** The Stanford parser was used to identify verbs and verbs represented as nouns. For each identified verb we manually checked whether an appropriate syntactic group and semantic group existed. To identify relevant syntactic groups we asked, "Is there an existing syntactic group which contains verbs conveying similar behavior?" To identify relevant semantic groups we asked, "Is there an existing semantic group which contains verbs conveying similar meaning?" If not, we created a new syntactic and/or semantic action group.

• **Step 2. Create Action Unit Construction Rules:** For each new syntactic group, we created a set of action unit construction rules. These rules specify the relevant properties (i.e. thematic roles such as location, agent, etc.) and the appropriate dependent POS which should, if present, be associated with the property.

Let us assume that no existing syntactic group exists for the verb *send*. First, we run the Stanford parser to identify verbs and their dependencies. We then inject a human reasoning step. For example, we might determine that the action "send" requires a direct object, i.e. "send what?" Furthermore, this direct object is generally an entity that is moved by the action. We therefore add "theme" as a property to the action unit and specify that the *direct object* serves this role. A similar process is used to add other relevant properties and their associated parts-of-speech.

• **Step 3. Create Link Heuristics:** Link heuristics are built around pairs of semantic groups. The occurrence of a previously unseen pair of semantic groups will trigger the need for a new link heuristic to be created through analyzing the likely thought process of a human analyst. First we determine the rationale for the link, and secondly we formulate this in terms of the thematic roles. We frequently engaged our industrial collaborator in this process so that he could confirm or refute our rationales for each link.

It is worth noting that our current implementation takes a rather greedy approach to constructing the various rules, heuristics, and concepts. Each of these elements is added to the knowledge base if it benefits the currently evaluated artifact and/or trace link. While, we considered implementing a "do no harm" policy, i.e. not adding new information if it harmed previous trace links, we rejected this idea in favor of a more global optimization to be explored in future work.

7.2 Integration with Term-based Approach

As previously explained, we implemented DoCIT in conjunction with the Vector Space Model (VSM) [14].

7.2.1 Vector Space Model

The VSM algorithm computes similarity between two documents. Each query and each document is represented as a term vector defined in the space of all terms found in the set of queries T = $t_1, t_2, ..., t_n$. More formally, a document d is represented as a vector $\vec{d} = (w_{1,d}, w_{2,d}, ..., w_{n,d})$, where $w_{i,d}$ represents the term weight associated with term i for document d. A query is similarly represented as $\vec{q} = (w_{1,q}, w_{2,q}, ..., w_{n,q})$. Each term t is assigned a weight using a standard weighting scheme known as $tf-idf$ [15], where tf represents the term frequency, and idf the inverse document frequency.

Term frequency with respect to document d is often computed as $tf(t_i, d) = (freq(t_i, d))/(|d|)$, where $freq(t_i, d)$ is the frequency of the term in the document, and $|d|$ is the length of the document. Inverse document frequency idf, is usually computed as $idf_{ti} = log_2(n/n_i)$ where n is the total number of documents in the traceable collection, and n_i is the number of documents in which term t_i occurs. The individual term weight for term i in document d is then computed as $w_{id} = tf(t_i, d) \times idf_{ti}$.

A similarity score $sim(d, q)$ is computed between document d and query q as follows:

$$sim(d,q) = \frac{\left(\sum_{i=1}^{n} w_{i,d} w_{i,q}\right)}{\left(\sqrt{\sum_{i=1}^{n} w_{i,d}} \cdot \sqrt{\sum_{i=1}^{n} w_{i,q}}\right)} \qquad (1)$$

Any query-document pair receiving a similarity score over a threshold value is treated as a candidate traceability link.

7.2.2 Merging Results

To use VSM in conjunction with DoCIT, the DoCIT approach is first applied to the source and target artifacts to produce a list of candidate links referred to from now on as candidate DoCIT-links. VSM is then applied to the same source and target artifacts to produce a ranked list of candidate VSM-links.DoCIT tends to produce fewer, but more accurate links than VSM. Therefore, a merged list of candidate links is created by placing DoCIT links at the top of the list ordered according to their VSM rankings, and then adding all remaining VSM-Links in their original order to the bottom of the merged list.

