ModelWriter: Text and Model-Synchronized Document Engineering Platform

Ferhat Erata[*§], Claire Gardent[†], Bikash Gyawali[†], Anastasia Shimorina[†], Yvan Lussaud[||], Bedir Tekinerdogan[*], Geylani Kardas[¶**] and Anne Monceaux[‡]

[*]Information Technology Group, Wageningen University and Research Centre, The Netherlands
[†]CNRS, LORIA, UMR 7503 Vandoeuvre-les-Nancy, F-54500, Nancy, France
[‡]System Engineering Platforms, Airbus Group Innovations, Toulouse, France
[§]UNIT Information Technologies R&D Ltd., Izmir, Turkey
[¶]Ege University, International Computer Institute, Izmir, Turkey
[**]KoçSistem Information and Communication Services Inc. Istanbul, Turkey
[||]OBEO, Nantes, France

***Abstract*—The ModelWriter platform provides a generic framework for automated traceability analysis. In this paper, we demonstrate how this framework can be used to trace the consistency and completeness of technical documents that consist of a set of System Installation Design Principles used by Airbus to ensure the correctness of aircraft system installation. We show in particular, how the platform allows the integration of two types of reasoning: reasoning about the meaning of text using semantic parsing and description logic theorem proving; and reasoning about document structure using first-order relational logic and finite model finding for traceability analysis.**

https://itea3.org/project/modelwriter.html

I. Introduction

Due to the complexity of software systems that commands, controls, and monitors safety-critical functions in airborne systems and the compliance with the DO-187C [1] standard, there is an increased need for sophisticated and highly automated tools for the analysis of the tracing among software artifacts (e.g., requirements, architecture models, source codes and test cases) to keep them synchronized and consistent during the development process. In this context, traceability [2] not only establishes and maintains consistency between these artifacts but also ensures that each requirement is tested, that each line of source code has a purpose for the fulfillment of a requirement, and so forth.

The implementation of traceability is highly contextual as the key or certification artifacts produced or used along the process differ depending on the product or the organization [3]. For that reason, the ModelWriter platform provides means for users to specify which artifacts they want to precisely identify and monitor and what meaning they want to assign to the traces between these artifacts.

The considered artifacts represented in our context by trace locations might be of different levels of granularity, ranging from a complete document or model to fragments of text or code. Focusing on documents and text, both the structure and the content might be used to reason about traceability.

To this end, the ModelWriter platform provides a generic traceability analysis applicable to text and model artifacts to synchronize them. Trace locations can be fragments of text, elements of an architectural model, and parts of program codes. Traces are relations between trace locations. The ModelWriter platform allows axiomatization of these relations and reasoning about them, i.e. supporting traceability analysis for different types of artifacts.

In this paper, we focus on demonstrating the features of the ModelWriter platform for traceability analysis applied to technical documentation. A particular challenge in this use case is to take into account the meaning of natural language. We integrate techniques from Natural Language Processing (NLP) and Automated Reasoning to reason both about the meaning and about the structure of text. We use techniques from semantic parsing to assign formal meaning representations to NL text. We then use techniques from theorem proving and model building to infer traceability relations between text fragments (here System Installation Design Principles), to check consistency and to ensure completeness.

II. The Airbus SIDP Usecase

We illustrate the workings of the ModelWriter platform based on a set of System Installation Design Principles (SIDP) used by Airbus to ensure the correctness of aircraft design. An SIDP rule is an installation requirement specifying properties which must be fulfilled for the system to be well-formed. In this usecase, SIDPs are trace locations and we define five types of trace links between these trace locations, namely CONTAINS, REFINES, CONFLICTS, EQUALS, and REQUIRES. In the following, we informally give the meaning of the trace-types adopted from the work of Goknil et. al. [4]. A formal semantics is provided in Section III-C.

Rule r_1 *contains Rule* $r_2 \dots r_n$ if $r_2 \dots r_n$ are parts of the whole r_1 (part-whole hierarchy). The contained rule is a subrule of the containing rule. *Rule* r_1 *refines* another *Rule* r_2 if r_1 is derived from r_2 by adding more details to its properties. The refined rule can be seen as an abstraction of the detailed rules. In Fig. 1 *contains* and *refines* traces are illustrated. Each box represents a property of the corresponding rule.

