FROM ENGINEERING MODELS TO KNOWLEDGE GRAPH: DELIVERING NEW INSIGHTS INTO MODELS

Audrey Berquand [(1)], Annalisa Riccardi [(1)]

(1) Intelligent Computational Engineering Lab, Mechanical and Aerospace Engineering Department, University of Strathclyde, Glasgow, United Kingdom, {audrey.berquand, annalisa.riccardi} @strath.ac.uk

ABSTRACT

Essential information on the early stages of a mission design is contained in Engineering Models. Yet, these models are often uneasy to visualise, query, let alone compare. This study demonstrates how Knowledge Graphs can overcome these data silos, interconnect information, provide a big-picture perspective, and infer new knowledge that would have remained hidden otherwise. Following the migration of CubeSats Engineering Models to a Knowledge Graph, two case studies are explored. The first case study illustrates how graph inference can derive implicit knowledge from existing explicit concepts. In the second case study, a Natural Language Processing layer is adjoined to the Knowledge Graph to enhances the analysis of textual content. The Natural Language Processing layer relies on the document embedding method doc2vec.

1. INTRODUCTION

The ECSS-E-TM-10-25A Technical Memorandum (TM) facilitates the common data definitions and exchange of concurrent engineering studies outputs such as Engineering Models (EMs). These models hold essential information on the mission's early design, yet they are uneasy to visualise, query, or compare. This paper explores the enhancement of data linkage, reusability, and interpretability of EMs by migrating them into a Knowledge Graph (KG). Furthermore, by augmenting the graph with a reasoner, an inference engine, and a Natural Language Processing (NLP) layer, new insights into the models can be devised.

This study demonstrates via two case studies how a KG populated with EMs facilitates information retrieval and reuse at the early stages of space mission design. The first case study relies on the graph inference engine and a set of manually defined rules to infer the mass budget of each design option within an iteration. In the second case study, the similarities of missions are assessed by embedding requirements sets with a doc2vec model. All code is available at github.com/strath-ace/smart-nlp. This study makes the following contributions:

a. Provides a pipeline in Python to automatically migrate any ECSS-E-TM-10-25A-based EM to a Grakn KG.
b. Provides rules to infer a mass budget for each design option of an iteration.
c. Trains a doc2vec model on ECSS data to assess past and current missions' similarities.

2. BACKGROUND

2.1. Engineering Models

The Model-Based System Engineering design approach relies on virtual representations of systems such as EMs. The European Cooperation for Space Standardisation (ECSS) provides the ECSS-E-TM-10-25A TM [1], a standard for model-based data exchange at the early phases of engineering design. This memorandum facilitates the common data definitions and exchange of EMs produced during concurrent engineering studies.

In this study, we address the migration of EMs following this standard. The models are generated and exported with the RHEA Group's Concurrent Design & Engineering Platform 4 – Community Edition (CDP4-CE). Each migration yields several JSON files containing all data related to one iteration of an EM (e.g., design options, requirements), as well as, the parameters, templates, catalogues, and reference data specific to a concurrent design facility (the Site Directory), generic and model-specific concepts (the Site Reference Data Library and the Model Reference Data Library). Figure 1 displays an extract of an exported JSON file, featuring one element of class *ElementDefinition* named '*Subsystem- Structure*'. The lengthy strings of numbers and characters are unique identification numbers, *iid*s. The JSON keys (e.g., owner, revisionNumber) either stand for attributes or relationships. *Iid*s are used to point to elements engaged in a relationship with the element of interest.


```
classKind:                "ElementDefinition"
containedElement:
  0:                      "255f8a7b-8012-4ae8-93a8-26a318b1edf5"
  1:                      "297e6467-76f7-4a48-9fce-c73e47004914"
definition:               []
excludedDomain:           []
excludedPerson:           []
hyperLink:                []
iid:                      "95dfd909-325c-4de6-84c0-425348d5cd23"
modifiedOn:               "0001-01-01T00:00:00.000Z"
name:                     "Subsystem - Structures"
owner:                    "6ec9de88-0dcc-419e-bf02-4ed7c5f11d98"
parameter:                []
parameterGroup:           []
referencedElement:        []
revisionNumber:           1194
shortName:                "Sub_S"
```


