Automated Requirements Formalisation for Agile MDE

Kevin Lano
Dept. of Informatics
King's College London, London, UK
Email: kevin.lano@kcl.ac.uk

Sobhan Yassipour-Tehrani
Dept. of Computer Science
Roehampton University, London, UK
Email: Sobhan.Tehrani@roehampton.ac.uk

M. A. Umar
Dept. of Informatics
King's College London, London, UK
Email: muhammad.umar@kcl.ac.uk

***Abstract*—Model-driven engineering (MDE) of software systems from precise specifications has become established as an important approach for rigorous software development. However, the use of MDE requires specialised skills and tools, which has limited its adoption.**

In this paper we describe techniques for automating the derivation of software specifications from requirements statements, in order to reduce the effort required in creating MDE specifications, and hence to improve the usability and agility of MDE. Natural language processing (NLP) and Machine learning (ML) are used to recognise the required data and behaviour elements of systems from textual and graphical documents, and formal specification models of the systems are created. These specifications can then be used as the basis of manual software development, or as the starting point for automated software production using MDE.

***Keywords*-Requirements formalisation; Model-driven engineering; Agile development.**

I. Introduction

Model-driven engineering (MDE) advocates the development of software systems based on the central use of precise models of the systems. The *technical* aspects necessary to employ MDE have been successfully addressed, with the standardisation of modelling languages such as UML and OCL, and the availability of many supporting tools for software specification and automated code generation (for example [11], [33], [26]). However, the *usability* of MDE and of specification/modelling languages remains a barrier to the widespread use of MDE, and most software practitioners still prefer to work at the code level. Graphical models such as UML class diagrams or state machines can be time-consuming to create and maintain for large systems. The semantics of these models may themselves be ambiguous.

We use natural language processing (NLP) and machine learning (ML) to derive models from natural language requirements statements and from sketches. This aims to reduce the effort involved in producing specifications from requirements. In an agile development context, parts of a system may be developed in different iterations, and their requirements are analysed using techniques such as interviews and exploratory prototyping in close collaboration with customer representatives. The techniques described in this paper can be used to accelerate this collaborative construction of models and prototypes from requirements, and enable customers to quickly see the semantic consequences of different requirements statements.

Our overall requirements formalisation process is shown in Figure 1. Natural language requirements in text form are used as the input, in addition, OCR is used to extract textual elements from schematic diagrams in requirements documents. An NLP toolset is used to produce an annotated text, including identification of nouns, verbs, etc, and sentence structures. These annotated sentences are then classified using a decision tree, and semantic analysis of the classified sentences is then carried out. This analysis uses a background knowledge base consisting of glossaries, both general and domain-specific. The result is a formal UML model consisting of class and use case definitions. These can be graphically visualised, and used to produce prototypes for review with customers. The process is iterative – firstly because errors in the NLP or semantic analysis steps may have arisen due to ambiguities in the requirements text, and secondly because customers may realise (as a result of the review step) that their original statements need revision or extension.

We use the AgileUML toolset as the basis for this work [11]. The toolset provides UML and transformation specification facilities, in the UML-RSDS subset of UML, including graphical class diagram and use case models, and additional state machine, interaction diagram and SysML support. To facilitate faster creation and editing of class diagrams and use cases, the textual KM3 notation can be used as an alternative to diagrams [12]. We aim to provide automated RE support for software modelling across a wide range of domains, however with the capability to utilise domain-specific background knowledge bases and models when these are available.

In Sections I-A and I-B we summarise the different forms of NLP and ML technique which are relevant for automated requirements engineering (RE). In Sections II-A, II-B and II-C we define specific techniques for formalisation of data and behavioural requirements from natural language statements. Section III evaluates these techniques on examples taken from diverse datasets of requirements statements.

Section IV surveys related work, and Section V gives conclusions. We provide all data of our evaluation cases and results in [5].

A. Natural Language Processing (NLP)

NLP is a collection of techniques for the processing of natural language text, including part-of-speech (POS) tagging/classification, tokenisation and splitting of text into sentences, lemmatisation, syntax analysis, dependency analysis

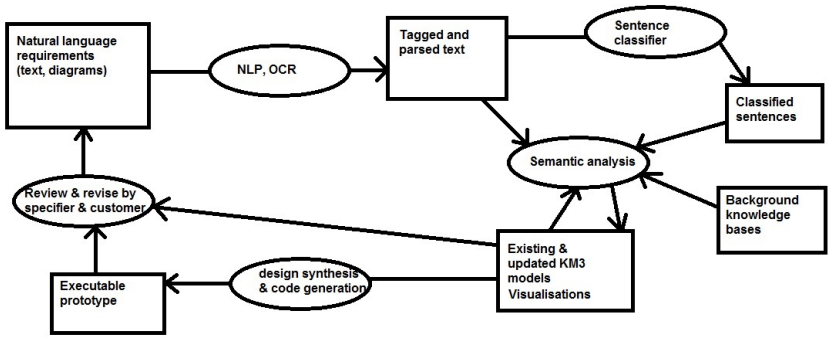

Fig. 1. Automated requirements engineering in the agile MDE process

and reference correlation. NLP tools include Stanford NLP [32], Apache OpenNLP [4], iOS NLP Framework, and WordNet [36].