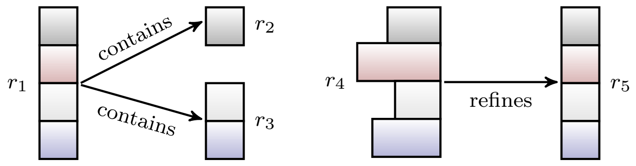


Fig. 1. r_1 *contains* r_2 and r_3, r_4 *refines* r_5

Rule r_1 *conflicts* with *Rule* r_2 if the fulfillment of r_1 excludes the fulfillment of r_2 and vice versa. The existence of a conflict trace indicates an inconsistency between two rules. *Rule* r_1 *equals* to *Rule* r_2 if r_1 states exactly the same properties with their constraints with r_2 and vice versa. *Rule* r_1 *requires Rule* r_2 if r_1 is fulfilled only when r_2 is fulfilled. The required rule can be seen as a pre-condition for the requiring rule. In the following Fig. 2 *conflicts*, *equals* and *requires* traces are illustrated.

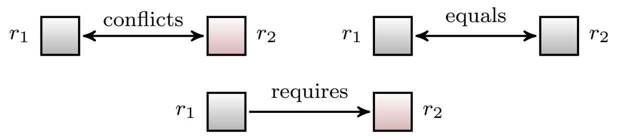


Fig. 2. Illustration of "*conflicts, equals and requires*"

Given a set of SIDPs, the ModelWriter platform can be used to check completeness and consistency as follows. First, the SIDPs are parsed and assigned Description Logic (DL) formulae representing their meaning (cf. Section III-A). Second, traces are either manually specified by the end user or inferred using semantic parsing and DL theorem proving (cf. Section III-B). Third, new trace links can be inferred from existing ones using Relational Logic (RL) (cf. Section III-C) and Model Finding (cf. Section III-D). Importantly, the inference of trace links allows for the detection of missing or inconsistent SIDPs.

Table I illustrates this process. Given the SIDPs r_1-r_6, CONFLICTS and REFINES trace links are first inferred using semantic parsing and the Hermit theorem prover [5] (DL lines in the table).

TABLE I
EXAMPLE SIDPS AND INFERENCE OF TRACE LINKS

Nr.	Artifact Annotations (Trace-locations)
r$_1$	Bracket shall be used in hydraulic area Alpha
r$_2$	Adhesive bonded bracket shall be used in hydraulic area
r$_3$	Adhesive bonded bracket shall be used in hydraulic area Alpha
r$_4$	Bracket shall be used in hydraulic area
r$_5$	Bracket shall be installed in hydraulic area
r$_6$	Bracket shall be installed in fuel tank

Nr.	Inferred Traces	Nr.	Inferred Traces
DL$_1$	$conflicts(r_5, r_6)$	RL$_1$	$conflicts(r_6, r_4)$
DL$_2$	$refines(r_3, r_2)$	RL$_2$	$requires(r_1, r_5)$
DL$_3$	$refines(r_2, r_4)$	RL$_3$	$conflicts(r_6, r_1)$
DL$_4$	$refines(r_1, r_4)$	RL$_4$	$requires(r_2, r_5)$
DL$_5$	$requires(r_4, r_5)$	RL$_5$	$conflicts(r_2, r_6)$

For example, the DL formulae obtained by parsing sentences r_5 and r_6 conflict with each other because the underlying ontology to which these axioms are added specifies that concepts "hydraulic area" and "fuel tank" are disjoint. Similarly, the axiom obtained for the sentence r_2 refines the axiom obtained for r_4 because the ontology specifies that "Bracket" is a sub concept of "Adhesive bonded bracket". In Fig. 3, Table I is represented as a digraph model in which the nodes represent trace-locations, i.e. SIDP rules listed in the table, and edges represents traces. A red edge specifically corresponds to the trace inferred using semantic parsing and DL theorem proving. The black one is an example trace, $refines(r_3, r_6)$ created by the user manually.

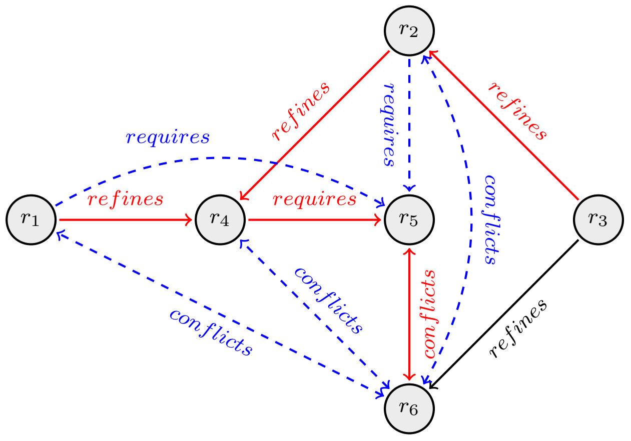


Fig. 3. Inferred Traces (red traces indicate reasoning using DL, blue indicates reasoning using RL, the black one indicates a manual trace)

Later, additional trace links are inferred using Relational Model Finding (RL lines in the Table I and dashed blue edges on Fig. 3). For instance, as part of the trace semantics of this use case, according to the axiom schema (3) formalized in Section III-C where *a*, *b* and *c* are artifact elements, if *a refines*, *requires* or *contains b*, while *b conflicts* with *c*, then *a* also *conflicts* with *c*. In this way, ModelWriter generates CONFLICTS traces such that combination of $conflicts(r_5, r_6)$ and $requires(r_4, r_5)$ makes $conflicts(r_6, r_4)$; on the other hand, according to axiom schema (1) described in Section III-C, the combination of $refines(r_2, r_4)$ and $requires(r_4, r_5)$ generates $requires(r_2, r_5)$ corresponding to the patterns shown in Fig. 4.