Figure 1. Extract of a migrated JSON file

2.2. Knowledge Graphs

KGs enable to organise data with different levels of depth and complexity. By exploiting the graph architecture, KGs can model different types of relations (edges) and entities (nodes). Differently from plain graph or non-relational databases, KGs have an additional embedded layer called reasoner (or inference engine) allowing to extract implicit knowledge from existing explicit concepts.

This study relies on Grakn [2], an open-source KG development tool. Grakn implements its own query and ontology language, Graql. Reference [3] provides an extensive comparison of Grakn with more classic semantic web technologies such as the Web Ontology Language (OWL) and Resource Description Framework (RDF), justifying the predilection for Grakn. For instance, Graql requires much less complexity to model and query highly interconnected data than SQL [4]. Unlike Neo4j, Grakn includes a reasoner and is more expressive semantically [5]. All data is stored locally, securing access to proprietary information. Figure 2 exposes how the class element from Figure 1 would be represented in a KG. In all following KG schemas, entities are represented in rectangles, attributes in circles, and relationships in diamonds. Inferred elements will be indicated with dotted lines and shapes.

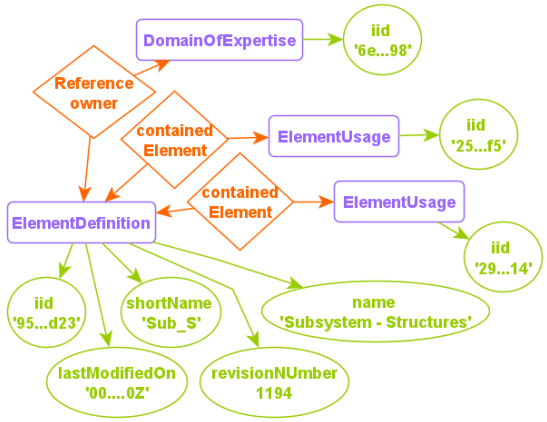

Figure 2. Representation of an ElementDefinition entity in the KG

2.3. Document Embedding with Doc2vec

Part of the information found in the EM, and subsequently migrated to the KG, is unstructured textual data. Requirements, for instance, are stored into an entity with an attribute *content* containing the requirement definition stored as a *string*. Enhancing the KG querying with an NLP layer enables semantic understanding and, therefore, grasping the meaning and context of the textual content.

This study implements a document-level embedding method, the Paragraph Vector algorithm [6], also known as doc2vec. This unsupervised algorithm builds upon the Word2vec model [7], itself based on neural networks, and used to learn word embeddings. By adjoining a Paragraph ID vector to this process, the authors of [6] were able to build a representation vector at document-level, independently of its length, representing the document concept. Paragraph Vector has two modes/architectures: Distributed Bag of Words (DBOW) and Distributed Memory (DM). In the DM architecture, similar to the word2vec Continuous Bag of Words architecture, a word is predicted based on its neighbours and the new Paragraph ID feature vector. In the DBOW architecture, similar to word2vec skip-gram, the Paragraph vector is used to classify the words of the document. Both architectures are respectively illustrated in Figure 3 and Figure 4. Although the DBOW architecture ignores the order of words, it has been found to perform better than the DM mode [8], [9]. Therefore, the DBOW architecture is chosen as the baseline for this study.