The standard parts of speech include [28]: Determiners – tagged as *_DT*, eg., "a", "the"; Nouns – *_NN* for singular nouns and *_NNS* for plural; Proper nouns – *_NNP* and *_NNPS*; Adjectives – *_JJ*, *_JJR* for relative adjectives, *_JJS* for superlatives; Modal verbs – *_MD* such as "should", "must"; Verbs – *_VB* for the base form of a verb, *_VBP* for present tense except 3rd person singular, *_VBZ* for present tense 3rd person singular, *_VBG* for gerunds, *_VBD* for past tense; Adverbs – *_RB*; Prepositions/subordinating conjunctions – *_IN*.

For specialised purposes, additional parts of speech may be defined, as in [37]. NLP has been a key element of automated RE approaches, eg., [6], [10]. However, the trained models available with the existing NLP tools are usually oriented towards general English text, which differs significantly from the subset of English typically used in software requirements statements. In particular, requirements statements do not usually use colloquial or casual English, and they use computing/software terminology. Words such as "persisted", "form", "schema" and "backlog" are used in a more specific way in software descriptions, compared to their use in general English. The existing models therefore sometimes misclassify words in requirements statements (eg., "stores" is misclassified as a noun by both Stanford NLP and Apache OpenNLP, even when used as a verb, and "existing" used as an adjective is often misclassified as a gerund). For this reason we decided to re-train a POS tagging model using a corpus of 19 requirements statements which we collected for data and behavioural requirements analysis (Sections II-A, II-B, II-C). The corpus contains 1267 sentences.

Retraining of the tagger reduced the error rates on 5 new test cases (containing 427 sentences) from 0.4% per word and 9.4% per sentence, to 0.2% per word and 5.6% per sentence. The retrained tagger was effective in removing the more serious errors originally encountered, however some tagging and syntax analysis errors still arise in some cases.

B. Machine Learning (ML)

Machine learning covers a wide range of techniques by which knowledge about patterns and relationships between data is gained and represented as implicit or explicit rules in a software system. ML can be used for classification, translation or prediction. In particular, ML is used to create the part-of-speech and other models used in NLP tools. ML techniques include K-nearest neighbours (KNN), decision trees, inductive logic programming (ILP) and neural nets. A key distinction can be made between techniques such as decision trees and ILP where explicit rules are learnt from data, and techniques such as neural nets where the learned knowledge is in an implicit form (consisting of the weights of connections in the trained network). In recent years there have been substantial advances in neural networks (recurrent neural networks or RNNs) which possess a 'memory' of a sequence of inputs, enabling them to perform prediction tasks and process data (such as natural language texts or programs) that consist of a connected sequence of elements (sentences) [25]. In principle, such networks should be capable of making use of information distributed amongst multiple sentences in a requirements statement, in order to provide a more integrated analysis, compared to approaches which analyse sentences in isolation. However, the main application of ML within requirements formalisation has been sentence and word classification, using neural nets or decision trees [35]. In this paper we also use a decision tree classifier to classify requirements sentences into categories.

Toolsets for ML include Google MLKit, Tensorflow, Keras, ScikitLearn and Theano.

II. Automated Requirements Formalisation of Software Applications

Software requirements statements are typically expressed in a combination of natural language text and informal diagrams, with text as the predominant element. Requirements statements may include non-technical issues, such as the roles of stakeholders in the development process [29]. In this paper we only consider the technical requirements of the data and functionality of the system to be constructed.

There are typically four main stages in any requirements engineering process [34]:

1) Domain analysis and requirements elicitation
2) Evaluation and negotiation
3) Requirements specification and formalisation
4) Requirements validation and verification.

Our automated requirements engineering techniques focus upon requirements *formalisation* in order to carry out the requirements specification of technical requirements, and do not address requirements elicitation in an active sense. However, the process may clarify the understanding of requirements by the customer and developer by providing a formal model and visualisation of requirements statements.

