Assisted Authoring of Model-Based Systems Engineering Documents

Thomas Boyer Chammard, Blake Regalia, Robert Karban, Ivan Gomes
{thomas.boyer.chammard,blake.d.regalia,robert.karban,ivan.gomes}@jpl.nasa.gov
Jet Propulsion Laboratory, California Institute of Technology
Pasadena, California

ABSTRACT

In systems engineering practices, system design and analysis have historically been performed using a document-centric approach where stakeholders produce a number of documents that represent their views on a system under development. Given the ad-hoc, disparate, and informal nature of natural language documents, these views become quickly inconsistent. Rigor in engineering work is also lost in the transition from model-based engineering design and analysis to engineering documents. Once the documents are delivered, the engineering portion of the work is disconnected. In the Open Model Based Engineering Environment (OpenMBEE), Cross-References (aka transclusions) synthesize relevant engineering information where model elements are not simply hyperlinked, but de-referenced in place in a document, upgrading a document-based process with model-based engineering technology. Those Cross-References are nowadays partially created manually, putting a burden on the engineer who is authoring the document. This paper presents an approach which can assist the engineer by providing machine-generated suggestions for Cross-References using language processing, graph analysis, and clustering technologies on model data managed by the OpenMBEE infrastructure.

CCS CONCEPTS

• **Software and its engineering** → **Unified Modeling Language (UML)**; **System modeling languages**; **Model-driven software engineering**; • **Information systems** → *Graph-based database models*; *Resource Description Framework (RDF).*

KEYWORDS

Graph analysis, clustering, entity linking, natural language processing, OpenMBEE, MBSE, SysML

1 INTRODUCTION AND MOTIVATION

In systems engineering practices, system design and analysis have historically been performed using a document-centric approach where stakeholders produce a number of documents that represent their respective views on a system under development. Given the ad-hoc, disparate, and informal nature of natural language documents, these views become quickly inconsistent. Currently, dependencies between these views are often implicit or informally defined in supplementary documents. Additionally, documents are often accompanied by a variety of discipline-specific engineering models that are siloed. It is often difficult to trace provenance across the different, distributed sources of information and verify their consistency. Moreover, the rigor of engineering work is lost in the transition from model-based engineering design and analysis to engineering documents. Once the documents are delivered, the engineering portion of the work is disconnected from the resulting artifacts. OpenMBEE is an integrated set of software applications and services which replaces silos of information with consistent, traceable, and precise engineering models and documents which is a key element of Model-Based Systems Engineering (MBSE). MBSE is the formalized application of modeling to support system requirements, design, analysis, verification, validation, and documentation activities, spanning the entire lifecycle of complex systems engineering projects. In OpenMBEE, Cross-References synthesize relevant engineering information where model elements are not simply hyperlinked, but de-referenced in place (i.e. transcluded), upgrading a document-based process with a model-based engineering technology. Using Cross-References, OpenMBEE upgrades a document-based process with a model-based engineering environment. They become the rendering of model information inline with unstructured narrative in model-based documents. Figure 1 shows an example of the Thirty Meter Telescope[1] production model in the OpenMBEE ViewEditor which is a web-based, document oriented editor implementing the Cross-Reference capability.

Therefore, the link to the authoritative source of information is preserved and maintained. The use of Cross-References also provides a new means of measuring the maturity of the information overall referred to as model hardness. Early lifecycle documents utilize a relatively low number of Cross-References. By the end of the life cycle, the interconnection and use of models and structured data should be rich, which is evidenced by a large number of Cross-References. Unstructured data indicates sections of the engineering design which require more development, and engineers can incrementally add formalism to their models as their thinking matures from concept to design. Figure 2 illustrates this concept.

Those Cross-References are nowadays either generated by DocGen [5, 10], which provides a structured representation of documents while still allowing incorporation of unstructured data into structured engineering data, constructing linked-data documents. Or, they are created manually using OpenMBEE View Editor. The latter puts a burden on the engineer authoring the document because either existing model elements have to be searched for or inserted as Cross-Reference into the narrative or elements of the

MODELS '20 Companion, October 18–23, 2020, Virtual Event, Canada
© 2020 California Institute of Technology. Government sponsorship acknowledged.
ACM ISBN 978-1-4503-8135-2/20/10.
https://doi.org/10.1145/3417990.3421406

