Enhancing agile model-driven engineering with automated formal verification of BPMN models

Kimia Kolahdouz[1], Shekoufeh Kolahdouz Rahimi[2,*] and Kevin Lano[3]

[1]Department of Software Engineering, University of Isfahan, Iran
[2]School of Arts, University of Roehampton, London, UK
[3]Department of Informatics, King's College London, UK

Abstract

Misinterpretation of requirements remains a major challenge in software development, often leading to project failures. Agile Model-Driven Engineering promotes iterative modeling and validation to enhance software quality and adaptability. Business Process Model and Notation (BPMN) serves as a widely adopted standard for visually representing system processes, ensuring a shared understanding among stakeholders. However, the complexity of BPMN can introduce structural errors, compromising model accuracy and reliability. This paper presents a formal verification approach leveraging Object Constraint Language (OCL) to enforce BPMN structural correctness, aligning with Agile principles by enabling early error detection and refinement. Natural Language Processing (NLP) techniques are employed to automate the translation of BPMN standard rules into OCL constraints, enhancing model validation.

Keywords

Business Process Model and Notation, Object Constraint Language, Natural Language Processing, Model Verification

1. Introduction

The failure of software projects is often due to misunderstandings of system requirements during the analysis phase. A key challenge in this phase is bridging the communication gap between business stakeholders and IT developers by establishing a common language [1]. To address this issue, there has been a growing need for a modeling language that is both expressive and formal, enabling clear and structured graphical representations of business processes. Among various notations, Business Process Model and Notation (BPMN) has become the most widely accepted due to its comprehensive set of symbols and ability to effectively describe process behaviors [2]. However, BPMN's structural rules are documented in natural language, making automated validation difficult and requiring expert intervention for verification.

This research aims to enhance BPMN's metamodel by automatically translating its structural specifications into Object Constraint Language (OCL) constraints, enabling formal verification without manual validation [3, 4]. The proposed approach leverages Natural Language Processing (NLP) to extract syntactic rules from BPMN specifications, structure them systematically, and generate corresponding OCL constraints. The methodology involves extracting BPMN well-formedness rules using regular expressions, followed by tokenisation, part-of-speech (POS) tagging, and segmentation. A domain-specific glossary refines these outputs to facilitate the extraction of structural information. In the post-processing stage, a translation algorithm converts processed rules into OCL constraints, which are then integrated into a simplified BPMN metamodel. This enhanced metamodel ensures compliance

Joint Proceedings of the STAF 2025 Workshops: OCL, OOPSLE, LLM4SE, ICMM, AgileMDE, AI4DPS, and TTC. Koblenz, Germany, June 10–13, 2025.

*Corresponding author.
† All authors contributed equally to this work.
kkolahdooz2020@gmail.com (K. Kolahdouz); Shekoufeh.Rahimi@roehampton.ac.uk (S. K. Rahimi); kevin.lano@kcl.ac.uk (K. Lano)
https://pure.roehampton.ac.uk/portal/en/persons/shekoufeh-kolahdouz-rahimi (S. K. Rahimi)
0000-0002-0566-5429 (S. K. Rahimi); 0000-0002-9706-1410 (K. Lano)

with structural rules and improves the reliability of process modeling in both industry and academia. By embedding automated verification into Agile Model-Driven Engineering workflows, this research contributes to a more adaptable and efficient modeling process.

2. State of The Art

Correia and Abreu [5] manually extracted BPMN rules from the OMG document, formalised them with OCL, and integrated them into the BPMN metamodel for validation using the USE tool. While their approach covered the full BPMN specification, it required significant time, expertise, and manual conversion, making it prone to human error and inefficiencies. Following that they extended their previous work by adopting a model-driven approach for converting BPMN models into the USE tool and enhancing the BPMN metamodel with optimised modeling and OCL formalisation [6]. However, the approach still required expertise in writing OCL constraints, remained prone to human error, and lacked a final tool for practical use by BPMN modelers. Petri nets are a well-established approach with numerous verification methods and tools available. Many studies have used Petri nets, Colored Petri Nets, or other workflow languages to first translate BPMN models into a formal language and then verify them using existing tools. However, these languages do not fully support all standard BPMN elements [7]. Kherbouche et al.[8] proposed converting BPMN process models into finite-state models to analyse system behavior. They introduced LTL formulas for model checking to ensure structural correctness and detect errors. However, their approach only considers five temporal properties, and due to the complexity of LTL formulation and the lack of fully automated methods, its efficiency decreases as the number of properties increases.