7.2.3 Learning New Facts and Rules

The process of discovering and specifying link heuristics is ongoing, and therefore any knowledge base needs the ability to grow and expand over time as new information, acronyms, or sentence structures are seen for the first time. We therefore developed a GUI tool which incrementally displays links which were missed by DoCIT but ranked highly by VSM. These links provide an ideal opportunity for eliciting knowledge from the domain expert who is performing the tracing task. The source and target artifacts of the link are parsed and action units are displayed. The tool then provides a GUI to help the user to construct new link heuristics.

8. EVALUATION

To evaluate DoCIT's ability to support trace link generation we designed and executed four different experiments labeled E1 to E4. The first evaluated DoCIT's ability to correctly identify and retrieve action units from both the SRS and SDD datasets. The second and third experiments evaluated DoCIT's ability to generate trace links correctly with varying degrees of support from the knowledge base, and finally the fourth one analyzed the efficacy of the link heuristics. An alpha level of 0.05 was used for all statistical tests.

8.1 E1: Action Unit Extraction

Our DoCIT solution is intended to work in complex software systems with real requirements. The first experiment was therefore designed to address the research question **(RQ 1:)** "Can DoCIT extract action units correctly and consistently from complex industrial requirements?"

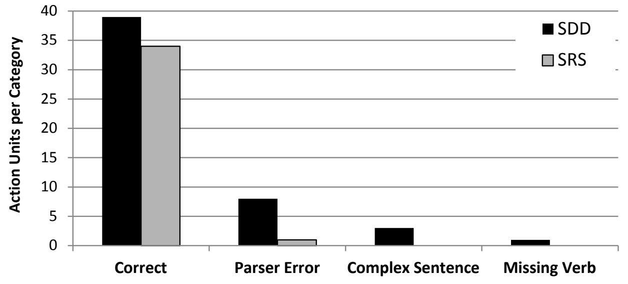


Figure 5: Analysis of Generated Action Units

8.1.1 Experimental Design

To answer this question we used DoCIT to extract action units from 20 randomly selected SRS artifacts and 20 randomly selected SDD artifacts. Each action unit was then evaluated by two researchers in our team. Action units that were correctly extracted according to the definitions in the action unit were classified as **Correctly parsed**. In addition, when the mapping was not successful, we classified the failure causes into the following three categories: (1) **Parser errors** i.e. the parser generated an incorrect tree structure or incorrectly assigned part-of-speech tags and as a result the retrieved action unit was incorrect, (2) **Convoluted sentence** i.e. the sentence was very confusing and produced multiple action units. As a result each action unit was too sparse to support effective link creation. An example of a convoluted sentence is "The system should provide the functionality by which the subsystem under its control can perform.....", (3) **Missing verb** i.e. a verb identified by the parser cannot be found in the set of verb syntactic groups.

8.1.2 Results and Analysis

Results are reported in Figure 5. The SDD sample produced 51 action units, while the SRS dataset produced 35, reflecting the fact that SDD artifacts in this dataset tend to contain longer, more complex descriptions than SRS ones.

Of the 51 SDD action units, 39 were correct, 8 failed due to parser errors, 3 due to convoluted sentences, and 1 due to incomplete verb knowledge in the knowledge base. Of the 35 SRS action units, 34 were correct and the single failure was due to a parser error. Action units were constructed for each identified verb, and no missing ones were observed.

These results show that DoCIT was able to correctly extract 76.5% of the action units from SDD and 97% from SRS. The main cause of failure in both cases was parser errors, which can be addressed in future work through adding additional pattern matching rules to the knowledge base.