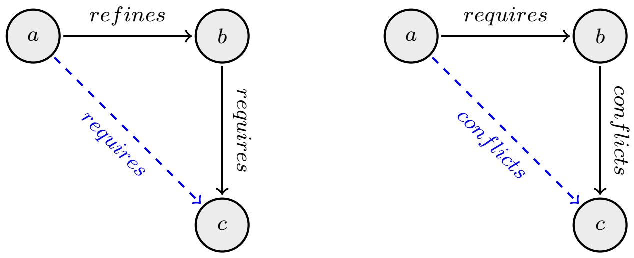


Fig. 4. Inferring "*requires*" with "*refines*" and inferring "*conflicts*"

Finally, in this example, DL-based reasoning process inferred only one CONFLICTS trace using the meaning of the sentences, i.e. r_5 conflicts with r_6 whereas the ModelWriter detects three more *conflicts* traces using the meaning of trace types by means of RL-based reasoning on top of DL-based reasoning. As a result, it can be seen that not only r_5 and r_6 but also r_4, r_1, and r_2 are inconsistent.

III. Overview of the Approach

We now describe the four main modules making up the ModelWriter platform. Section III-A introduces the semantic parser, i.e., the module that converts text to Description Logic formulae. Section III-B explains how the Hermit reasoner can be used to detect REFINES, CONFLICTS and EQUALS trace links between text fragments (here, SIDPs). Section III-C shows how Alloy formalism [6] can be customized to axiomatize trace types and semantics. Finally, Section III-D explains how the KodKod model finder [7] is used to infer new trace links between SIDPs to detect the inconsistent SIDPs.

A. Mapping Text to Description Logic Formulae

The semantic parser used in ModelWriter to convert text to DL formulae is described in details in [8]. In what follows, we briefly summarize its working and some evaluation results on a set of 960 SIDPs used for testing.

The ModelWriter semantic processing framework combines an automatically derived lexicon, a small hand-written grammar, a parsing algorithm to convert text to DL formulae and a generation algorithm to convert DL formulae to text. This framework is modular and robust. It is modular in that, different lexicons or grammars may be plugged to meet the requirements of the semantic application being considered. For instance, the lexicon (which relates words and concepts) could be built using a concept extraction tool, i.e. a text mining tool that extracts concepts from text (e.g., [9]). And the grammar could be replaced by a grammar describing the syntax of other document styles such as cooking recipes. It is robust in that, in the presence of unknown words, the parser can skip words and deliver a connected (partial) parse. Fig. 5 outlines our approach showing the interaction of various components. The generation algorithm can be used to automatically synchronize documents with models (e.g., by generating a text verbalising the meaning of a DL constraint added to the model). It is also used to verify the quality of the DL formules derived by parsing: given a sentence S and the DL formula ϕ derived for S by the parser, are the sentences S' that can be generated from ϕ similar to S?

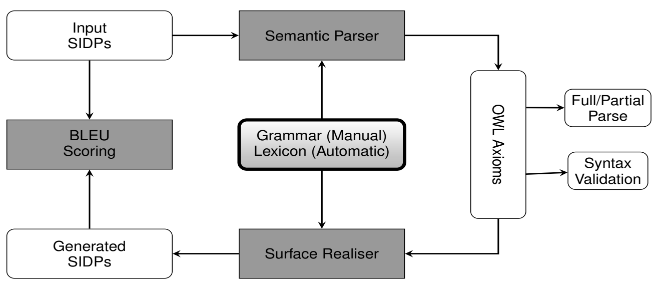


Fig. 5. Parsing and Generation of Airbus SIDPs.

The lexicon maps verbs and noun phrases to grammar rules and to complex and simple concepts respectively. Fig. 6 shows an illustrative example with a lexical entry on the left and the corresponding grammar unit on the right. During generation/parsing, the semantic literals listed in the lexicon (here, `Use` and `useArg2inv`) are used to instantiate the variables (here, `A2` and `Rel`) in the semantic schema (here, `L`$_0$`:subset(X,L`$_1$`) L`$_2$`:exists(A2,L`$_3$`) L`$_3$`:Rel(Y)`). Similarly, the Anchor value (*used*) is used to label the terminal node marked with the anchor sign (◇) and each coanchor is used to label the terminal node with corresponding name. Thus, the strings *shall* and *be* will be used to label the terminal nodes $V1$ and $V2$ respectively.