3. The Process of Automatic Generation of OCL Constraints Using the Proposed Approach

This research leverages NLP to automate the extraction and formalisation of OCL constraints from BPMN models, which are documented in natural language, making validation challenging. Focusing on syntactic rules in Chapters 8 and 10 of the BPMN specification [2], the study narrows its scope to 228 pages. The extraction process applies NLP techniques such as tokenisation, parsing, and regular expressions to identify rule-defining sentences, supported by the NLTK library [9, 10] and a domain-specific glossary for accuracy. A rule-based approach then analyses sentence structures to generate formal OCL constraints [11], improving BPMN model verification and reducing manual effort.

3.1. Extracting Rules from the Text Using Regular Expressions

To design an appropriate regular expression (Regex) for extracting all rule-defining sentences from the text, it is first necessary to identify the syntactic structure and notation of these sentences within the document. Upon examining the selected text, it was observed that all rules are written as a bullet-pointed list using a specific bullet symbol (Figure 1). At this stage, a suitable extraction pattern must be designed by constructing a sequence of characters in the form of a regular expression. This pattern begins with the Unicode representation of the specific bullet symbol and ends with the character corresponding to a line break. Once the necessary text preprocessing is performed and the designed regular expression is refined, the selected text from the previous stage is processed. Each single-sentence or multi-sentence rule is then extracted and stored in a separate line within a text file, resulting in a total of 178 extracted rules. The extracted rules are classified into mandatory and optional categories based on the verbs used in the sentences. Verbs synonymous with 'must' indicate mandatory rules, while those similar to 'may' signify optional rules (Table 1). After identifying these verbs, the output text file from the previous stage is processed, and the classified rules are stored separately. Out of 178 extracted rules, 118 were mandatory, and 60 were optional. Since OCL constraints apply only to mandatory rules, the optional

- A **Gateway** MAY be a source of a **Sequence Flow**; it can have zero, one, or more *outgoing* **Sequence Flows**.
- A **Gateway** MUST have either multiple *incoming* **Sequence Flows** or multiple *outgoing* **Sequence Flows** (i.e., it MUST merge or split the flow).
  - A **Gateway** with a `gatewayDirection` of `unspecified` MAY have both multiple *incoming* and *outgoing* **Sequence Flows**.
  - A **Gateway** with a `gatewayDirection` of `mixed` MUST have both multiple *incoming* and *outgoing* **Sequence Flows**.
  - A **Gateway** with a `gatewayDirection` of `converging` MUST have multiple *incoming* **Sequence Flows**, but MUST NOT have multiple *outgoing* **Sequence Flows**.
  - A **Gateway** with a `gatewayDirection` of `diverging` MUST have multiple *outgoing* **Sequence Flows**, but MUST NOT have multiple *incoming* **Sequence Flows**.

Figure 1: An example of rules as a list of items

ones were discarded (Figure 2), and further processing was conducted solely on the mandatory rules file (Figure 3).

Table 1
Verb categorisation to distinguish mandatory and optional rules.

Kind of Verb	Verbs in Selected Text
Mandatory	MUST, SHALL, SHOULD
Optional	MAY, MAY NOT, RECOMMENDED, OPTIONAL

Figure 2: A section of the text file of removed optional rules.