8.2 E2: When the KB is Relatively Complete

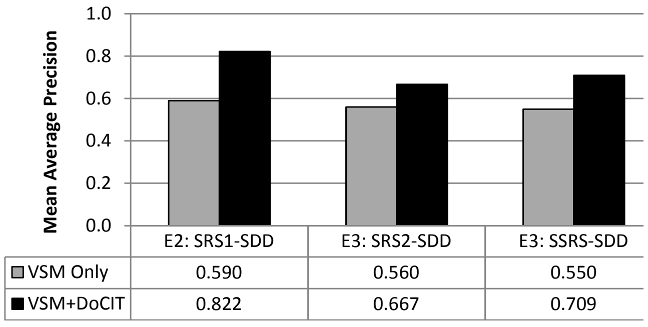
The second experiment evaluated DoCIT's ability to generate trace links when the knowledge base was customized for the dataset. This represents an ideal scenario in which the knowledge base provides relatively complete coverage of the concepts in a dataset. We addressed the research question **(RQ 2:)** "Given a relatively complete knowledge base can DoCIT return an accurate set of trace links?"

8.2.1 Metrics

Results were evaluated using three standard metrics. Recall measures the fraction of relevant links that are retrieved while precision measures the fraction of retrieved links that are relevant. A third metric, Mean Average Precision (MAP), measures the extent to which correct lists are returned at the top of a ranked list. MAP requires an ordered ranking of the links and so is only reported for results that include VSM. As our implementation of MAP examines all correct links it achieves recall of 1.0. The use of MAP as a traceability measure has been advocated in numerous papers [32].

First, the average precision (AP) of each correct link is computed as follows:

$$AP = \frac{\sum_{r=1}^{N}(Precision(r) \times isRelevant(r))}{|RelevantDocuments|} \quad (2)$$



	E2: SRS1-SDD	E3: SRS2-SDD	E3: SSRS-SDD
▨ VSM Only	0.590	0.560	0.550
■ VSM+DoCIT	0.822	0.667	0.709

Figure 6: MAP by query and link for each of the three datasets in Experiments 2 and 3.

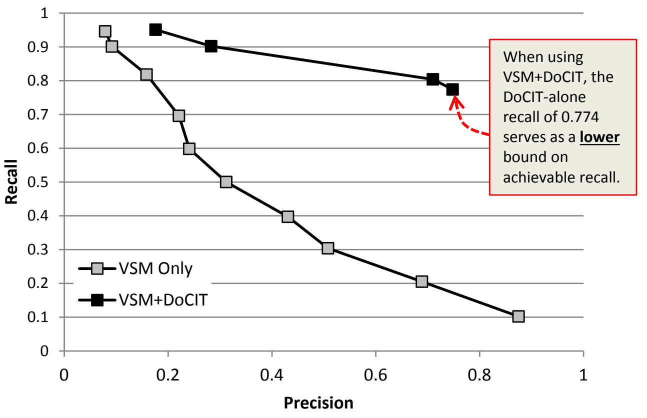


Figure 7: Precision versus recall for Experiment #1

where r is the rank of the target artifact in an ordered list of links, $isRelevant()$ is a binary function assigned 1 if the link is relevant (i.e. marked as correct in the reference set) and 0 otherwise, $P(r)$ is the precision computed after truncating the list immediately below that ranked position, and N is the total number of documents. We compute MAP scores at recall values of 1.00 across all queries as follows:

$$MAP_{@recall=1.00} = \frac{\sum_{q=1}^{Q} AP(q)}{Q} \quad (3)$$

where, q is a single query and Q is the total number of queries. Please note that all subsequent references to MAP imply scores at 100% recall.

8.2.2 Experimental Design

For this experiment we randomly selected 31 SRS artifacts and the complete set of SDDs with links from any SRS artifact. This resulted in 241 SDD artifacts, producing a total of 7,471 potential links. Of these 205 were included in the trace matrix created and provided by our industrial collaborators. We refer to this subset of the dataset as *SRS1*.

Starting with the initial knowledge base, two researchers on the team systematically evaluated the 205 trace links, and following the greedy process prescribed in Section 7.1, systematically added additional concepts to the knowledge base and created new link heuristics. Once each link had been examined and the knowledge base expanded, trace links were generated for the 31 SRS using both VSM and DoCIT.