We aim to extract formalised class diagrams and use case models suitable for the AgileUML subset of UML, so that these can be used as the starting point of model-driven development of the software application. The formalisation process can be divided into extraction of a data model (Section II-A) and extraction of a behavioural model (Sections II-B, II-C). In both parts, a three stage process is used: (i) NLP to obtain linguistic information on requirements statements; (ii) classify requirements statements using a ML classifier or heuristics; (iii) semantic analysis of classified statements.

A. Deriving data model specifications from requirements statements

There has been extensive research into the extraction of UML and other models from requirements statements expressed in natural language [35], [38]. However, despite advances in NLP and ML techniques [25] for text analysis and translation, it is still beyond the state of the art to automatically extract technical information from arbitrary English text, so we restrict consideration to requirements statements which consist of sentences in present tense, and to be in subject-verb-object format with a direct object (SVO format). There are many different styles of requirements statements. Embedded control systems are usually defined with respect to particular hardware (the EUC), and a detailed specification of the EUC should already exist, and can be referred to in order to identify named entities in the requirements statement. For example, the production cell case study [18] and diffusion pump [9] are of this kind. This is also the case for telecom system specifications based on particular communications technologies and protocols [1]. However, even general information system requirements statements may be expressed in terms of specific implementations and APIs, and in such cases, we assume the existence of formal models of the relevant components. Our approach accommodates this situation by integrating the requirements formalisation approach with the model editor of AgileUML, so that any prior existing model can be loaded and its elements searched to identify and classify named terms in the analysed requirements statement. For the NWDAF telecoms system, a detailed data model is defined in [2], and we formalise this as a KM3 model (mmNWDAFcomplete.km3 in [5]) to support behavioural requirements formalisation. In the first stage of formalisation, we restrict attention to specifications of data.

To assist in statement classification and semantic analysis, we define a knowledge base of software requirements terminology, including parts-of-speech (POS) information on words, synonyms, and semantic properties of terms (eg., that "radius" is likely to denote a non-negative real-valued attribute).

For specific domains such as finance or telecoms, a domain-specific knowledge base is also necessary, to capture information on terms with a specialised meaning in the domain, such as "share", "term", "equity" and "fixed-income" in finance, and "connection" and "cell" in telecoms. The domain-specific knowledge base takes precedence over the general knowledge base, and both could be used to retrain NLP models to provide correct POS and sentence analysis for requirements statements (Section I-A).

We use the Stanford NLP toolset [32] and Apache OpenNLP [4] to perform part-of-speech tagging of English sentences, to construct syntax trees, and to identify dependencies within sentences. The resulting information is then used as input for a decision-tree classifier (CART) which identifies sentences as either:

1) Definitions of classes, including attributes of the class
2) Definitions of specialisation/generalisation relations between classes
3) Definitions of associations (reference features of classes)
4) Definitions of invariants or other logical constraints of classes.

Sentences in the requirements statements should be decomposed so that each sentence falls into only one of the above categories.

After classification, a detailed syntactic and semantic analysis can be applied to extract model content from the text information. Successive sentences are analysed taking account of the partial formal models already obtained from preceding sentences, so that models are progressively elaborated. Contradictions, redundancy and omissions within requirements statements could also be identified during this process.

Examples of data definition requirements sentences from the k3 case of [13] are:

```
A class shall be either a non-clinical class
or a clinical class.

A non-clinical class shall specify the
course name, lecture room requirements
and instructor needs.

A clinical class shall specify the course
name, lecture room requirements,
clinical site needs,
lecture instructor needs and
clinical lab instructor needs.
```

In order to classify sentences as one of the above four categories, a CART decision tree was constructed from training data, with decisions based on several features: (i) the similarity of the main verb to one characteristic of the classification category (eg., similarity to "have", "has", "holds",

"consists", "specifies", "comprises", "is defined by", "stores", "records" or "maintains", in the case of class definitions); (ii) occurrences of terms (such as "instance" after a candidate class name) indicating the classification category; (iii) sentence structure and intra-sentence dependencies from NLP syntax and dependency analysis; (iv) classifications of terms as attributes, classes or references from the general knowledge base.

In order to detect synonyms, we use a thesaurus and word distance measures of name edit distance (NSS) [17] and name semantic similarity (NMS) [14]. Users can set a threshold for the level of similarity which is to be regarded as significant.

We also use the thesaurus format [7] to represent domain and general background knowledge in knowledge bases. An example from the general requirements knowledge base is:

```
<CONCEPT>
<DESCRIPTOR>diameter</DESCRIPTOR>
<POS>NN</POS>
<SEM type="double"
  stereotype="nonNegative">attribute</SEM>
<PT>diameter</PT>
<NT>gauge</NT>
<NT>thickness</NT>
<NT>width</NT>
</CONCEPT>
```

Any preferred term (*PT*) or non-preferred term (*NT*) in the concept word group will be treated as having the same semantics and part of speech defined in the *SEM* and *POS* clauses.