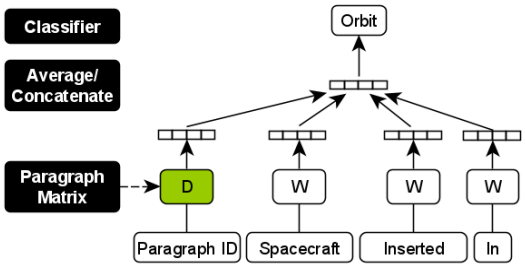

Figure 3. DM architecture based on [6]

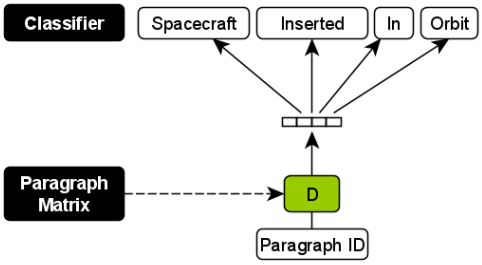

Figure 4. DBOW architecture based on [6]

3. CORPUS

3.1. Engineering Models Corpus

The EMs corpus includes three models generated with the CDP4-CE. Two of these models, STRATHcube and NEACORE, result from feasibility studies led at the University of Strathclyde's Concurrent & Collaborative Design Studio, respectively in 2020 and 2019. The third model, based on the QARMAN mission, is generated from online data.

STRATHcube's primary payload is a 3D phased array antenna for space debris detection. A secondary objective is to perform measurements during re-entry using several heat flux/pressure sensors and UV/visual spectrometers. A third experiment involves a laser onboard the International Space Station (ISS), from which the CubeSat could be launched. The mission is expected to run for a minimum of 6 months. The last iteration of the model, including three design options, is migrated to a Grakn KG.

NEACORE is an interplanetary mission involving up to six 12U CubeSats, to be flown on a single launcher between 2022 and 2023. The mission aims to estimate the relative position, velocity, and 2D shape of near-Earth objects (NEOs). The spacecraft design needed to be flexible to accommodate a camera and either a LIDAR or a spectrometer. Therefore, two design options were explored. The mission is expected to last between 3 and 6 years, with a low-thrust propulsion system.

The third EM is based on the QARMAN mission developed by the von Karman Institute [10], [11]. This 3U CubeSat deployed from the ISS in February 2020, is the first CubeSat designed to survive atmospheric re-entry. This mission was chosen for its similarities with STRATHcube in orbit, size, deployment, and payloads. The requirements from the mission were manually inferred and inserted into a CDP4-CE EM.

3.2. Doc2Vec model training corpus

The doc2vec model training corpus includes 27,016 requirements extracted from a collection of ECSS Active Standards [12]. Each requirement is composed of 39 tokens on average. The requirements are pre-processed with a domain-specific pipeline including ECSS multi-words and acronyms developed by [13].

4. METHODOLOGY

To migrate the EMs into a KG, the graph structure must first be established through a schema, defining the authorised entities, attributes, and relationships. Then, the data imported from the EMs can be inserted into the KG. Figure 5 displays the methodology followed to migrate the EMs. In the second part, a doc2vec model is trained to furthermore analyse the KG's content.

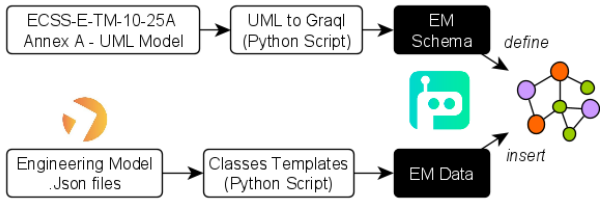

Figure 5. EMs migration flowchart: from CDP4-CE EMs to a Grakn KG.

4.1. Schema Migration: from UML to Graql

The schema layer is based on the ECSS-E-TM-10-25A Annex A Unified Modeling Language (UML) model encompassing all concepts (e.g., classes, properties, relationships) found in an EM. These concepts are mapped into Graql ones (e.g., entities, attributes, roles) as shown in Table 1. Graql recognizes a limited range of data types, allowing to define "*long*", "*double*", "*string*", '*boolean*", and "*datetime*". By default, other data types found in the UML model are mapped to a "*string*".