Semantics:
Rel = Use
A2 = useArg2inv
Tree: nx0V
Anchor: *used*
Coanchor: V1 $\rightarrow$ *shall/V*
Coanchor: V2 $\rightarrow$ *be/V*

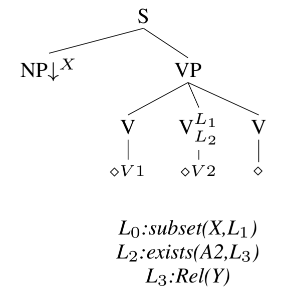


Fig. 6. Example Lexical Entry and Grammar Unit

The grammar provides a declarative specification of how text relates to meaning (as represented by OWL DL [10] formulae). We use a Feature-Based Lexicalised Tree Adjoining Grammar (FB-LTAG) [11] augmented with a unification-based flat semantics. Fig. 7 shows an example FB-LTAG for the words “not", “pipes" and “shall be used". An FB-LTAG tree is a set of initial and auxiliary trees which have been lexicalised using the lexicon and can be combined using either substitution or adjunction. Auxiliary trees are trees such as the tree for “not" which contains a foot node (marked with *) whose category (here AUX) matches that of the root node. Initial trees are trees such as that of “pipes" and “shall be used" whose terminal nodes may be substitution nodes (marked with $\downarrow$). Substitution inserts a tree with root category C into a substitution node of the same category. For instance, the tree for “pipes" may be substituted in the $NP^Y_\downarrow$ node of the “shall be used" tree. Adjunction inserts an auxiliary tree with foot node category C into a tree at a node of category C. For instance, the tree for “not" may be adjoined into the tree for “shall be used" at the AUX node.

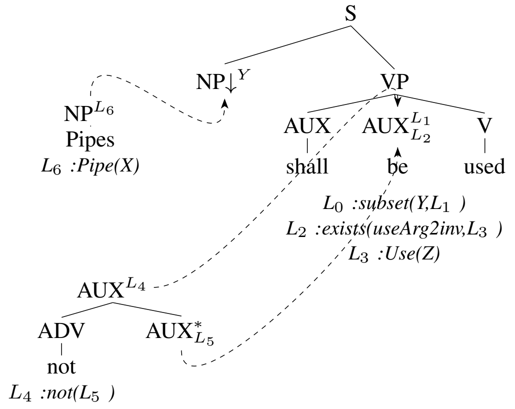
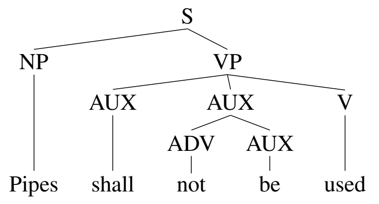


Fig. 7. Example FB-LTAG with Unification-Based Semantics. The variables decorating the tree nodes (e.g., X) abbreviate feature structures of the form $[idx : X]$ where X is a unification variable.

The parser and the generator exploit the grammar and the lexicon to map natural language to OWL DL formulae (semantic parsing) and OWL DL formulae to natural language (generation) respectively. For instance, given the sentence "Pipes shall not be used", the parser will first select the grammar trees associated with "Pipes", "shall be used" and "not" and then combines these trees using substitution and adjunction. As shown in Fig. 8, the semantics derived for the input sentence is then the union of the semantics of these trees modulo unification. Conversely, given the flat semantics shown in the figure the generator will generate the sentence "Pipes shall not be used", by first selecting grammar trees whose semantics subsumes the input and then combining them using substitution and operation. The generated sentences are given by the yield of the derived trees whose root is of category S (sentence) and whose semantics is exactly the input semantics.



L_6:Pipe(X) L_0 :subset(L_6,L_4) L_4:not(L_5)
L_5:exists(useArg2inv,L_3) L_3:Use(Z)

$$Pipe \sqsubseteq \neg\exists useArg2^{-}.(Use)$$

Fig. 8. Derived Tree. The flat semantics representation produced by the grammar is equivalent to the Description Logic Formula shown.