The data requirements statement classifier was trained on a set of 19 cases (application requirements documents) containing 95 requirements statements. The trained classifier and the requirements formalisation algorithm were then evaluated on a separate test set of 11 cases containing 67 statements (Table I). These cases include extracts from [29], [20] and [15], and three cases from [13]. Thus 59% of the dataset is used for training, and 41% for testing.

The induced decision tree is:

```
|--- has-verb <= 0.50
|   |--- is-verb <= 0.50
|   |   |--- relates-verb <= 0.50
|   |   |   |--- class: other
|   |   |--- relates-verb >  0.50
|   |   |   |--- class:
          association-definition
|   |--- is-verb >  0.50
|   |   |--- class: specialisation
|--- has-verb >  0.50
|   |--- class: class-definition
```

It is noticeable that decisions are based entirely on verb kind, and that aspects such as the kind of elements in the statement or occurrences of words such as "instance/instances" were not significant, due to the low frequency of such discriminator information in the dataset. We encoded the above rules into the AgileUML toolset to perform the sentence classification stage.

Subsequent to classification, detailed semantic analysis takes place in three phases successively applied over the sequence of sentences in the requirements statement:

1) Recognition of classes:
   a) Classes already existing in the software model or recognised in a previous sentence;
   b) Classes according to background knowledge in the general or specialised knowledge base (eg, "bond");
   c) Classes according to their role in the sentence (eg., as the subject of the principal form of class definition sentence);
   d) Proper nouns (tagged as NNP or NNPS) – the singular form of the noun is taken as the class name. A more liberal approach could be to also consider other nouns (NN or NNS) with an initial capital as proper nouns, if they do not appear as the first word of a sentence.

   Inheritance relations are established based on the sentences classified as defining specialisations.
2) Recognition of attributes:
   a) Attributes already known from the pre-existing software model or from previous sentences;
   b) Attributes according to background knowledge (eg., "radius", "name");
   c) Attributes according to the role in the sentence (eg., in the object of class definition sentences, and not a class or role).

   The types, multiplicities and stereotypes of attributes are also extracted.
3) Reference recognition:
   a) References already known from the pre-existing software model or from previous sentences;
   b) References according to background knowledge (eg., "parent", "neighbours");
   c) Reference according to sentence form (eg., "Each project consists of a series of workpackages");
   d) Name is initial lower case, and initial uppercased singular form of name is the name of a known class (eg, "bonds", "investor").

   Reference types, multiplicities and stereotypes are also extracted (eg., "series" suggests an $\{ordered\}$ stereotype on a feature).

Formalised data models are represented as KM3 format specifications of class diagrams.

Traceability of the specification with respect to the requirements is achieved by recording a many-many relation which identifies which requirements statement elements contributed to the definition of each derived model element. The originating elements of a model element x are recorded as stereotypes of x. Thus it is possible to identify which requirements statements contributed to the definition of x.

For example, the formalisation of the k3 case includes the following classes:

```
class Class {
  stereotype originator="1";
}

class Non-clinicalClass extends Class {
  stereotype originator="1";
  stereotype modifiedBy="2";

  attribute courseName : String;
  attribute lectureRoomRequirements :
    Set(String);
  attribute instructorNeeds : Set(String);
}

class ClinicalClass extends Class {
  stereotype originator="1";
  stereotype modifiedBy="3";

  attribute courseName : String;
  attribute lectureRoomRequirements :
    Set(String);
  attribute clinicalSiteNeeds : Set(String);
  attribute lectureInstructorNeeds :
     Set(String);
  attribute clinicalLabInstructorNeeds :
     Set(String);
}
```

originator refers to the sentence which led to the creation of the class, and *modifiedBy* to sentences which led to augmentation of its definition.

Traceability assists in an iterative approach to requirements formalisation, whereby a specifier can identify the requirements statements which lead to incorrect model elements, and hence modify the statements before re-applying the formalisation procedure. Discussion and agreement of changes with the customer/stakeholders may need to take place if they significantly impact the requirements. Ideally, the textual requirements specification and its formalisation should be maintained together.

B. Deriving behavioural model specifications from requirements statements

Subsequent to data formalisation, a further phase of behavioural formalisation can be applied, which can utilise the extracted data specification for a case. For example, the NWDAF case [2], stroke assistant case [15] and k3 case [13] each contain both data and behavioural requirements.

We assume that behavioural requirements are of two kinds:

- Expected functionalities/services offered by the system, expressed as user stories. These can be formalised as UML use cases in AgileUML.
- Operation specifications for operations/methods of particular classes, expressed in terms of the expected inputs and outputs of these operations.