8.2.3 *Results*

MAP scores are reported in Figure 6 and show that VSM alone returned a MAP of 0.59 compared to 0.82 when used in conjunction with DoCIT. A Wilcoxon Signed Rank test showed a significant difference when the VSM MAP score was compared against VSM+Docit MAP scores for each query at p<0.001. Figure 7 reports precision at various recall levels and also shows a marked improvement in accuracy when VSM+DoCIT is used instead of just VSM. For example, at recall levels of approximately 0.80, VSM returns precision of only 0.17 compared to 0.71 for VSM+DoCIT. This is a non-trivial improvement in accuracy and supports our overall conjecture that a more intelligent domain-specific system has the ability to far outperform a term-based approach such as VSM. We chose to not compare our approach against alternate ontology-based tracing techniques as published results have shown inconsistent improvements over VSM [20].

8.2.4 *Towards Even Greater Accuracy*

Our goal is for DoCIT to ultimately achieve MAP scores close to 1.00 (i.e. to place all of the correct links close to the top of a ranked list) when the underlying knowledge base is complete. However, while DoCIT significantly improves MAP and precision, there is still room for improvement. We therefore analyzed both false positives and false negatives and make the following observations listed in order of their impact: (1) Errors of commission (i.e. false positives) were primarily caused by the fact that DoCIT generates a link if even one pair of action units are related by the link heuristics. This approach fails to filter out links that should have been rejected when leading or trailing conditions do not match. (2) Errors in omission (i.e. false negatives) were primarily caused by incomplete knowledge in the knowledge base. (3) A few errors were caused by parsing mistakes as previously discussed in Section 8.1. All three error types will be addressed in the next phases of our work.

8.3 E3: When the KB is Incomplete

Software projects are rarely static. New artifact types are added, and new domain concepts including syntactic and semantic verb groups may be introduced as the project evolves. As a result we can never expect, and certainly not guarantee, a complete knowledge base. This introduces the research question **(RQ3:)** "When a knowledge base is incomplete does DoCIT still improve trace accuracy?"

8.3.1 *Experimental Design*

We conducted an experiment against two different datasets for which the KB provided only partial coverage.

For the first dataset we randomly selected 31 different SRS artifacts (i.e. not ones used in the previous experiment) and the same complete set of 241 SDDs. This produced a total of 7,471 potential links of which 157 were marked as correct by our industrial collaborators. We refer to this subset of the dataset as *SRS2*. Because the initial KB was constructed through examining a different subset of SRS-SDD trace links (i.e. SRS1), we would expect partial knowledge coverage of concepts in SRS2 in the KB.

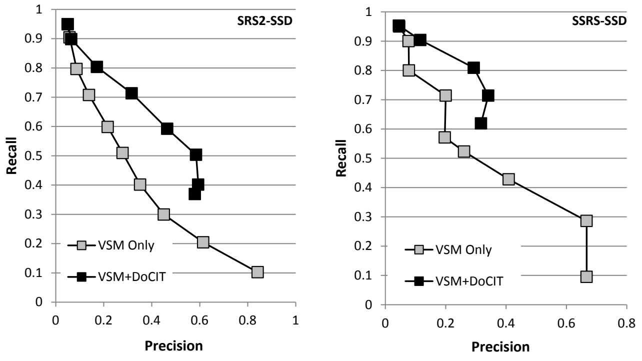


Figure 8: Incomplete knowledge for Experiment #3.

For the second dataset we selected 10 SSRS artifacts and traced them to the same 241 SDDs. SSRS represented a completely new type of artifact from the transportation dataset that had not been previously been used to build the Knowledge Base. There were a total of 21 correct trace links out of the potential set of 2,410. We refer to this dataset as *SSRS*. As SSRS represents an entirely new artifact type that was not used in the KB construction, we expect less knowledge coverage than for SRS2.

The tracing experiment was conducted in the same way as the previous one, with the important difference that no new information was added to the knowledge base. Traces were generated based purely upon previously defined facts in the knowledge base.