[1]TMT SysML Model: https://github.com/Open-MBEE/TMT-SysML-Model

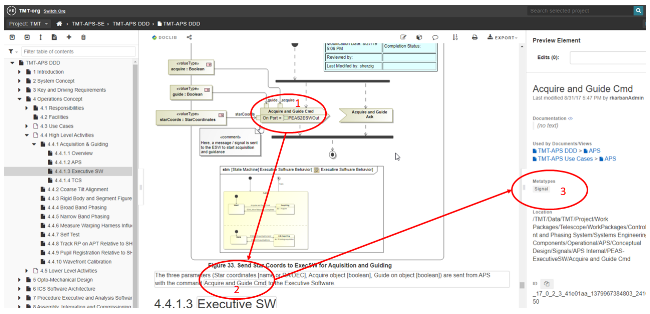
TMT-org Switch Org
Search selected project
Project: TMT
TMT-APS-SE › TMT-APS DDD › TMT-APS DDD
DOCLIB
EXPORT
Filter table of contents
TMT-APS DDD
1 Introduction
2 System Concept
3 Key and Driving Requirements
4 Operations Concept
4.1 Responsibilities
4.2 Facilities
4.3 Use Cases
4.4 High Level Activities
4.4.1 Acquisition & Guiding
4.4.1.1 Overview
4.4.1.2 APS
4.4.1.3 Executive SW
4.4.1.4 TCS
4.4.2 Coarse Tilt Alignment
4.4.3 Rigid Body and Segment Figure
4.4.4 Broad Band Phasing
4.4.5 Narrow Band Phasing
4.4.6 Measure Warping Harness Influ
4.4.7 Self Test
4.4.8 Track RP on APT Relative to SH
4.4.9 Pupil Registration Relative to SH
4.4.10 Wavefront Calibration
4.5 Lower Level Activities
5 Opto-Mechanical Design
6 ICS Software Architecture
7 Procedure Executive and Analysis Softwar
8 Assembly, Integration and Commissioning
«valueType» acquire : Boolean
«valueType» guide : Boolean
«valueType» starCoords : StarCoordinates
starCoords
guide
acquire
Acquire and Guide Cmd
On Port = PEAS2ESWOut
Acquire and Guide Ack
«comment» Here, a message / signal is sent to the ESW to start acquisition and guidance
5:06 PM
Completion Status:
Reviewed by:
Last Modifed by: sherzig
stm [State Machine] Executive Software Behavior[Executive Software Behavior]
1
2
3
Preview Element
Edits (0):
Acquire and Guide Cmd
Last modified 8/31/17 5:47 PM by rkarbanAdmin
Documentation
(no text)
Used by Documents/Views
TMT-APS DDD > APS
TMT-APS Use Cases > APS
Metatypes
Signal
Location
ID

Figure 33. Send Star Coords to ExecSW for Aquisition and Guiding

The three parameters (Star coordinates [name or RA/DEC], Acquire object [boolean], Guide on object [boolean]) are sent from APS with the command Acquire and Guide Cmd to the Executive Software.

4.4.1.3 Executive SW

/TMT/Data/TMT/Project/Work Packages/Telescope/WorkPackages/Control and Phasing System/Systems Engineering Components/Operational/APS/Conceptual Design/Signals/APS Internal/PEAS-ExecutiveSW/Acquire and Guide Cmd

_17_0_2_3_41e01aa_1379967384803_24150

Figure 1: A document from the TMT model rendered in the ViewEditor web interface. Annotations shown in red:

1. **(1) Model element, signal "Acquire and Guide Cmd", in the context of the rendered diagram**
2. **(2) Cross-Reference of the signal model element name in-line with narrative**
3. **(3) Specification of the signal**

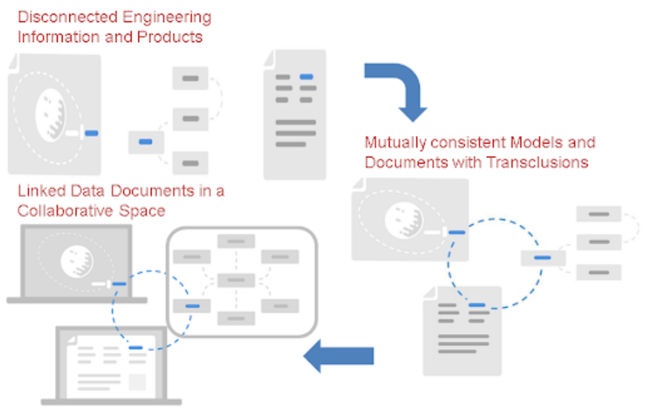
Disconnected Engineering Information and Products
Mutually consistent Models and Documents with Transclusions
Linked Data Documents in a Collaborative Space

Figure 2: A piece of engineering information, shown in blue, exists in models, processes, and documents and is initially disconnected. In the next step, Cross-References are used to create linked-data documents that close the gap between model and documentation, ensuring that they are mutually consistent and correspondent. In the third step, those documents are made available in a collaborative space where engineers author, review, modify, and release those documents.

document have to be turned into a model element and then Cross-Referenced. Of course there are many model elements which are not referenced and many text fragments which could potentially be model elements.