In this section we focus upon the first case, and Section II-C considers the formalisation of operations. The sentence classification stage for behavioural requirements uses heuristic rules (based on the main verb and structure of the sentences) instead of an ML-derived classifier.

User stories are the principal form of behaviour requirement used in agile methods. They express some unit of functionality which a user of the system expects from the system, in other words, the use cases which the system should support. User stories are typically expressed in the format

[*actor identification*] *goal* [*justification*]

where the actor identification and justification are optional.

For example:

"As an A, I wish to B, in order to C"

The actor identification defines which system actor the use case is for. The goal describes the intended use case actions, and the justification explains its purpose. These correspond to the graphical use case form of Figure 2.

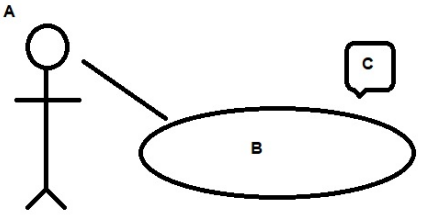

Fig. 2. Use case derivation from user stories

Other alternative formats for user stories are

"The A should be able to B, in order to C"
"The A must be able to B, so that C"
"The system must allow A to B"
"The system shall provide B"
"A will be able to B"

or simply *B* by itself.

In each format, a reference to any actor *A* occurs prior to *B*. A modal verb ("should", "must", etc) identifies the obligation to provide the functionality *B*. Statements *B* and *C* typically refer to the classes and features recognised in the data formalisation stage. *B* should begin with a verb in infinitive form (the tag *VB*). In the absence of a formalised data model, classes are recognised according to background knowledge and their role in the sentence.

In addition to the background knowledge file *output/background.txt* on nouns, we also provide a knowledge base *output/verbs.txt* of verbs and their classifications (*read*, *edit*, *create*, *delete*, *other*). This file is in the same thesaurus format as the background file for nouns, and can be edited by the user to provide further information on verbs. User stories that represent standard functionalities such as creating, editing, deleting and reading instances of a class, can be classified into these categories based on the verbs used in the goal part *B* of the user story. The classified sentence is then

formalised as a use case in KM3 format. The semantics of the use case depends on the classification of the user story, for example:

- *createE* use case – has a result variable of type *E*, and input parameters *p*$*x* for any data feature *p* of *E* referred to in the statement. The use case postcondition

$$E{\rightarrow}exists(ex \mid ex.p_1 = p_1\$x \ \& \ ... \ \& \ ex.p_n = p_n\$x \ \& \ result = ex)$$

  can be specified, where the conjunction is over all the input parameters p_1 to p_n for *E* features.
- *editE* use case – has a parameter *ex* of type *E*, and input parameters *p*$*x* for any data feature *p* of *E* referred to in the statement. There are also parameters for other classes mentioned in the statement. The postcondition

$$ex.p_1 = p_1\$x \ \& \ ... \ \& \ ex.p_n = p_n\$x$$

  can be specified.

The actor *A* of a use case is recorded in a stereotype *actor* = "*A*" of the formalised use case. If *A* is a class name, then an instance of the actor becomes the first parameter of the use case. Eg., an example of a creational use case is the following from the k3 user stories:

```
Program Administrators/Nursing Staff
Members shall be able to
create a new Program of Study.
```

This is formalised as:

```
usecase createANewProgramOfStudy : Program
{ parameter px :
    ProgramAdministrators_
             NursingStaffMembers;
  stereotype originator="4";
  stereotype
    actor="ProgramAdministrators_
          NursingStaffMembers";
  stereotype create;

  ::
    true =>
      Program->exists( programx |
                    result = programx );
}
```

Data dependencies between classes and use cases can then be computed and shown visually.

C. Deriving operation specifications from requirements statements

Operation definitions in natural language typically consist of several parts:

- The operation name and identification of the class/module it belongs to;
- Input parameters;
- Preconditions/assumptions and other restrictions on how it can be used;
- Internal behaviour and processing, including calls to other operations;
- Output parameters;
- Postconditions and results.

All of these elements will be used in producing formalised operation specifications. Other aspects, such as non-functional requirements on performance, are not analysed.

The first step in operation formalisation is to classify statements and statement parts into the above categories. This is carried out based on the terminology used, eg., words such as "responds", "returns" are indicative of a description of output parameters. The identification of parameter names and types is performed in the same manner as the identification of class data features (Section II-A). Sentences which refer only to input parameters are assumed to express preconditions, whilst sentences involving output parameters are considered to express postconditions. Internal processing, including conditional behaviour, is recognised from the form of the main verb, and from occurrences of conditional terms. Such processing is listed, in the order of the textual statements, within the operation activity. Reference correlation is used to link implicit references to the same operation in separate sentences. This analysis is not able to extract precise functional definitions. Some form of specification by-example analysis would be necessary to achieve this [16].