8.3.2 *Results and Analysis*

Results are also reported in Figure 6. In the case of SRS2, MAP increased from 0.560 to 0.667 and for SSRS it increased from 0.550 to 0.709. Results for SSRS were better than we had anticipated because DoCIT improved five of the ten queries, and in three of these cases returned perfect MAP scores. A Wilcoxon Signed Rank test showed a significant difference for SRS2 at p=0.013, but did not show significance for SSRS at p=0.056. Improvements in precision vs. recall were not as marked as the improvements for SRS1, however, as depicted in Figure 8. At fixed recall values precision was markedly higher for VSM+DoCIT than for DoCIT alone in both datasets. We therefore conclude that even though DoCIT's knowledge base was incomplete with respect to the two datasets, the quality of the links still showed marked improvements.

8.4 E4: Greedy Link Heuristic Construction

Given our greedy approach to the construction of link heuristics, we explored the research question **(RQ4:)** "Are all existing link heuristics globally effective?". To answer this question we simply counted the number of times each heuristic was applied for datasets SRS1 and SRS2, versus the number of times it resulted in a correct link.

Results show that 51% produced correct links with 100% accuracy, an additional 35% produced correct links with at least 70% accuracy, and the remaining 14% produced correct links with at least 33% accuracy. Even in the worst case of 33% accuracy, they tend to outperform the accuracy of term-based retrieval approaches. Nevertheless this needs further investigation and global optimization will be addressed in future work.

9. THREATS TO VALIDITY

Our results show that integrating DoCIT with VSM led to significant improvements in accuracy of the generated trace links, especially when an adequate knowledge base was present. Construct validity threats were primarily addressed through using utilizing metrics broadly accepted to differentiate between high and low quality links [14]. The generated links were compared against a golden-answer set provided by our industrial collaborators. In cases where additional links were identified as potentially valid, they were reviewed by the industrial collaborators, and when deemed correct, were added to the answer set. Nevertheless, it is possible that additional correct links exist but were not found. Similarly internal validity was addressed through conducting a controlled experiment to compare results obtained using VSM alone versus VSM plus DoCIT.

The major threat of our study focuses around external validity which speaks to the generalizability of the approach. Our end goal is to build an intelligent traceability system that can be reused across a very narrow set of systems in the same regulatory domain. However, given the significant time and effort it took to obtain the datasets, to compare alternate techniques and ultimately develop a solution based on action units and link heuristics, to build a domain ontology, and then to carefully validate results, we only able to conduct our study for a single project in a highly focused domain. The two primary threats to generalizability are (1) whether our action unit extractor will work on a variety of artifacts and styles, across other projects, and in other domains, and (2) whether the link heuristics can be generalize across other projects. On the other hand, we do not yet claim generalizability. The goal of this research was to demonstrate that an intelligent traceability solution could significantly improve trace results in a case project. We leave the task of generalizability to future work.

10. RELATED WORK

Related work falls under two major categories of transforming unstructured text into formal specifications and use of ontology to support the tracing process.

Breaux, Anton, et al. have developed techniques for transforming natural language text into formal or semi-formal representations [9, 24]. Their focus has been on the areas of security, privacy [9], and accessibility [8] and has emphasized elements such as *rights*, *obligations*, and *permissions*. Their work evaluated compliance of industrial requirements to accessibility standards by extracting requirements from the standards using an approach they refer to as the frame-based method, and then performing a gap analysis between the two sets of requirements [8]. While they have made some attempts to automate the extraction of various elements from text, their approach is primarily manual in nature, and does not address the issues of automating the traceability process. Spanoudakis et al. [30] developed a rule-based approach for generating traceability relations between artifacts; however their rules were primarily syntactical in nature and did not attempt the kind of semantic analysis used by DoCIT.

Several researchers have previously explored the use of ontology for tracing purposes. Kof et al. [18] extracted domain-specific concepts from the set of traceable artifacts, and then used WordNet to find synonyms in order to map similar concepts and to establish trace links [18]; however they did not provide a rigorous analysis to evaluate whether the use of the ontology actually improves the quality of the generated trace links. Li et al. also explored the use of an ontology for traceability purposes [20]. Their approach used knowledge from the ontology to establish connections between concepts in source and target artifacts. The strength of the dependency was related to the distance between concepts in the ontology, and the overall similarity score between pairs of artifacts was increased accordingly. Both Kof and Li therefore used ontology to augment the basic approach that computes the similarity between two artifacts based on the similarity of their terms. However, unlike DoCIT, their approach did not attempt to establish semantically-aware link heuristics.