In this paper, we describe our approach to assist the engineer with creating Cross-References by automating the discovery of likely Cross-Reference targets and proposing them for review. Rather than taking a fully automated approach to creating transclusions, we start with a Human-in-the-loop approach which enables the system to collect training data for future machine-learning tasks as well as usage statistics for latent tool evaluation. The effort focuses on mitigating the trade-off between the time spent creating Cross-References and the consistency of the model that engineers are currently facing, thus enhancing the process of transitioning from a document-based to a Model-Based Systems Engineering.

The approach will be presented with a small illustrative example, and was developed and tested on the TMT open-source SysML model.

2 RELATED WORK

Creating structured references to named entities out of unstructured, human-readable text documents is a well-studied topic in Natural Language Processing (NLP). Information Extraction and Named Entity Recognition and Disambiguation (NERD) more specifically, are tasks within this domain that identify which words and phrases within a given text fragment or document are references. Entity Linking is an extension of this task that also determines which entity (if any) each of those references should link to within a given knowledge base. In this work, we apply these technologies in a Model-Based Systems Engineering context by treating SysML model data as a knowledge graph.

A recent study by Shen et. al [14] thoroughly describes the techniques associated with Entity Linking, covering a broad spectrum of methodologies, from graph-based approaches to unsupervised Vector Space Model ranking methods. Hoffart et al. [9] propose a method that builds a subgraph of entities mentioned in the text by using an algorithm that approximates the best joint mention-entity mappings to disambiguate. Additionally, Hakimov et. al [7] present an approach that uses a graph-based Centrality Scoring for disambiguation. The work presented in this paper is similar in concept to these approaches. Although, we will not use an open knowledge base of entities (such as DBpedia or Wikidata), but rather the graph of entities that constitute our Systems Engineering model.

Kalashnikov et al. use context clustering [2], and rely on relationships' connection strength as a means to perform disambiguation. For that matter, they formulated the Context Attraction Principle (CAP) which we use the premise to our approach for Entity Linking.

Work by Ferragina and Scaiella [6] address the unique challenges of annotating short texts containing cross-references by demonstrating the use of their original research tool, TAGME, to perform entity linking against Wikipedia articles. However, TAGME does not generalize well to our use case as it relies on a rich and broad knowledge graph to build context for each document it analyzes. Our approach attempts to identify the names of engineering requirements, systems, actions, and concepts scoped to a domain-specific SysML model.

In Requirements Engineering, the analysis of user reviews is a rich and growing source of information [3]. In [4], Dalpiaz et al. propose RE-SWOT, a tool eliciting requirements from App Store reviews through competitor analysis. This tool extracts, from customer reviews, keywords that correspond to features of applications. The RE-SWOT Natural Language Processing (NLP) module is very similar to the one proposed in this paper, the goal being the extraction of patterns of Parts-Of-Speech (POS) tags, with the merging of feature labels based on a string similarity score. The authors do not, however, proceed to any sort of disambiguation or linking to a dictionary of existing features, as their goal is purely analytic.

3 METHODS

In order to assist the engineer with discovering transclusions in the text documents of a given SysML model, we present a novel technique that performs entity linking without pretrained features. Instead, our approach relies on the Context Attraction Principle to form an implicit context about an input text via the topology of a knowledge graph (in this case the SysML model's instance data). This technique provides some basic context during the disambiguation phase of entity linking in the absence of more complex models. As demonstrated by Kalashnikov et al. [11], this context is based on the connection strength between two elements, which is approximated in our approach by computing the length of the shortest path between them.

We provide a simple example SysML model in Figure 4 to illustrate the process of how references to model elements are discovered are linked within a document text.

In this initial experiment, we limit the domain to structural SysML elements only (e.g. Blocks and Properties) in order to scope our evaluation to a manageable subset of the model. Other element types, such as behavioral elements, activities, and actions, are reserved for future work.