While the grammar integrates a so-called flat semantics, as shown in Fig. 9, there is a direct translation from this semantics to OWL functional syntax. Further details about Semantic Parser can be found at:

https://modelwriter.github.io/semanticparser

B. Inferring Traces using DL Theorem Proving

We use Hermit theorem prover to detect inconsistencies, entailment and equivalence between two SIDPs s_1 and s_2. Given the DL formulae ϕ_1 and ϕ_2 associated by the semantic parsing process to s_1 and s_2, we determine these relations as follows: (i) if $\phi_1 \sqcap \phi_2$ is not satisfiable, we infer a CONFLICTS trace between s_1 and s_2, (ii) if $\neg\phi_1 \sqcup \phi_2$ is satisfiable, we infer a REQUIRES trace between s_1 and s_2, and (iii) if $\phi_1 \equiv \phi_2$ is satisfiable, we infer an EQUALS trace between s_1 and s_2.

C. Formal Semantics of Trace-types

Tarski is the module of ModelWriter for automated reasoning about traces based on configurable trace semantics, recently described in [12] and demonstrated in [13]. The tool provides an enhanced text editor to allow users to define new trace types in a restricted form of Alloy [6] to express complex constraints among traces.

In the following, we axiomatize trace semantics based on the informal definition explained in Section II using *First-order Predicate Logic (FOL)* with the signature:

$$\Sigma_T : \{=, \in\} \cup \Sigma_T^1 \cup \Sigma_T^2$$
$$\Sigma_T^1 : \{Artifact, Requirement, Specification\}$$
$$\Sigma_T^2 : \{requires, refines, contains, equals, conflicts\}$$

Σ_T^1 is the set of unary predicate symbols and Σ_T^2 is the set of binary predicate symbols. For simplicity, we assume that the universe only consists of the type, *Artifact* which is partitioned into disjoint subsets of *Requirement* and *Specification*. From now on, A represents the set of *Artifacts*. The symbols $=$ and $\in$ are interpreted and represent *equality* and *membership* respectively. In the following, several axiom schemas are listed to formalize Traceability Theory, that is used in the SIDP case.

Reasoning about REQUIRES traces is stated as follows:

$$\vdash \forall a,b,c \in A \mid (a,b) \in \Box \wedge (b,c) \in \triangle \rightarrow (a,c) \in \triangle \quad (1)$$
$$\vdash \forall a,b,c \in A \mid (a,b) \in \triangle \wedge (b,c) \in \Box \rightarrow (a,c) \in \triangle \quad (2)$$
$$\text{where } \Box \in \{\text{requires}, \text{refines}, \text{contains}\} \text{ and } \triangle := \text{requires}$$

The following axiom schema is being used for generating CONFLICTS traces.

$$\vdash \forall a,b,c \in A \mid (a,b) \in \Box \wedge (b,c) \in \triangle \rightarrow (a,c) \in \triangle \quad (3)$$
$$\vdash \forall a \in A \mid (a,a) \in \triangle \quad (4)$$
$$\text{where } \Box \in \{\text{requires}, \text{refines}, \text{contains}\} \text{ and } \triangle := \text{conflicts}$$

Reasoning about EQUALS traces:

$$\vdash \forall a,b,c \in A \mid (a,b) \in \text{equals} \wedge (b,c) \in \Box \rightarrow (a,c) \in \Box \quad (5)$$
$$\vdash \forall a,b,c \in A \mid (a,b) \in \text{equals} \wedge (c,b) \in \Box \rightarrow (c,a) \in \Box \quad (6)$$
$$\vdash \forall a \in A \mid (a,a) \in \text{equals} \quad (7)$$
$$\text{where } \Box \in \{\text{contains}, \text{requires}, \text{refines}, \text{conflicts}\}$$

In the following axiom schema, transitivity (8) is used for reasoning new traces, whereas anti-symmetry (9) and irreflexivity (10) are used to check consistency.

$$\vdash \forall a,b,c \in A \mid (a,b) \in \Box \wedge (b,c) \in \Box \rightarrow (a,c) \in \Box, \quad (8)$$
$$\vdash \forall a,b \in A \mid (a,b) \in \Box \wedge (b,a) \in \Box \rightarrow a = b, \quad (9)$$
$$\vdash \forall a \in A \mid (a,a) \notin \Box, \quad (10)$$
$$\text{where } \Box \in \{\text{contains}, \text{requires}, \text{refines}\}$$

CONTAINS traces is left-unique (injective relation) in some scenarios that induces an inconsistency when transitivity axiom (8) for CONTAINS is instantiated in the specification.

$$\vdash \forall a,a',b \in A \mid (a,b) \in \Box \wedge (a',b) \in \Box \rightarrow a = a' \quad (11)$$
$$\text{where } \Box := \text{contains}$$

We encode above axioms in First-order Relational Logic using the Tarski's text editor to configure the Tarski module.