Table 1. Mapping of UML concepts to Graql

UML Model	Graql Schema
Class: Class name Class attributes – value type Class attributes – reference types	Entity: Entity name Entity attributes Entity roles
Association relationship → Directed composition ◆—	N-ry relationship
Inheritance relationship —▷	sub (e.g. e2 sub e1)
Property (referencing to a value type)	attribute

All concepts from the UML model, 127 entities, 108 attributes, and 148 relationships, are mapped to a Grakn schema. To distinguish the various types of UML relationships, the relationship is either annotated as a "Containment" (49% of relationships) or a "Reference" (51% of relationships) in Graql.

4.2. Engineering Models Migration

Once the structure of the KG is defined, data is inserted into the graph. The EMs iterations are exported from the CDP4-CE as JSON files. A Migration Pipeline is built in Python 3, relying on the Grakn Python Client [14] to commit new data to the server. Each entity/class requires a template function to generate a specific commit query to insert the entity/attributes/relationship into the graph keyspace.

4.3. Training of Doc2vec model

The model is trained with the open-source Gensim Python library [15]. The hyper-parameters, displayed in Table 2, are set accordingly to the recommendations found in [8].

Table 2. Hyper-parameters for model training

Parameter	Setting	Parameter Description
Vector Size	300	Dimension of the representation vectors
Epochs	400	Number of training iterations
Mode	DBOW	DBOW or DM mode
Minimum Count	1	Minimum word frequency in corpus threshold
Window	15	Left/right context window size
Subsampling	10^{-5}	Threshold to downsample high-frequency words
Negative Sampling	5	Number of negative word samples

The ECSS requirements corpus introduced in 3.2 is divided into a training set (80%) and a testing set (20%). Each requirement is considered a document. Following training, a 'sanity-check' revealed that the model would associate each document/requirement from the training set to itself with an accuracy of 0.99. Treating the testing set as unseen documents, the average cosine similarity of a document with itself is around 0.98.

5. CASE STUDIES

Once the EMs have been migrated and the doc2vec model trained, two case studies are explored to illustrate the potential of graph inference and the combination of graph technology with NLP.

5.1. Case Study 1: Inferring mass budgets

Computing the mass budget is a classic system engineering task. In this case study, we demonstrate how a mass budget can be inferred with rules and automated reasoning for each design option.

There is no relationship to express that an element is contained within the mass budget of a design option in the ECSS-E-TM-10-25A standard. However, this relationship can be inferred. A parameter is set as option dependent or independent via an attribute *isOptionDependent* of type *Boolean*.

As shown in Figure 6, when a parameter is option dependent, its *ParameterValueSet*, containing the published mass value, refers to the option it depends on. A parameter value may vary depending on the design option, therefore the relationship between the parameter and the option is done at the *ParameterValueSet* level. To ensure that only parameters identified as a mass are retained, the *Parameter* must refer to a *SimpleQuantityKind* with an attribute *name* '*mass*'. When those conditions are met, a new relationship *includedInMassBudget* is created between the *ParameterValueSet* and the *Option*. Inferred elements are indicated with dotted lines in the schema.

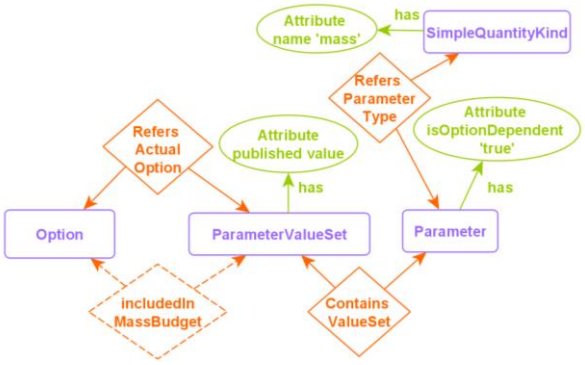

Figure 6. Inferring an includedInMassBudget relationship when the parameter is option dependent.

Even if a *Parameter* is option independent (*isOptionDependent* set to '*False*'), it does not necessarily mean that it should be linked to all options' mass budgets. The usage of the element containing the parameter must be verified as it might exclude the option as shown in Figure 7.