3.1 Querying SysML Models

The data model for UML, and by extension SysML, are based on the entity-attribute-relationship model, or the entity-relationship (ER) model more broadly, from classical conceptual modeling and traditional database design [1]. In their metamodels, relationships are defined to exist between entities of certain types. When these conceptual models are mapped to a logical schema, it is typically in the form of a relational model or one of its derivatives. For instance, OpenMBEE's Model Management System (MMS) uses a key-value database to store SysML model elements. This makes for very efficient element access by certain properties, but lends itself poorly to graph pattern matching which may involve many joins given arbitrary-length paths. Graph models provide inherent benefits for many natural language processing (NLP) tasks where the objective is to create structured data out of unstructured, human-readable text, such as named-entity recognition and disambiguation (NERD). As we discuss in further detail below, we adapt MMS to a graph data model by forming a knowledge graph out of SysML model elements. Additionally, our approach materializes and appends the results of its entity linking task into the same knowledge graph. This knowledge graph can in turn be used to improve entity disambiguation, i.e., by forming a context, or "user intent" out of a given document's text.

RDF is a graph-based data model, language, and technology for creating, storing, and reasoning on knowledge graphs. SPARQL is the query language for RDF. MMS-RDF[2] is a SysML to RDF conversion tool. It first generates an OWL ontology directly from the UML metamodel, then appends SysML extensions to that ontology, and finally adds MMS-specific vocabulary definitions, such as properties for describing commit metadata. MMS-RDF then utilizes this generated ontology to perform an Extract-Transform-Load (ETL) on a SysML model by pulling data from MMS, transforming each model element into a set of RDF triples, and loading the resulting graph into a triplestore where a SPARQL endpoint is exposed for querying. The overall process is illustrated in Figure 3. In order to facilitate shortest path queries using a graph-traversal query language such as Gremlin, MMS-RDF also creates a Labeled Property Graph (LPG) view of the instance data graph, which is exposed via Gremlin endpoint. The RDF graph includes both ABox and TBox statements, meaning that in addition to instance data, it also covers class hierarchies, multiplicities, and domain and range restrictions. On the other hand, the simpler LPG view strictly covers instance data in order to complement RDF query capabilities, such as graph-traversal queries for shortest paths between two nodes.

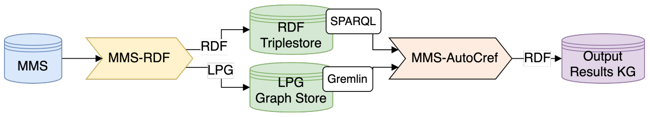

Figure 3: Starting with the SysML model in MMS, MMS-RDF generates a complete representation of the model as RDF, and an instance-data only view of the model as a LPG, then pushes these to their respective graph query services where they are exposed to our application (MMS-AutoCref) through SPARQL and Gremlin endpoints. Finally, MMS-AutoCref produces output results as RDF triples and inserts them into a user-specified knowledge graph (KG).

3.2 Entity Linking

In NLP, entity linking (also referred to as entity disambiguation) is used to predict which entity a given phrase should identify in a knowledge base. Techniques for entity linking systems can generally be broken down into one of two categories, text-based and graph-based. In text-based entity linking, entities are ranked based on features trained by a corpus of documents. In graph-based entity linking, the topology of a knowledge graph containing many linked documents is used to generate features which attempt to capture the context of entity relationships. It is important to note that these two approaches can be used in tandem, although they occur at different stages of an NLP pipeline.

In our experiment, we implement a simple entity linking system that performs three distinct steps: (1) Parsing, (2) Selection, and (3) Ranking. We describe each of these steps below.

Step 1: Parsing. *Given a human-readable document in HTML or plain text, strip away markup and parse the remaining text using an NLP pipeline in order to extract all noun phrases.*

Our implementation uses spaCy[3], an open-source NLP library with a python API, and a pretrained model for the English language trained by written texts from the web including blogs, news and comments[4]. The NLP pipeline we set up for this experiment consists of three stages: (1) Tokenization, which uses a pretrained English model to turn a series of characters into a structured series

[2]MMS-RDF: https://github.com/Open-MBEE/mms-rdf

[3]spaCy: https://spacy.io/

[4]spaCy English model: https://spacy.io/models/en

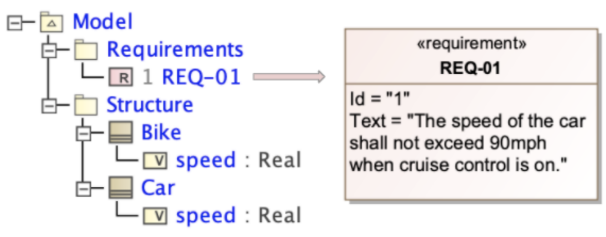

Figure 4: A SysML visualization of the simple illustrative example model for this paper. In the single requirement text, two transclusions to model elements can be identified.

of words/phrases; (2) Part-of-Speech (POS) tagging, which assigns each token a part-of-speech label such as "noun", "verb", "adjective", and so on; and (3) noun-chunk merging, where the pretrained English model forms noun-phrases by merging adjacent tokens that likely refer to an entity, e.g., "The Golden Gate Bridge". Our objective is to find phrases in the input text that refer to named elements in the SysML model. The string matching process is described in Step 2.