3.2. Preprocessing Stage of Rules

The preprocessing stage of rule extraction was conducted using NLTK, chosen for its efficiency in tokenisation and part-of-speech tagging [10]. While other NLP tools like Stanford Core NLP offer advanced capabilities, they require more computational resources and longer processing times. Given the need for accuracy and efficiency, NLTK was preferred for this study. The preprocessing involved sentence

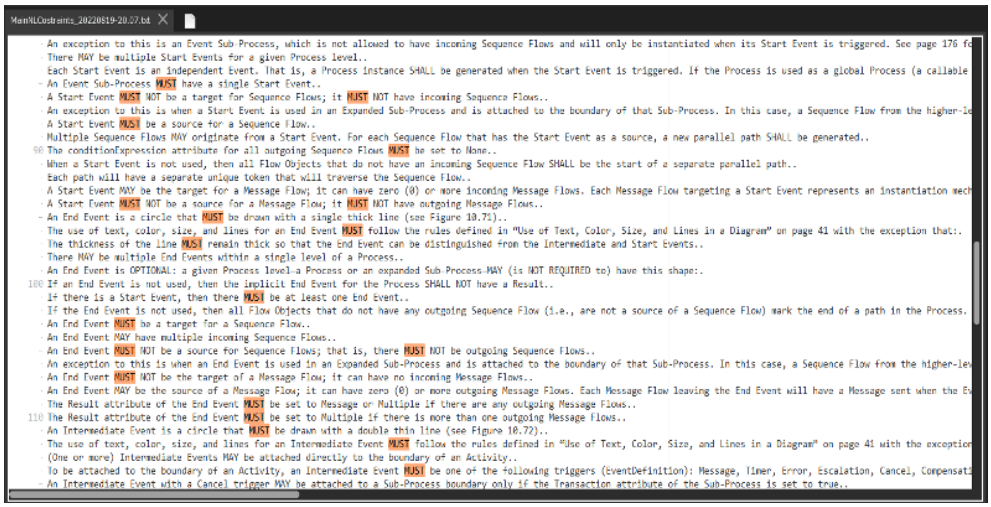

Figure 3: A section of the text file of extracted mandatory rules.

boundary detection, tokenisation, part-of-speech tagging, and segmentation, applied sequentially to extracted mandatory rules.

- **Sentence Boundary Detection:** The first step in breaking the text into smaller sentences aims to simplify text analysis for subsequent stages. In this research, sentence boundaries are defined based on the completion of each rule. The output of this stage consists of a set of distinct rules, each of which may contain either a single sentence (Figure 4) or multiple sentences (Figure 5). For the sake of clarity in this study, the term "sentence" will refer to either a single sentence or a set of multiple sentences that together form a complete rule. This distinction ensures a consistent understanding of the extracted rules throughout the research process.

A Gateway with a gatewayDirection of mixed MUST have both multiple incoming and outgoing Sequence Flows.

Figure 4: Single-sentence rule

If the inline Event Sub-Process completes without "rethrowing" the Event, the Activity is considered to have completed and normal Sequence Flow resumes. In other terms, the Event Sub-Process "absorbs" the Event..

Figure 5: Multi-sentence rule

- **Tokenisation:** Refers to breaking each sentence into a set of tokens, including words and punctuation marks. Figure 6 illustrates the output of the tokenisation step applied to the selected extracted rule. The output of this step serves as the input for subsequent processing stages. Based on the separated tokens generated in this stage, POS tagging is performed in the next step.
- **POS Tagging**: Is a process in traditional grammar that categorises words based on their syntactic roles, such as nouns, verbs, and adjectives. During this phase, each word in the text is analysed and assigned an appropriate tag according to its grammatical function. To enhance understanding, a selection of these tags, along with explanations and examples, is provided in Table 2. Figure 7 also illustrates the output of this stage.