This logic transcribes into two rules, detailed in Table 3. Grakn requires the "*when*" side of a rule to be a conjunctive pattern while the rule's result, the "*then*" side, is atomic, meaning only one fact is inferred.

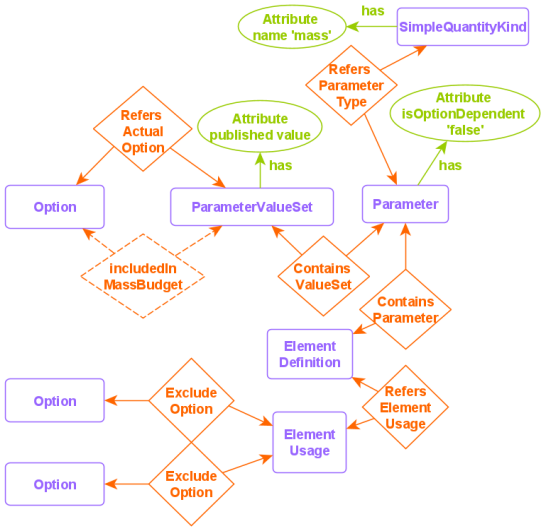

Figure 7. Inferring an includedInMassBudget relationship when the parameter is option independent.

Table 3. Pseudo-code of Rules

Rule 1: The Parameter is option dependent
when {*1. There is a ParameterValueSet, contained by an option dependent Parameter,*
2. The same Parameter refers to a SimpleQuantityKind with name "mass"
3. There is an element of class Option which the element of ParameterValueSet refers to as ActualOption.}**, then**
{*The element of class ParameterValueSet is included in the Option's mass budget.***}**

Rule 2: The Parameter is option independent
when {*1. There is a ParameterValueSet, contained by an option independent Parameter,*
2. The same Parameter Type refers to a SimpleQuantityKind with name "mass"
3. The ElementUsage associated with the same Parameter through an ElementDefinition does not exclude the Option.
}, then
{*The element of class ParameterValueSet is included in the Option's mass budget.***}**

The outcomes of the inference are visualised via the Grakn Workbase, displayed in Figure 8 and Figure 9. For clarity, only the relevant nodes and edges are shown. In Figure 8, three new relationships, appearing in purple, have successfully been inferred between each option-dependent parameter's values and the corresponding option they referred to. In Figure 9, relationships were inferred only between the parameter's mass value and the options 1 and 2 which were not excluded from the element usage.

The generation of these new relationships dramatically decreases the complexity of extracting the mass budget from the EMs. Via the Python Client, the values related to an option by an *includedInMassBudget* relationship are queried. The number of elements, scale, and mass margins associated with each value are furthermore extracted from the graph.

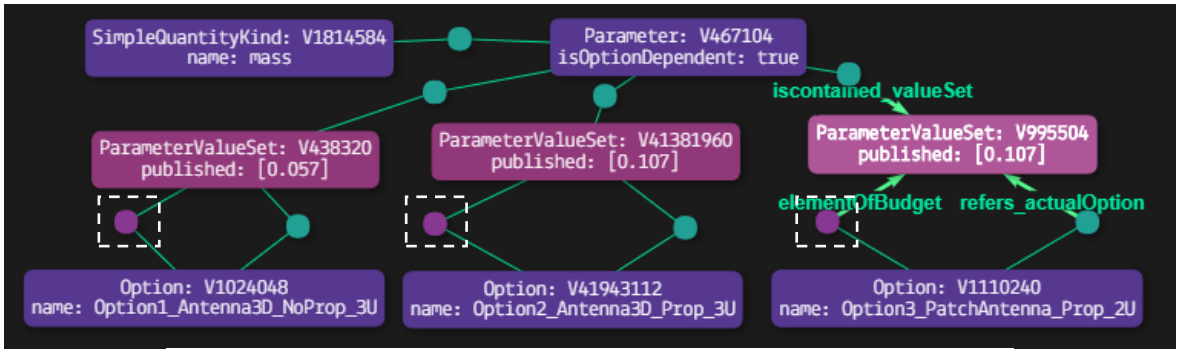

Figure 8. Inference outcomes from Rule 1 visualised with the Grakn Workbase, the three inferred edges are denoted by purple circles and framed in boxes (added manually to figure)