The output string of this process is intended to replace the original document, so any markup that was stripped away for parsing must be restored. It's important to note that this approach does not preclude one from utilizing the semi-structured data found in the original HTML document. For example, anchored texts in the input documents, such as manually created cross-references to other model elements, can be used to refine the context before entity linking takes place. However, we see the use of semi-structured input data as an added feature for future work, and instead focus here on the fundamental component of working with unstructured input text.

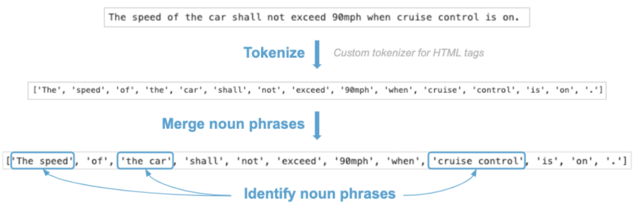
The speed of the car shall not exceed 90mph when cruise control is on.
Tokenize
Custom tokenizer for HTML tags
['The', 'speed', 'of', 'the', 'car', 'shall', 'not', 'exceed', '90mph', 'when', 'cruise', 'control', 'is', 'on', '.']
Merge noun phrases
['The speed', 'of', 'the car', 'shall', 'not', 'exceed', '90mph', 'when', 'cruise control', 'is', 'on', '.']
Identify noun phrases

Figure 5: The text is first tokenized, then tokens are tagged with PoS labels, and finally noun-phrases are formed by merging adjacent tokens where appropriate. The noun-phrases are now considered extracted and fed into Step 2.

Step 2: Selection. *For each noun-phrase extracted from the input text, select all named elements from the model that fuzzy match the given noun-phrase's token string.*

Each of the noun-phrases that was extracted in Step 1 is used to search for candidate references targets by fuzzy string matching against the names of model elements. Each comparison is based on the string similarity between the noun-phrase (string A) and the name of the model element (string B). For this experiment, we employ certain common stopwords [15] for noun-phrases, such as ignoring the word "the" before comparison, to improve fuzzy string matching against named model elements.

Slight variations in the text of named elements may occur in the SysML model between the entity and its references due to a number of reasons including typos, plurality, possessivity, and differences in spelling caused by international collaboration (e.g., British and American spellings of certain English words). Stemming and lemmatization, in which variations of a word are dealt with by reducing them to a common, unique representation, are commonly used techniques to overcome some of these issues. However, variations due to typos and punctuation are not directly handled by these techniques alone and systems typically end up using some form of fuzzy string matching near the end of a pipeline to catch outliers. Levenshtein distance [8] is a metric to measure the distance between two strings by counting the number of insertions, substitutions, or deletions needed to transform string A into string B. In lieu of more complicated techniques that require trained featuresets, Levenshtein distance can provide an approximation for semantic similarity between two entity names with a few modifications. We use a modified Levenshtein distance algorithm implemented by the python library fuzzywuzzy[5], which first converts both strings to lowercase, removes non-alphanumeric characters, computes their Levenshtein distance, and then scores their similarity on a scale from 0 - 100. Our primary reason for selecting this algorithm is due to how it better compensates for shorter versus longer strings than naive Levenshtein distance (for example, "cat" and "car" score 67 similarity while "principle" and "principal" score 89). A minimum score of 90 was selected as the lower threshold value for determining if a given comparison should be considered as a fuzzy match. This threshold was chosen conservatively after manually labeling scores in trials with the TMT model. Future work will leverage a supervised learning model to assist with score classification, and consider techniques better suited for measuring semantic similarity. Finally, since a single noun-phrase may match multiple element targets, disambiguation is handled in Step 3.