III. Evaluation

In this section we evaluate our requirements formalisation process using a wide range of requirements statements across several different domains. We use three industrial cases from avionics, medical and telecoms domains, and the published datasets [21], [13].

Our evaluation procedure is to assess the correctness of sentence classifications with respect to a manual classification produced by a requirements engineer (the second author), and to assess the correctness of formalised models with respect to reference models manually produced by a modelling expert (the first author). The elements (classes, attributes, references and use cases) of formalised models are compared wrt the elements of reference models. We use the F-measure $F = \frac{2*p*r}{p+r}$ where precision $p = \frac{correct\ \ identifications}{total\ \ identified}$ and recall $r = \frac{correct\ \ identifications}{total\ \ correct}$. Thus for example, if the behavioural formalisation derives two correct use cases out of three identified, but two further use cases have not been recognised, we have $p = 2/3$, $r = 2/4$, and $F = 0.57$. In principle this evaluation procedure could be automated.

A. Data requirements formalisation

Table I shows the accuracy measures for the classifier and formalisation process on the evaluation cases. The size of the case is shown as the number of sentences in the case. The execution time taken for classification and formalisation is also shown.

The results show a consistently high accuracy in classification, and a generally good accuracy for formalisation. Both recall and precision should be high for formalisation,

TABLE I
EVALUATION OF DATA REQUIREMENTS FORMALISATION.

Case (Size)	*Classification F-measure*	*Formalisation F-measure*	*Execution Time (s)*
1 (1)	1.0	0.9	0.316
2 (5)	1.0	0.95	0.366
3 (6)	1.0	0.82	0.405
4 (5)	0.8	0.74	0.385
5 (5)	0.8	0.96	0.385
6 (14)	0.93	0.81	0.654
7 (4) Track former [20]	0.75	0.74	0.517
8 (8) Stroke assistant [15]	0.88	0.85	0.304
k1 (2) [13]	1.0	0.86	0.4
k2 (8) [13]	0.88	0.77	1.1
k3 (9) [13]	1.0	0.84	0.67
Averages	0.91	0.84	0.5

since incorrect and incomplete formalisations are equally a hindrance to developers. In all of these evaluation cases recall and precision for formalisation were both over 0.65.

B. Formalisation of user stories

A large dataset of behavioural requirements statements for 22 software applications defined by user stories is given by [21]. In total there are 1678 user stories in the dataset cases. We use this dataset to evaluate our use case formalisation approach, together with 5 other cases, including the large CONNECT case of 408 user stories from [22]. Table II shows the accuracy of use case classification and the accuracy of the formalised UML models, with respect to the results obtained by the modelling expert. The combined time taken for classification and formalisation is also shown.

TABLE II
EVALUATION OF USE CASE REQUIREMENTS FORMALISATION.

Case (# sentences)	*Classifier F-measure*	*Formalisation F-measure*	*Execution Time (s)*
Stroke assistant (8) [15]	1.0	0.92	0.37
Banking app (4)	1.0	0.89	0.33
g02 (99)	0.93	0.92	1.42
g03 (60)	0.95	0.91	0.98
g04 (51)	0.98	0.96	0.74
g05 (53)	1.0	0.92	1.03
g08 (68)	0.97	0.93	1.2
g10 (98)	0.96	0.94	0.55
g11 (74)	0.98	0.97	1.1
g12 (53)	0.96	0.88	0.78
g13 (53)	1.0	0.95	0.92
g14 (67)	0.98	0.95	1.1
g16 (66)	0.98	0.95	1.1
g17 (64)	1.0	0.9	1.1
g18 (101)	1.0	0.94	1.5
g19 (138)	0.99	0.97	0.85
g21 (73)	0.96	0.95	1.1
g22 (83)	0.86	0.96	0.7
g23 (56)	0.98	0.94	0.47
g24 (52)	1.0	0.98	0.47
g25 (100)	0.98	0.96	0.68
g26 (100)	0.98	0.97	0.69
g27 (115)	0.97	0.93	2.03
g28 (60)	1.0	0.93	0.8
CONNECT (408) [22]	0.89	0.93	10.2
FABS (94) [22]	0.97	0.91	1.7
k3ucs (26) [13]	0.96	0.94	0.82
Averages	0.97	0.94	1.29

The averages for use case formalisation are higher than those for data formalisation (Table I), which is perhaps due to the more structured nature of user stories, compared to the wide variety of ways in which data requirements can be expressed. On the other hand, the execution time is longer, due to the substantially larger texts in the behaviour requirements cases. The main source of imprecision in behaviour formalisation is due to errors in tagging, eg., tagging of "And" as a *NNP* instead of *CC*. In addition, (i) general concepts such as "Application", "UI" or "System" are incorrectly recognised as classes; (ii) successive nouns in the actor identification part are not combined, but become separate classes, eg., "Camp Administrator" is represented as two separate classes.