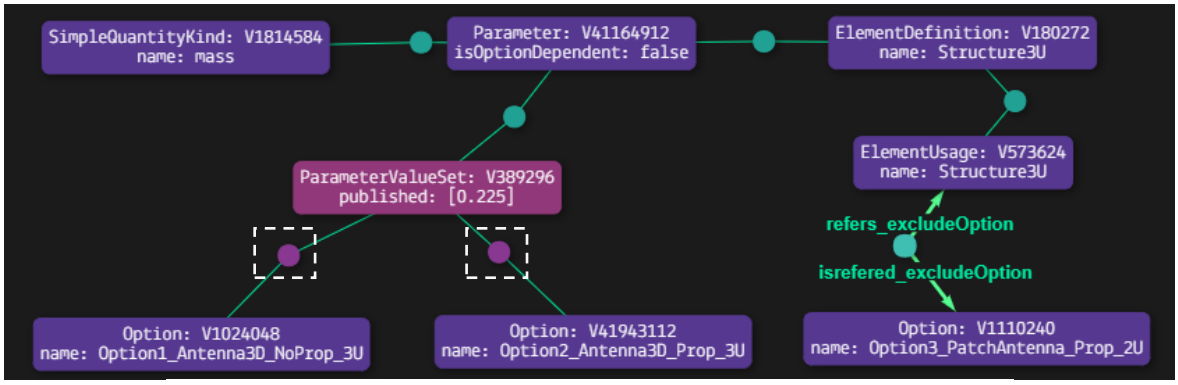

Figure 9. Inference outcomes from Rule 2 visualised with the Grakn Workbase, the two inferred edges are denoted by purple circles and framed in boxes (added manually to figure)

Table 4 compares the mass budgets of STRATHcube and NEACORE manually computed at the time of the studies, with the budgets inferred from the KG. The slight dissimilarities observed mostly originate from missing mass margins in the original models. Following the ESA CE margin philosophy [16], a mass margin of 20% is assumed by default. However, during manual computation, discussions with the experts often revealed that the actual mass margin was lower. This analysis also unveiled that the mass parameter of one equipment from STRATHcube's first design option was missing, contributing to the delta mass observed. In the case of the NEACORE's second design option, the comparison exposed an error in the manual computation, as some equipment had been wrongly incorporated into the option's budget. Removing these items from the manual computation decreased the mass delta to 0.3%.

Table 4. Comparison of mass budgets manually computed and inferred from the KG, per design option.

Satellite	Design Option	Manual Computation [kg]	Inferred from Graph [kg]	Δ [%]
STRATH cube	1	3.78	3.76	0.53
	2	5.03	5.06	0.60
	3	3.17	3.20	0.94
NEACORE	1	22.65	22.44	0.66
	2	21.27	20.65	2.5

5.2. Case Study 2: Inferring Mission Similarities

To kick-start a study and support the parameters' initialisation, it is common to investigate previous similar missions. Centralising the EMs in a KG enables to navigate through different models and assess their similarities more quickly and efficiently. In this second case study, requirements embedding with a doc2vec model are used to assess the similarities between three missions merged into a KG. As summarised in Table 5., the STRATHCube mission should emerge as more similar to the QARMAN mission than to NEACORE.