Step 3: Ranking. *Determine which model element out of each reference's choice set should be linked to based on the local SysML model context of the input document.*

At this step, we have a set of candidate model elements, i.e., a 'choice set', (all with high string similarity scores) for each reference in the input document. Since some of the choices may have the exact same name, we must use a technique that allows us to capture the local context of the input document in order to give preference to certain elements. Kalashnikov et al. offer a solution here in the form of the Context Attraction Principle (CAP) for graph-based reference disambiguation [11, 12], which states:

> *If reference r made in the context of entity x refer to an entity y_j, whereas the description provided by r matches multiple entities $y_1, \ldots, y_j, \ldots, y_N$, then x and y_j are likely to be more strongly connected to each other via chains of relationships than x and y_l $(l = 1, 2, \ldots, N; l)$.*

The CAP loosely encodes the notion that the distance between two nodes in a knowledge graph is inversely proportional to the

[5]fuzzywuzzy: https://github.com/seatgeek/fuzzywuzzy

relatedness of the concepts they represent. In other words, the closer two nodes appear in the graph, the more likely their concepts are to be more strongly connected to each other than to concepts belonging to nodes further away.

To give an illustration of this technique, recall the running example from Figure 4. Three model elements are selected in Step 2 from the requirement text: `Car`, `Car::speed` and `Bike::speed`. In the model, the speed attribute of the Car is 1 hop away from `Car`, whereas the speed attribute of the Bike is 3 hops away from `Car`. Therefore, the connection strength between `Car` and its owned `Car::speed` property is greater than the one between `Car` and the `Bike::speed` property. Following the CAP, because the entity `Car` appears in the input document, linking the phrase "speed" results in giving preference to `Car::speed` over `Bike::speed`.

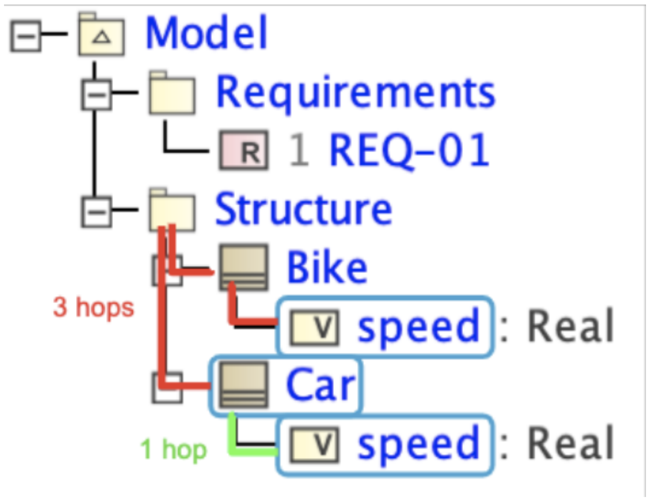

Figure 6: Annotations on the illustrative example showing relative connection strengths between the candidate elements selected from the input text. `Car::speed` is closer to `Car` than `Bike::speed`, hence a higher connection strength and ultimately a higher ranking.

To assist with disambiguation, matches to other elements within the document provide an implicit context via their connections in the knowledge graph. To create and apply this context, we perform agglomerative hierarchical clustering [13] on an induced subgraph in which the shortest paths between all pairwise combinations of nodes in the choice set are replaced by edge weights corresponding to the lengths of those paths. In other words, the shortest path between two nodes in the SysML knowledge graph turns into a single edge in the derivative graph, whose weight is determined by the number of hops in the original path. This edge weight serves as the connection strength [11] for Kalashnikov's CAP.

When the agglomerative hierarchical clustering algorithm is run on the modified subgraph, the order in which the nodes entered a cluster, or how 'high' they appear on the hierarchical dendrogram, determines their ranking in the implicit context. The earlier a node entered any cluster (i.e. the higher the proximity to other matches), the higher the ranking and thus the more strongly related that node's concept is to the context.

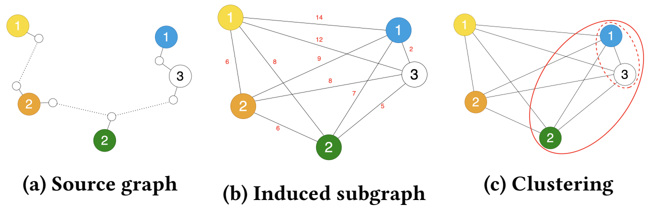

Figure 7: Overall process to rank candidates for disambiguation. In the above diagrams, candidate entities for linking are shown as labeled nodes. The label corresponds to which reference token it is competing for in the text, e.g., the label "1" indicates that a node is competing for the first noun-phrase, such as "speed". "2" would therefore be another noun-phrase that multiple model elements could match, such as "cruise control". Finally, "3" appears once, meaning that only a single model element matched its corresponding noun-phrase, such as "car". Subfigures: (a) A selection of the candidate entities from the source knowledge graph. (b) An induced subgraph is created and paths are replaced by weighted edges representing the length of the shortest path between each pair of nodes. (c) Agglomerative hierarchical clustering takes place. Nodes 1-blue and 2-green entered the cluster before 1-yellow and 2-orange, and are therefore ranked higher.