These issues could be corrected by (i) defining a separate list of keywords/blocked words, which cannot be used as class names; (ii) automatically combining successive nouns in the actor identification.

We prefer to leave (ii) as a developer choice, so that if they want the combined term to become a class, they can write it as one word "CampAdministrator", etc.

C. Operation requirements formalisation

To evaluate the effectiveness of the operation formalisation approach, we use 5 examples of operation specification, including the track former case [20] and the Network Data Analytics Function (NWDAF) from [1]. Only relatively few examples of operation specification were available in the cases we considered. Table III shows the results of this evaluation.

TABLE III
EVALUATION OF OPERATION REQUIREMENTS FORMALISATION.

Case (# sentences)	*Classifier F-measure*	*Formalisation F-measure*	*Execution Time (s)*
Ops. Ex. 1 (7)	1.0	1.0	0.833
Pre-post test (11)	0.82	0.77	0.919
Ops. Ex. 2 (5)	1.0	0.82	0.739
Track former [20] (7)	0.86	0.72	1.225
NWDAF [1] (7)	0.57	1.0	0.818
Averages	0.85	0.86	0.91

The results for operation formalisation are somewhat lower than for user story formalisation, due to the greater variability in the forms in which operation requirements are expressed.

IV. RELATED WORK

Two recent systematic literature reviews (SLR) of papers in the field of automated requirements engineering are [35], [38]. The review [35] considers 54 studies in the field of ML and NLP-assisted RE for software modelling, while [38] considers 404 papers which use NLP techniques for general RE activities. The surveys identified that most adopted approaches were semi-automated rather than fully-automated. A lack of widespread adoption and of industrial evaluation is highlighted as a problem with the surveyed approaches [38].

Subsequent to these SLRs, further relevant papers [8], [30], [31], [37] have been published.

In general, NLP/ML approaches for requirements formalisation have used NLP analysis of source texts to produce

enhanced requirements statements, which are then used as inputs to a semantic analysis stage, which identifies semantic elements in the statements and extracts these as formal model elements and relations. The approaches for semantic analysis divide between heuristic approaches with expert-provided explicit rules used to classify and formalise statement elements, eg., [23], [27], or ML approaches with supervised learning, eg., [39]. Heuristic approaches have the advantage of encoding expert knowledge, however they can be inflexible in their analysis, and restrictive in the forms of input texts which they can process, whilst ML approaches are in-principle more flexible and wider in their coverage of requirements statements, but can lack rationales for their empirically-derived rules. Our approach uses a hybrid of heuristic and ML techniques based on NLP-derived linguistic information, as described in Sections II-A, II-B, II-C. We found that a heuristic approach was sufficient for user story formalisation, since these have a restricted format and consistent structure, but ML was necessary to analyse the more diverse forms of data requirements statements.

Apart from the lack of industrial application identified by [38], systematic comparative evaluation of different requirements formalisation approaches is lacking. In addition, flexibility and user-customisation of tools for requirements formalisation is an important issue. In future work we aim to construct a tool framework for automated RE which facilitates the use of alternative approaches and tool customisation. This will enable comparative evaluations of different approaches to be systematically carried out.

Because natural language is ambiguous and imprecise, it is unrealistic to expect any requirements formalisation process to be 100% automated and accurate. In particular, the limitations of NLP impose a certain margin of error on any analysis based on NLP [19]. Instead, as in the approach of [8], we aim to provide a tool which assists the software specifier by deriving the substantial majority of the semantic content of a requirements statement from natural language documents, and then enables iterative refinement of the specification based on interactive correction and clarification of the requirements statement.

V. Summary

Overall, our requirements formalisation approach provides a means to rapidly build models from natural language requirements, and to visualise, analyse and prototype systems from formalised requirements. The approach has the distinctive advantage of being integrated into an established MDE toolset, enabling the use of existing models to support the formalisation of new requirements.

References

[1] 3GPP, https://www.3gpp.org, 2020.

[2] 3GPP, *TS 29.520 V17.1.0*, Network Data Analytics Services, 2020.

[3] AgileUML repository, https://github.com/eclipse/agileuml/, 2021.

[4] Apache OpenNLP Toolkit, opennlp.apache.org, 2021.

[5] *AI for MDE dataset*, Doi 10.5281/zenodo.5089886, https://zenodo.org/record/5089886, 2021.