Table 5. Overview of missions' specifications

CubeSat	Size	Orbit	Scientific Goals
STRATHcube	3U	LEO (from ISS or launcher)	Space Debris Mitigation, Re-entry measurements, Wireless Power Transmission
NEACORE	12U	Inter-planetary (from launcher)	Relative position, velocity, 2D shape of NEOs
QARMAN	3U	LEO (from ISS)	Re-entry measurements

Each iteration of an EM contains several requirements, which will be considered as one 'document'. The spacecraft's requirements are therefore not embedded separately but rather as a whole set. Using the embedding learnt by the model, unseen documents, requirements sets of each iteration, extracted from the KG via the Python Client, are represented as vectors. Similar requirements sets should have close vectors representation. With the cosine similarity measures, the similarity between two vectors, therefore, two documents, is deduced. This methodology is summarised in Figure 10 for a case where *n* iterations with *n* different requirements sets are extracted from the KG.

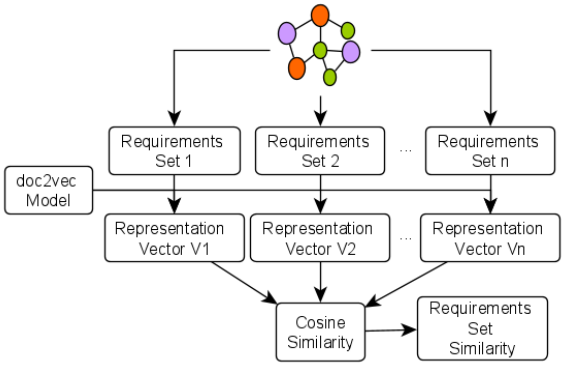

Figure 10. Process for Mission Similarity Assessment

Through the Grakn Python Client, requirements from each EMs are extracted: 126 requirements from STRATHcube, 23 from NEACORE, and 11 from QARMAN. Using the doc2vec model previously trained on the ECSS requirements, the three sets of requirements are separately embedded into three representation vectors. The similarity between these three sets of requirements are then computed with a cosine similarity measure. The results are displayed in Table 6. The cosine similarity of a requirement set w.r.t itself is kept as a 'sanity check' of the model.

Although the cosine similarity between STRATHcube and QARMAN is low (0.27/0.25), there is a significant difference with NEACORE (0.01/0.02) confirming that the former is more similar to STRATHcube than the latter. The methodology is therefore successful in assessing missions' similarities. To balance the different size of requirements set, keywords were originally selected to target requirements related to orbit, payloads, deployment, and dimensions. However, this strategy did not yield better results and was discarded.

Table 6. Cosine similarity of requirements sets (iteration number in brackets)

Mission	STRATHcube	NEACORE	QARMAN
STRATHcube (5)	0.99	0.01	**0.27**
NEACORE (4)	0.02	0.99	0.08
QARMAN (1)	**0.25**	0.06	0.98

6. CONCLUSION

This study provides a pipeline for automatically migrating EMs based on the ECSS-E-TM-10-25A TM to a Grakn KG. With the KG's inference engine, new graph edges, relationships, were inferred facilitating access to the models' content. To provide innovative insights into the KG's textual content, a document embedding model, doc2vec, was trained with ECSS requirements, to assess missions' similarities. This study has successfully demonstrated the potential of combining KG technology and NLP to enhance the data linkage, reusability, and interpretability of EMs.

In future work, additional rules could be defined to furthermore exploit the inference potential of the KG reasoner. Additional NLP methods such as Topic Modeling or the BERT language model could be implemented to unlock new insights into the models. Instead of relying on baseline hyper-parameters, the doc2vec model could be finetuned to this specific application.

7. ABBREVIATIONS AND ACRONYMS

CDF	Concurrent Design Facility
CE	Concurrent Engineering
CPD4-CE	Concurrent Design & Engineering Platform 4 – Community Edition
DBOW	Distributed Bag of Words
DM	Distributed Memory
ECSS	European Cooperation for Space Standardisation
EM	Engineering Model
ESA	European Space Agency
ISS	International Space Station
KG	Knowledge Graph
LEO	Low Earth Orbit
NEACORE	Nanospacecraft Exploration of Asteroids by Collision and Flyby Reconnaissance
NEO	Near Earth Objects
NLP	Natural Language Processing
QARMAN	QubeSat for Aerothermodynamic Research and Measurements on Ablation
STRATHcube	Space Debris Tracking, Re-entry Analysis and Wireless Power Transmission Student Partnership CubeSat
TM	Technical Memorandum
UML	Unified Modeling Language