Figure 7 illustrates the process. The nodes represent model elements. The colors on the nodes are mere visual aids. The ID on each node corresponds to the noun-chunk that the model element has been matched to. The noun-chunks 1 and 2 have each been matched to 2 different model elements, and the noun-chunk 3 has been matched to only one model element.

Following the hypothesis, we can now rank which model element is the more relevant for noun-chunk 1 and 2. For noun-chunk 1, the blue node entered a cluster before the yellow node. Therefore, the blue node is ranked higher (meaning more relevant) for noun-chunk 1 than the yellow node. For noun-chunk 2, the green node entered a cluster before the orange node so the green node is ranked more relevant.

3.3 Preliminary Experimental Output

The tests were conducted with the open-source TMT model[6]. We manually annotated 175 requirements and compared the output against our entity linking approach. Of the 175 analyzed requirements, 89 made references to model elements for a total of 125 transclusions. Out of these 125 transclusions, the algorithm correctly identified 84 (67%). Of the 41 transclusions that were not identified, 30 were due to an unsuccessful comparison of the strings, most of the time due to acronymization in the text (e.g. "M1" instead of "Primary Mirror", "DMS" instead of "Data Management System"). The remainder of the unsuccessful transclusions were due to incorrect parsing (e.g. in a requirement like "The APT system shall [...]",

[6]https://github.com/Open-MBEE/TMT-SysML-Model

the identified noun-phrase is "The APT system" which is too different from the corresponding "APT" model element to be correctly identified).

4 DISCUSSION AND LIMITATIONS

4.1 Edge Cases

The following edge cases were identified along with possible solutions to them.

The input text contains only one noun-chunk token which matches multiple model elements. In this case, disambiguation is impossible since there are no other elements to form an implicit context. To address this case, we suggest adding the node that corresponds to the input document (in this case, the SysML Requirement model element) to the subgraph before clustering takes place. In this scenario, the input document node provides a default context within the model for disambiguation. While there are several considerations for how this affects the context (e.g., the package locality of related concepts in the SysML model), future work is needed to evaluate the alternatives.

All candidate model elements are equidistant in the induced subgraph. This can be especially difficult when enumerations of instances have similar names (e.g. if instead of Car and Bike, we had Car1 and Car2 in our illustrative example). However, in this case, it becomes difficult to surmise a ground truth for disambiguation, even for a human. As a first step, we designed our approach to take a modest 'backseat' to the authoring engineers by *assisting* them with choosing which occurrences should be transcluded.

4.2 Shortest path queries and connection strength

The shortest path queries are computed using Gremlin on the LPG view of the SysML model instance data. This is the only use we are making of the LPG and is therefore the only reason we are required to use both RDF and LPG. The queries for shortest paths could also be done using the RDF graph and SPARQL. However, this would impact our approach as explained hereafter. Transitioning from a Gremlin to a SPARQL shortest path query would require the use of multiple queries to probe the shortest path one hop length at a time. Furthermore, querying the RDF graph would also require limiting the predicates to instance data properties only, a quality that is designed into the creation of the LPG view.

On a similar note, the connection strength used for disambiguation does not need to be solely determined by the length of the path. Currently, we are measuring the connection strength as the length of the shortest path, which is one of the many considered ways described by Kalashnikov et al. [17]. But since the relationships in the model have different semantics, it follows that they should weigh differently on the various connection strengths. For example, relationships that carry a strong meaning towards the system at a conceptual level might be prioritized over relationships that are only inherent to the modeling methodology (e.g. containment). Future work needs to assess whether such relative weights should be engineered in top-down manner (i.e., decided by model designers), learned in a bottom-up approach (i.e., training models using corpora of labeled data), or derived in a hybrid approach (i.e., human-in-the-loop machine learning).

4.3 Compound References

In MMS, individual references can be combined into compound references to convey a more significant meaning. For example, a single compound reference could consist of a value property, its value and its unit, instead of three separate references. Presenting the user with such compound references is not yet a feature in the current state of our prototype, but the way the ranking step uses clustering for disambiguation lends itself to this type of extension.