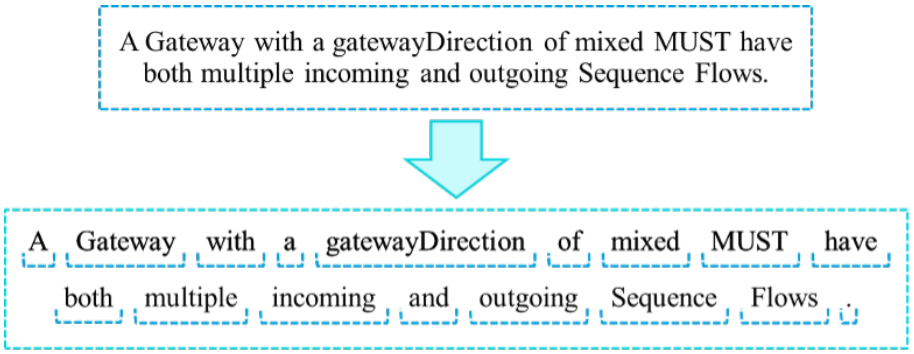

Figure 6: Output of Tokenisation

Table 2
Categorisation of default POS tagging labels

Tag	Description	Tag	Description
CC	Coordain, Conjunction	SYM	Symbol
CD	Cardinal number	TO	"to"
DT	Determiner	UH	Interjection
EX	Existential 'there'	VB	Verb, base form
FW	Foreign word	VBD	Verb, past tense
IN	Preposition/sub-conj	VBG	Verb gerund
JJ	Adjective	VBN	Verb, past participle
JJR	Adj., comparative	VBP	Verb non-3sg pres
JJS	Adj., superlative	VBZ	Verb, 3sg pres
LS	List item marker	WDT	Wh-determiner
MD	Modal	WP	Wh-pronoun
NN	Noun, sign. or mass	WP$	Possesive wh-
NNS	Noun, plural	WRB	Wh-adverb
NNP	Proper noun, singular	$	Dollar sign
NNPS	Proper noun, plural	#	Pound sign
PDT	Predeterminer	"	
POS	Possessive ending	"	Right quote
PRP$	Possessive pronoun	)	Left parenthesis
RB	Adverb	(	Right parenthesis
RBR	Adverb, comparative	,	Comma
RBS	Adverb, superlative	.	Sentence-final punc
RP	Particle	:	Mid-sentence punc

- **Segmentation:** The main goal of this process is to identify groups of words that form a sentence. In other words, segmentation helps divide each sentence into its subcomponents, which can be groups of words or symbols. The output of this stage can be seen in Figure 8.
  After completing these four stages and before proceeding to the post-processing phase, it is necessary to customise and update some of the assigned tags from the POS tagging stage using a specialised vocabulary list. However, since this vocabulary list is utilised in the sentence structure analysis stage, the details regarding its design will be provided in the next section.

3.3. Post-Processing Stage and Proposed Algorithm Development

This stage represents the most critical part of the proposed approach and constitutes the primary innovation of this research. Given that the proposed algorithm translates natural language rules into

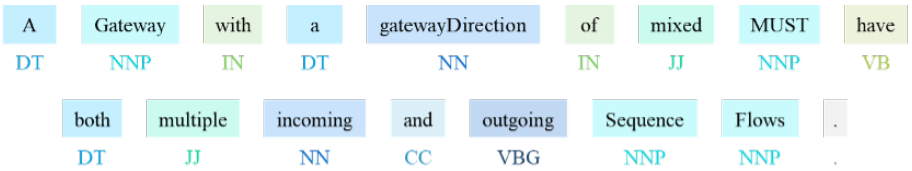

Figure 7: Output of the POS Tagging Stage

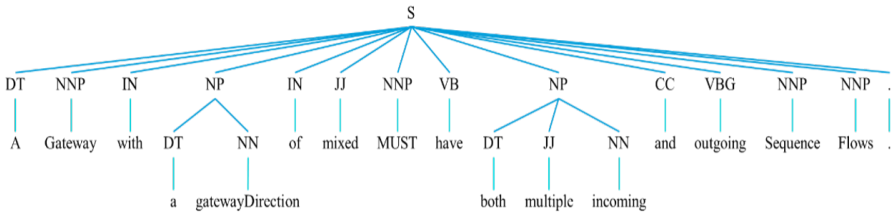

Figure 8: Output of the Segmentation Stage

Table 3
The designed glossary for this research

glossary lists	Label	Samples
Element's appearance	EAP	draw, line, size, shape, square, circle, diamond, color
BPMN elements	BPE	Start Event, End Event, Gateway, Sequence Flow, Lane, Pool
Metamodel Enummerations	ENUM	ItemKind, ProcessType, GatewayDirection, EventBasedGatewayType
Obligatory terms	OT	MUST, MUST NOT, MAY, MAY NOT, Optional, Recommended

OCL constraints, this phase focuses on determining the overall structure of sentences using the assigned tags for each token. Consequently, the necessary information is extracted from each sentence. This extracted information is then appropriately placed within the syntactic structure of OCL constraints, thereby completing the automation process for generating OCL constraints from natural language rules. In the following, this process will be explained.