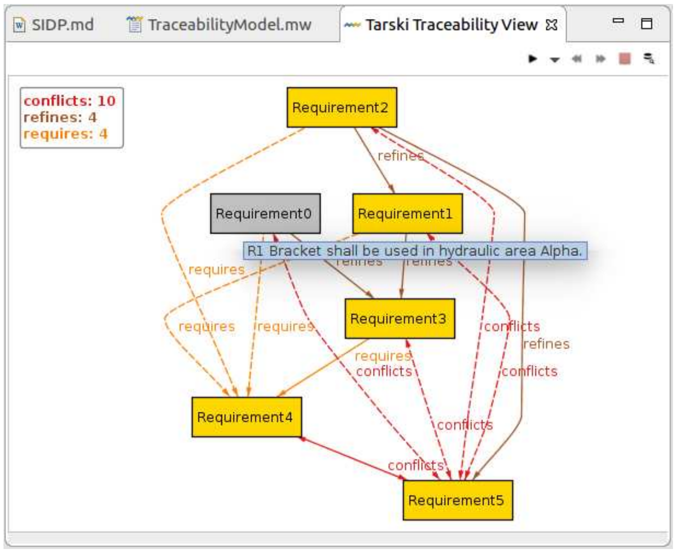
D. Inferring Trace Links using Model Finding

We employ Kodkod [7], [14], an efficient SAT-based constraint solver for FOL with relational algebra and partial models, for automated trace reasoning using the trace semantics that user provides. Once the user performs reasoning operations about traces, the result is reported back to the user by dashed traces as shown in Fig. 10. If there exists different solutions, the user can traverse them back and forth. He can also accept the inferred traces, and perform another analysis operation including inferred traces. Further details about Tarski can be found at:

https://modelwriter.github.io/Tarski/

$$\tau(\phi) = \begin{cases} \text{ObjectSomeValuesFrom(:R } \tau(C)) & \text{if } \phi = l_i : exists(R, l_j) \;\; l_j : C \\ \text{SubClassOf}(\tau(C_1)\ \tau(C_2)) & \text{if } \phi = l_i : subset(l_j, l_k) \;\; l_j : C_1 \;\; l_k : C_2 \\ \text{ObjectIntersectionOf}(\tau(C_1)\ \tau(C_2)) & \text{if } \phi = l_i : and(l_j, l_k) \;\; l_j : C_1 \;\; l_k : C_2 \\ (\tau(C1) \sqcap \tau(C2)) & \text{if } \phi = l_i : and(l_j, l_k) \;\; l_j : C1 \;\; l_k : C2 \\ (\tau(C1) \sqcup \tau(C2)) & \text{if } \phi = l_i : or(l_j, l_k) \;\; l_j : C1 \;\; l_k : C2 \\ \text{not}(\tau(C)) & \text{if } \phi = l_i : not(l_j) \;\; l_j : C \\ \text{R}^{-} & \text{if } \phi = Rinv \\ \text{C} & \text{if } \phi = l_i : C(x) \end{cases}$$

Fig. 9. Mapping Flat Semantics to Owl Functional Syntax



R1 Bracket shall be used in hydraulic area Alpha.

Fig. 10. Inferred Relations based on the current snapshot

IV. Evaluation

We evaluate the Semantic Parsing approach of ModelWriter on a dataset of 960 SIDPs provided by Airbus which demonstrates (i) that the approach is robust (97.50% of the SIDPs can be parsed) and (ii) that DL axioms assigned to full parses are likely to be correct[1]. Regarding inference phase, since we observed that DL-based reasoning is relatively faster than SAT-based reasoning in the context of SIDPs, we only focus on the Tarski module to evaluate the performance of ModelWriter approach. Table II shows the solving results of three configurations of the formal trace specification running with Alloy Analyzer [6], KodKod [7] and Z3 [15]. The Minisat [16] SAT Solver is chosen for both Alloy (alloy4.2-2015-02-22.jar) and KodKod (Kodkod 2.1) solvers. From Alloy to SMT solver translation for these cases, we employ the translation method proposed by El Ghazi et.al. [17] and the problems are encoded in SMT-LIB [18] syntax which is fed into Z3 solver. Transitive closure and integer arithmetic are not used in these use cases to fairly benchmark the results with the SMT solver. In SMT-LIB, the logic is set for Equality Logic with Uninterpreted Functions (UF). Evaluation results are obtained on a machine, that runs 64 bit Debian linux operating system with 8 GB of memory and 2.90GHz Intel i7-3520M CPU. Solving times are indicated in milliseconds. The best results are obtained by the direct use of KodKod API since to find satisfiable models, KodKod allows us to configure lower and upper bounds for the solution space employing different pre-processing techniques such as slicing, incremental upper bounds and unrolling transitivity constraints. The evaluation shows that our tool is practical and beneficial in industrial settings to specify trace semantics for automated trace reasoning. We plan to conduct more case studies to better evaluate the practical utility and usability of the platform.