Existing tracing tools such as Poirot [21] include capabilities to expand acronyms and utilize lists of matching words in an attempt to bridge the semantic gap. However, they fail to capture the rich set of relationships and inference rules that can be modeled in an expert system. Zhang et al. published a paper entitled "Ontological approach for the semantic recovery of trace links between software artefacts" [33]; however the ontology is used only to represent concepts such as *design patterns*, *sentences*, *paragraphs*, and *classes*, as opposed to domain concepts such as *vehicle* or *signal* found in the domain being traced. Assawamekin et al. utilized Natural Language Processing techniques to construct separate ontologies to represent different stakeholder perspectives and then to discover mappings between them [4]. The ontology is expressed in first-order logic and the mapping is solved by a SAT-solver. However the authors provided only a simple example, and did not empirically evaluate their approach.

Ontologies have been used in the requirements domain for several different purposes, and several techniques therefore exist for extracting an ontology from a requirements specifications [12, 11, 25, 4]. We plan to examine these automated ontology extracting techniques in our future work.

11. CONCLUSION

The work described in this paper has tackled the significant challenge of automating the creation of trace links in a safety-critical, communication and control domain. We successfully demonstrated that within a large industrial project, given a knowledge base with sufficient coverage of the concepts, DoCIT was able to significantly improved the accuracy of generated trace links. These results support our earlier conjecture that intelligent traceability provides a viable approach for overcoming the term-mismatch barrier of existing tracing techniques and ultimately of delivering automated tracing techniques that can be effective for industry.

In ongoing work, we are developing techniques for automating the creation and expansion of a DoCIT knowledge base for new domains. Our work will focus on discovering the right semantic and syntactic groupings, correct dependency mappings, new relationships, and appropriate trace link heuristics in a more dynamic way for a given domain. We also plan to evaluate the efficacy across multiple domains in order to demonstrate broader generalizability.

12. ACKNOWLEDGMENTS

The work in this paper was partially funded by the US National Science Foundation Grant CCF:1319680.

13. REFERENCES

[1] CoEST: Center of excellence for software traceability, http://www.CoEST.org.

[2] Owl2 web ontology language primer, June 2013.

[3] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Trans. Softw. Eng.*, 28(10):970–983, 2002.

[4] N. Assawamekin, T. Sunetnanta, and C. Pluempitiwiriyawej. Ontology-based multiperspective requirements traceability framework. *Knowl. Inf. Syst.*, 25(3):493–522, 2010.

[5] H. Asuncion and R. N. Taylor. Capturing custom link semantics among heterogeneous artifacts and tools. In *Traceability in Emerging Forms of Software Engineering*, 2009.

[6] B. Berenbach, D. Gruseman, and J. Cleland-Huang. Application of just in time tracing to regulatory codes. In *Proceedings of the Conference on Systems Engineering Research*, 2010.

[7] E. Bouillon, P. Mäder, and I. Philippow. A survey on usage scenarios for requirements traceability in practice. In *Requirements Engineering: Foundation for Software Quality*, volume 7830 of *Lecture Notes in Computer Science*, pages 158–173. Springer, 2013.

[8] T. D. Breaux, A. I. Antón, K. Boucher, and M. Dorfman. Legal requirements, compliance and practice: An industry case study in accessibility. In *RE*, pages 43–52, 2008.

[9] T. D. Breaux, A. I. Antón, and J. Doyle. Semantic parameterization: A process for modeling domain descriptions. *ACM Trans. Softw. Eng. Methodol.*, 18(2):5:1–5:27, Nov. 2008.

[10] J. Cleland-Huang and J. Guo. Towards more intelligent trace retrieval algorithms. In *(RAISE) Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, 2014.

[11] R. Gacitua and P. Sawyer. Ensemble methods for ontology learning - an empirical experiment to evaluate combinations of concept acquisition techniques. In *ACIS-ICIS*, pages 328–333, 2008.