8. ACKNOWLEDGEMENTS

The authors would like to warmly thank the Grakn Lab team for their invaluable support and wonderful open-source tool. The authors also warmly thank Sabrina Mirtcheva (ESA) for her support in accessing the raw ECSS Requirements corpus. The development of the Design Engineering Assistant is done in the frame of a Networking Partnership Initiative (NPI) involving ESA and the industrial partners RHEA Systems, Airbus and

satsearch. The authors therefore thank the NPI partners, especially Tiago Soares (ESA), Hans-Peter de Koning and Sam Generé (RHEA) for their invaluable support, and expertise.

9. REFERENCES

[1] ECSS (2010). ECSS-E-TM-10-25A – Engineering design model data exchange – CDF. Online at https://ecss.nl/hbstms/ecss-e-tm-10-25a-engineering-design-model-data-exchange-cdf-20-october-2010/ (as of July 2020)

[2] Grakn Lab. Online at https://grakn.ai/ (as of July 2020)

[3] Berquand, A., Murdaca, F., Riccardi, A., Soares, T., Generé, S., Brauer, N., and Kumar, K. (2019). Artificial Intelligence for the Early Design Phases of Space Missions. In *Proc. 2019 IEEE Aerospace Conference*, Big Sky, MT, USA, doi: 10.1109/AERO.2019.8742082.

[4] Klarman, S. (2017). Knowledge Graph Representation: GRAKN.AI or OWL? Online at https://blog.grakn.ai/knowledge-graph-representation-grakn-ai-or-owl-506065bd3f24 (as of July 2020)

[5] Altinok, D. (2020). Neo4j vs GRAKN Part II: Semantics. Online at: towardsdatascience.com/neo4j-vs-grakn-part-ii-semantics-11a0847ae7a2 (as of July 2020)

[6] Le, Q., & Mikolov, T. (2014). Distributed Representations of Sentences and Documents. In *Proc. of the 31st International Conference on Machine Learning*, Beijing, China. JMLR: W&CP volume 32

[7] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. *Advances in neural information processing systems*, pp. 3111–3119.

[8] Lau, J.H. & Baldwin,T (2016). An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation. In *Proc. 1st Workshop on Representation Learning for NLP. Pp 78-86. ACL. Berlin, Germany.* doi: 10.18653/v1/W16-1609

[9] Mendsaikhan, O., Hasegawa, H., Yamaguchi, Y. & Shimada, H. (2019). Identification of Cybersecurity Specific Content Using the Doc2Vec Language Model. In *Proc. 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, Milwaukee, WI, USA, pp. 396-401, doi: 10.1109/COMPSAC.2019.00064

[10] EoPortal Directory, QARMAN. Online at: https://directory.eoportal.org/web/eoportal/satellite-missions/q/qarman (as of July 2020)

[11] Bailet, G., Sakraker, I., Scholz & T., Muylaert, J. (2012). Qubesat for Aerothermodynamic Research and Measurement on AblatioN. In *Proc. 9th International Planetary Probe WorkShop*, Toulouse, France.

[12] ECSS Active Standards. Online at https://ecss.nl/standards/active-standards/(as of July 2020)

[13] Berquand, A., Moshfeghi, Y. & Riccardi, A. (2020). Space mission design ontology: extraction of domain-specific entities and concepts similarity analysis. In *Proc. of AIAA Scitech 2020 Forum*, Orlando, FL, doi: https://doi.org/10.2514/6.2020-2253

[14] Grakn Labs. Python Client API. Online at: https://dev.grakn.ai/docs/client-api/python (as of July 2020)

[15] Řehůřek; R. & Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proc. of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, Malta. doi: 10.13140/2.1.2393.1847

[16] Biesbroek, R. & Vennekens, J. (2017). Introduction to Concurrent Engineering. ESA Academy Presentation. Redu, Belgium.