[1]A parse for a sentence S deriving a DL formulae ϕ for S is likely to be correct if at least one of the sentences regenerated from ϕ is highly similar to the input sentence S.

TABLE II
Comparisons of Several Use Cases for Trace Inferring

	Artifacts	Traces	Inferred	Alloy	KodKod	Z3
#1	123	102	89	67922	25668	40900
#2	56	27	25	4428	84	480
#3	42	103	75	724	1	1460

V. Related Work

Many existing works on semantic parsing describe the task of obtaining axiomatic representation of natural langauge sentences. However, they suffer from two main limitations: (i) use of controlled languages such as Attempto Controlled English [19] (e.g. [20], [21]) and/or (ii) inability to deduce complex axioms involving logical connectives, role restrictions and other expressive features of OWL (e.g. [22], [23]), as noted in [24]. In contrast, we work on human authored real-world text (Airbus SIDPs) and produce complex OWL axioms involving the following DL constructs: $\top$ (the most general concept), disjunction, conjunction, negation, role inverse, universal and existential restrictions. Moreover, we extended the scope of our application by deducing traces among the semantic parse outputs. Such traces were then used as baseline input to Tarski platform which could infer additional traces propagating over the whole system.

Similarly, several approaches and tools have been proposed for automated trace reasoning using the trace semantics [4], [25]–[31]. These approaches employ a predefined set of trace types and their corresponding semantics. For instance, Goknil et al. [4] provide a tool for inferencing and consistency checking of traces between requirements using a set of trace types and their formal semantics. Similarly, Egyed and Grünbacher [26] propose a trace generation approach. They do not allow the user to introduce new trace types and their semantics for automated reasoning. In the development of complex systems, it is required to enable the adoption of various trace types, and herewith automated reasoning using their semantics. Tarski module of ModelWriter allows the user to interactively define new trace types with their semantics to be used in automated reasoning about traces.

VI. Conclusion

We presented an integrated platform for automatically mapping natural language text to trace types and performing further inference on those traces. Starting with the semantic parser module, we showed how complex axioms could be derived to represent text coming from real world use cases. We identified the traces among the parse outputs and fed it to the Tarski tool. The Tarski tool, in turn, allowed users to specify configurable trace semantics for various forms of automated trace reasoning such as inference and consistency checking. The key characteristics of our tool are (1) automatic identification of traces existing in text using semantic parsing (2) allowing user to define new trace types and their semantics which can be later configured, (3) deducing new traces based on the traces which the user has already specified, and (4) identifying traces whose existence causes a contradiction.

Acknowledgment

This work is conducted within ModelWriter project [32] and partially supported by the Scientific and Technological Research Council of Turkey under project #9140014, #9150181 and Industry and Digital Affairs of France under contract #142930204. The authors would like to acknowledge networking support by European Cooperation in Science and Technology Action IC1404 "Multi-Paradigm Modelling for Cyber-Physical Systems".

References

[1] RTCA and EUROCAE, "DO-178C: Software considerations in airborne systems and equipment certification," 2017.

[2] I. C. Society, P. Bourque, and R. E. Fairley, *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*, 3rd ed. Los Alamitos, CA, USA: IEEE Computer Society Press, 2014.

[3] S. Maro, A. Anjorin, R. Wohlrab, and J. P. Steghöfer, "Traceability maintenance: Factors and guidelines," in *31st IEEE/ACM International Conference on Automated Software Engineering*, 2016, pp. 414–425.

[4] A. Goknil, I. Kurtev, K. van den Berg, and J.-W. Veldhuis, "Semantics of trace relations in requirements models for consistency checking and inferencing," *Software and System Modeling*, vol. 10, no. 1, pp. 31–54, 2011.

[5] R. Shearer, B. Motik, and I. Horrocks, "Hermit: A highly-efficient owl reasoner." in *5th OWL Experienced and Directions Workshop*, vol. 432, 2008, p. 91.

[6] D. Jackson, *Software Abstractions: Logic, Language, and Analysis*. MIT press, 2012.

[7] E. Torlak, "A constraint solver for software engineering: Finding models and cores of large relational specifications," Ph.D. dissertation, Massachusetts Institute of Technology, 2008.

[8] B. Gyawali, A. Shimorina, C. Gardent, S. Cruz-Lara, and M. Mahfoudh, "Mapping natural language to description logic," in *The Semantic Web: 14th International Conference, ESWC 2017*, 2017, pp. 273–288.