[12] R. Gacitua, P. Sawyer, and P. Rayson. A flexible framework to experiment with ontology learning techniques. *Knowl.-Based Syst.*, 21(3):192–199, 2008.

[13] J. Guo, J. Cleland-Huang, and B. Berenbach. Foundations for an expert system in domain-specific traceability. In *RE*, 2013.

[14] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram. Advancing candidate link generation for requirements tracing: The study of methods. *IEEE Trans. Softw. Eng.*, 32(1):4–19, 2006.

[15] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram. Advancing candidate link generation for requirements tracing: The study of methods. *IEEE Trans. Software Eng.*, 32(1):4–19, 2006.

[16] P. Jackson. *Introduction To Expert Systems (3 ed.)*. Addison Wesley, 1998.

[17] D. Klein and C. D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, pages 423–430, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[18] L. Kof, R. Gacitua, M. Rouncefield, and P. Sawyer. Concept mapping as a means of requirements tracing. In *MaRK'10)*, 2010.

[19] N. G. Leveson. *Safeware, System Safety and Computers*. Addison Wesley, 1995.

[20] Y. Li and J. Cleland-Huang. Ontology-based trace retrieval. In *Traceability in Emerging Forms of Software Engineering (TEFSE2013*, San Francisco, USA, May 2009.

[21] J. Lin, C. C. Lin, J. Cleland-Huang, R. Settimi, J. Amaya, G. Bedford, B. Berenbach, O. B. Khadra, C. Duan, and X. Zou. Poirot: A distributed tool supporting enterprise-wide automated traceability. In *RE*, pages 356–357, 2006.

[22] P. Mäder, O. Gotel, and I. Philippow. Motivation matters in the traceability trenches. In *Proceedings 17th International Conference on Requirements Engineering*, pages 143–148. IEEE, 2009.

[23] P. Mäder, P. L. Jones, Y. Zhang, and J. Cleland-Huang. Strategic traceability for safety-critical projects. *IEEE Software*, 30(3):58–66, 2013.

[24] J. Maxwell and A. Anton. Developing production rule models to aid in acquiring requirements from legal texts. In *Requirements Engineering Conference, 2009. RE '09. 17th IEEE International*, pages 101–110, 2009.

[25] A. D. Nicola, M. Missikoff, and R. Navigli. A software engineering approach to ontology building. *Inf. Syst.*, 34(2):258–275, 2009.

[26] T. Parsons. Thematic relations and arguments. *Linguistic Inquiry*, 26(4):635–662, 1995.

[27] P. Rempel, P. Mäder, and T. Kuschke. An empirical study on project-specific traceability strategies. In *21st Intn'l Requirements Engineering Conference (RE'13)*, Rio de Janeiro, Brasil, July 2013.

[28] P. Rempel, P. Mäder, T. Kuschke, and J. Cleland-Huang. Mind the gap: Assessing the conformance of software traceability to relevant guidelines. In *36th Intn'l Conf. on Software Engineering (ICSE)*, 2014.

[29] P. Shvaiko and J. Euzenat. Ontology matching: State of the art and future challenges. *IEEE Trans. Knowl. Data Eng.*, 25(1):158–176, 2013.

[30] G. Spanoudakis, A. Zisman, E. Pérez-Miñana, and P. Krause. Rule-based generation of requirements traceability relations. *Journal of Systems and Software*, 72(2):105–127, 2004.

[31] Stanford. Protégé: Open source ontology editor, 2013.

[32] S. K. Sundaram, J. H. Hayes, A. Dekhtyar, and E. A. Holbrook. Assessing traceability of software engineering artifacts. *Requir. Eng.*, 15:313–335, September 2010.

[33] Y. Zhang, R. Witte, J. Rilling, and V. Haarslev. Ontological approach for the semantic recovery of traceability links between software artefacts. *Software, IET*, 2(3):185 –203, june 2008.

[34] X. Zou, R. Settimi, and J. Cleland-Huang. Improving automated requirements trace retrieval: a study of term-based enhancement methods. *Empirical Software Engineering*, 15(2):119–146, 2010.