[6] V. Berzins, et al., *Innovations in natural language document processing for requirements engineering*, Innovations for Requirement Analysis conference, LNCS vol. 5320, Springer, 2008, pp. 125–146.

[7] British Standards Institution, *Thesauri*, in BS 8723: Structured vocabularies for information retrieval, 2005.

[8] L. Burgueno, R. Clariso, S. Gerard, S. Li, J. Cabot, *An NLP-based architecture for the autocompletion of partial domain models*, Adv. Inf. Sys. Engineering, Springer, 2021, pp. 91–106.

[9] J. Campos, M. Harrison, *Comparing specification styles: the interactive behaviour of an infusion pump*, EICS 2011.

[10] O. Dawood, A. Sahraoui, *From requirements engineering to UML using NLP*, European Journal of Eng. Res. & Science, 2(1), 44, 2017.

[11] Eclipse AgileUML project, https://projects.eclipse.org/projects/modeling.agileuml, 2021.

[12] F. Jouault, J. Bezivin, *KM3: a DSL for metamodel specification*, ATLAS team, INRIA, 2006.

[13] www.kaggle.com/iamsouvik/software-requirements-dataset, accessed 20th May 2021.

[14] M. Kessentini, A. Ouni, P. Langer, M. Wimmer, S. Bechikh, *Search-based metamodel matching with structural and syntactic measures*, JSS, vol. 97 (2014), pp. 1–14.

[15] King's Health Partners, *Stroke Recovery Assistant project brief, version 1.0*, March 2012.

[16] K. Lano et al, *Model Transformation Development using Automated Requirements Analysis, Metamodel Matching and Transformation By-Example*, ACM TOSEM, to appear, 2021.

[17] I. Levenshtein, *Binary codes capable of correcting deletions, insertions and reversals*, Cybernetics and control theory, 10(8), 1966, pp. 707–710.

[18] C. Lewerentz, T. Lindner, *Task description of a flexible production cell with real time properties*, KORSO, LNCS vol. 1009, 2005, pp. 388–416.

[19] C. Manning, *Part-of-speech tagging from 97% to 100%: is it time for some linguistics?*, CICLing 2011.

[20] Marconi Command and Control Systems, *A requirement specification for a simple track former*, 1990.

[21] Mendeley user story dataset, https://data.mendeley.com/dataset/7zbk8zsd8y/1, accessed January 5th, 2021.

[22] Mendeley user story dataset, https://data.mendeley.com/datasets/bw9md35c29/1, accessed May 20th, 2021.

[23] P. More, R. Phalnikar, *Generating UML diagrams from natural language specifications*, Int. Journal of Applied Inf. Sys., vol. 1, 2012, pp. 19–23.

[24] OMG, *Object Constraint Language 2.4 Specification*, 2014.

[25] D. Otter et al., *A survey of the usages of deep learning in natural language processing*, IEEE Trans. Neural network learning systems, April 2020, pp. 1–21.

[26] Papyrus toolset, https://www.eclipse.org/papyrus, 2020.

[27] M. Robeer et al., *Automated extraction of conceptual models from user stories via NLP*, 24th Int. Requirements Eng. Conf., IEEE, 2016, pp. 196–205.

[28] B. Santorini, *Part-of-speech tagging guidelines for the Penn Treebank project*, Dept. of Information and Computer Science, University of Pennsylvania, 1990.

[29] J. Robertson, S. Robertson, *Volere requirements specification template*, https://www.volere.org, 2020.

[30] R. Saini et al., *DoMoBOT: A bot for automated and interactive domain modelling*, MDE Intelligence, 2020.

[31] R. Saini et al., *Towards queryable and traceable domain models*, 28th Int. Requirements Eng. Conf., IEEE, 2020, pp. 334–339.

[32] Stanford NLP, https://nlp.stanford.edu/software/, 2020.

[33] Simulink toolset, https://www.mathworks.com/products/simulink.html, 2020.

[34] I. Sommerville, P. Sawyer, *Requirements Engineering*, Wiley, 1997.

[35] M. Umar, *Automated requirements engineering framework for agile development*, ICSEA 2020.

[36] WordNet, wordnet.princeton.edu, 2021.

[37] X. Xu, K. Chen, H. Cai, *Automatic utility permitting within highway right-of-way via a generic UML/OCL model and NLP*, J. Constr. Eng. Management, 146(12), 2020.

[38] L. Zhao et al., *Natural language processing for requirements engineering: a systematic mapping study*, ACM Computing Surveys, 2020.

[39] S. Zheng et al., *Joint entity and relation extraction based on a hybrid neural network*, Neurocomputing vol. 257, 2017, pp. 59–66.