[9] E. Bozsak, M. Ehrig, S. Handschuh, A. Hotho, A. Maedche, B. Motik, D. Oberle, C. Schmitz, S. Staab, L. Stojanovic *et al.*, "KAON – towards a large scale Semantic Web," in *E-Commerce and Web Technologies*. Springer, 2002, pp. 304–313.

[10] D. L. McGuinness, F. Van Harmelen *et al.*, "OWL web ontology language overview," *W3C recommendation*, vol. 10, no. 10, p. 2004, 2004.

[11] C. Gardent and L. Kallmeyer, "Semantic construction in feature-based TAG," in *Proceedings of EACL*. Association for Computational Linguistics, 2003, pp. 123–130.

[12] F. Erata, M. Challenger, B. Tekinerdogan, A. Monceaux, E. Tüzün, and G. Kardas, "Tarski: A platform for automated analysis of dynamically configurable traceability semantics," in *the 32nd ACM SIGAPP Symposium on Applied Computing (SAC'17)*, 2017, pp. 1607–1614.

[13] F. Erata, A. Goknil, B. Tekinerdogan, and G. Kardas, "A tool for automated reasoning about traces based on configurable formal semantics," in *11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2017)*, 2017, pp. 959–963.

[14] E. Torlak and D. Jackson, "Kodkod: A relational model finder," in *the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07)*, 2007, pp. 632–647.

[15] L. D. Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08)*, 2008, pp. 337–340.

[16] N. Eén and N. Sörensson, "An extensible sat-solver," in *the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, 2003, pp. 502–518.

[17] A. A. E. Ghazi and M. Taghdiri, "Relational reasoning via smt solving," in *the 17th International Conference on Formal Methods (FM'11)*, 2011, pp. 133–148.

[18] C. Barrett, A. Stump, C. Tinelli *et al.*, "The smt-lib standard: Version 2.0," in *the 8th International Workshop on Satisfiability Modulo Theories*, vol. 13, 2010, p. 14.

[19] K. Kaljurand and N. E. Fuchs, "Verbalizing OWL in Attempto Controlled English," in *OWLED*, vol. 258, 2007.

[20] V. Tablan, T. Polajnar, H. Cunningham, and K. Bontcheva, "User-friendly ontology authoring using a controlled language," in *Proceedings of LREC*, 2006.

[21] A. Bernstein, E. Kaufmann, A. Göhring, and C. Kiefer, "Querying ontologies: A controlled english interface for end-users," in *International Semantic Web Conference*. Springer, 2005, pp. 112–126.

[22] P. Buitelaar, P. Cimiano, and B. Magnini, *Ontology learning from text: methods, evaluation and applications*. IOS press, 2005, vol. 123.

[23] M. Ruiz-Casado, E. Alfonseca, and P. Castells, "Automatic Extraction of Semantic Relationships for Wordnet by Means of Pattern Learning from Wikipedia," in *International Conference on Application of Natural Language to Information Systems*. Springer, 2005, pp. 67–79.

[24] J. Völker, P. Hitzler, and P. Cimiano, "Acquisition of OWL DL axioms from lexical resources," in *European Semantic Web Conference*. Springer, 2007, pp. 670–685.

[25] N. Aizenbud-Reshef, R. F. Paige, J. Rubin, Y. Shaham-Gafni, and D. S. Kolovos, "Operational semantics for traceability," in *ECMDA Traceability Workshop (ECMDA-TW'05)*, 2005, pp. 8–14.

[26] A. Egyed and P. Grünbacher, "Supporting software understanding with automated requirements traceability," *International Journal of Software Engineering and Knowledge Engineering*, vol. 15, no. 5, pp. 783–810, 2005.

[27] A. Egyed, "A scenario-driven approach to trace dependency analysis," *IEEE Transactions on Software Engineering*, vol. 29, no. 2, pp. 116–132, 2003.

[28] J. Cleland-Huang, C. K. Chang, and M. J. Christensen, "Event-based traceability for managing evolutionary change," *IEEE Transactions on Software Engineering*, vol. 29, no. 9, pp. 796–810, 2003.

[29] L. C. Lamb, W. Jirapanthong, and A. Zisman, "Formalizing traceability relations for product lines," in *the 6th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE'11)*, 2011, pp. 42–45.

[30] A. Goknil, I. Kurtev, and K. V. D. Berg, "Generation and validation of traces between requirements and architecture based on formal trace semantics," *Journal of Systems and Software*, vol. 88, pp. 112–137, 2014.

[31] N. Drivalos, D. S. Kolovos, R. F. Paige, and K. J. Fernandes, "Engineering a dsl for software traceability," in *1st International Conference on Software Language Engineering (SLE'08)*, 2008, pp. 151–167.

[32] F. Erata, "ModelWriter: Text & model synchronized document engineering platform," https://itea3.org/project/modelwriter.html, Sep 2014.