5 CONCLUSIONS AND FUTURE WORK

The presented approach allows for an assisted process of creating transclusions by detecting them automatically within the human-readable text of engineering documents, and then presenting them to the engineer for review. Our approach also ensures that results can be ranked and presented to the user by relevance, allowing for other applications such as auto-completion and path refactoring.

In the future, we plan to enhance our process by including behavioral elements (e.g. activities, actions) in the pipeline, to cover the entire model. We also plan to improve the identification of model elements in the text (Step 1) by training a Named Entity Recognition [21] machine learning model, commonly used for similar tasks in Natural Language Processing (NLP). This NER model would help further optimize the process of ranking, by disambiguating the type of model element that the system is looking for on a specific text chunk. It would also provide a human-in-the-loop for labeling data in real-time, improving the entity disambiguation (machine learning) model as the SysML model grows and the user accepts or rejects transclusions.

Additional directions for future include the creation of an IDE-like auto-complete feature in View Editor, using the context of the text as it is being typed to recommend elements of the corresponding context in the model. One could also detect textual elements that indicate importance to the narrative via information value theorem and summarization, or suggest the creation of new model elements based on extracted phrases. We also see the potential for creating tranclusions to quantitative values and constraints. For example "speed shall not exceed 90 mph" could translate into a SysML constraint "speed<90" and an accompanying value property.

ACKNOWLEDGMENTS

This research was carried out at the Jet Propulsion Laboratory (JPL), California Institute of Technology, under a contract with the National Aeronautics and Space Administration (NASA).

REFERENCES

[1] Peter Pin-Shan Chen. 1976. The entity-relationship model—toward a unified view of data. *ACM transactions on database systems (TODS)* 1, 1 (1976), 9–36.
[2] Zhaoqi Chen, Dmitri V Kalashnikov, and Sharad Mehrotra. 2009. Exploiting context analysis for combining multiple entity resolution systems. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. 207–218.
[3] Jacek Dąbrowski, Emmanuel Letier, Anna Perini, and Angelo Susi. 2019. Finding and analyzing app reviews related to specific features: A research preview. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 183–189.
[4] Fabiano Dalpiaz and Micaela Parente. 2019. RE-SWOT: from user feedback to requirements via competitor analysis. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 55–70.

[5] Christopher Delp, Doris Lam, Elyse Fosse, and Cin-Young Lee. 2013. Model based document and report generation for systems engineering. In *2013 IEEE Aerospace Conference*. IEEE, 1–11.

[6] Paolo Ferragina and Ugo Scaiella. 2011. Fast and accurate annotation of short texts with wikipedia pages. *IEEE software* 29, 1 (2011), 70–75.

[7] Sherzod Hakimov, Salih Atilay Oto, and Erdogan Dogdu. 2012. Named entity recognition and disambiguation using linked data and graph-based centrality scoring. In *Proceedings of the 4th international workshop on semantic web information management*. 1–7.

[8] Patrick AV Hall and Geoff R Dowling. 1980. Approximate string matching. *ACM computing surveys (CSUR)* 12, 4 (1980), 381–402.

[9] Johannes Hoffart, Mohamed Amir Yosef, Ilaria Bordino, Hagen Fürstenau, Manfred Pinkal, Marc Spaniol, Bilyana Taneva, Stefan Thater, and Gerhard Weikum. 2011. Robust disambiguation of named entities in text. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. 782–792.

[10] Maddalena Jackson, Christopher Delp, Duane Bindschadler, Marc Sarrel, Ryan Wollaeger, and Doris Lam. 2011. Dynamic gate product and artifact generation from system models. In *2011 Aerospace Conference*. IEEE, 1–10.

[11] Dmitri V Kalashnikov and Sharad Mehrotra. 2006. Domain-independent data cleaning via analysis of entity-relationship graph. *ACM Transactions on Database Systems (TODS)* 31, 2 (2006), 716–767.

[12] Rabia Nuray-Turan, Dmitri V Kalashnikov, and Sharad Mehrotra. 2007. Self-tuning in graph-based reference disambiguation. In *International Conference on Database Systems for Advanced Applications*. Springer, 325–336.

[13] Yogita Rani^1 and Harish Rohil. 2013. A study of hierarchical clustering algorithm. *ter S & on Te SIT* 2 (2013), 113.

[14] Wei Shen, Jianyong Wang, and Jiawei Han. 2014. Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Transactions on Knowledge and Data Engineering* 27, 2 (2014), 443–460.

[15] W John Wilbur and Karl Sirotkin. 1992. The automatic identification of stop words. *Journal of information science* 18, 1 (1992), 45–55.