- **Sentence Structure Analysis**: This research identifies sentence structures by analysing patterns among tokens and their assigned tags. Each pattern represents a meaningful sequence of tags, providing a roadmap to locate target information. The accuracy of this identification process heavily relies on the assigned tags, making the tagging strategy a crucial parameter. To enhance sentence structure analysis, a domain-specific glossary is employed. This glossary consists of categorised lists, each containing terms with shared functions, which are assigned personalised tags [12]. These tags facilitate the quick identification of relevant terms within sentences, improving both structure determination and information extraction. The composition of these lists is tailored to the research objectives and may vary accordingly. Table 3 presents the glossary developed for this study. The first category focuses on filtering extracted rules by removing those unrelated to BPMN syntax verification, such as notation rules that specify visual representations of elements. This step, labeled EAP, refines the dataset before further processing. As a result, 41 out of 118 mandatory rules were eliminated, leaving 77 final rules.
  To accurately generate OCL constraints, it is crucial to identify the relevant components within the BPMN metamodel. Lists one and two are designated for recognising classes and enumerations, respectively. Given the extensive number of these elements, regular expressions were employed for automated extraction. Additionally, compound sentences containing both mandatory and optional verbs required precise verb labeling to enhance sentence structure detection. Figure 9 illustrates the updated POS labels, revealing recurring token placement patterns in the extracted rules.
  A single structural pattern was sufficient for this study, as it remained consistent across all analysed rules with only minor variations, which were effectively managed by the designed algorithm. The semi-formal

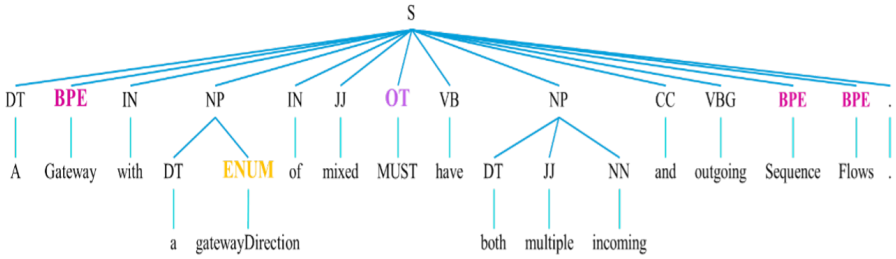

Figure 9: Updating part-of-speech tags based on glossary lists

Table 4
Categorisation of operations used in OCL constraints

	Operations used on collections
1	size()
2	count(object)
3	includes/includesAll(object) excludes/excludesAll(object)
4	isEmpty/notEmpty()

and structured nature of the BPMN specification document ensures that identifying one rule's pattern aids in recognising others. Moreover, since this research focuses on verifying BPMN process models, the BPMN 2.0 metamodel was simplified by retaining its core structures, while omitting non-essential elements. As a result, the extracted rules share structural features, allowing a unified approach to pattern identification.

- **Extraction of Target Information and Rule Generation for OCL Expressions**:
To refine the pattern identified in the previous stage, the equivalent OCL constraint for the motivational example is first examined. This analysis clarifies the essential information needed to construct the constraint and its extraction points within the rule, shaping the structure of the information extraction algorithm. The algorithm simultaneously identifies the information and maps it to the appropriate OCL syntax, generating OCL expressions accordingly. The identification of target information and the corresponding OCL constraint structure is guided by an understanding of the BPMN metamodel and OCL syntax. Sentence structures are categorised based on standard OCL functions and operations for collections, ensuring a uniform constraint structure with only minor variations. Table 4 presents this categorisation, which facilitates accurate information extraction, often requiring multiple categories to analyse a single rule comprehensively.
To better understand the process of writing an OCL constraint, the position of the element within the BPMN metamodel structure must first be identified. In this research, the design of equivalent constraints has been carried out based on the researcher's understanding of the metamodel. Here, the part of the metamodel is presented in Figure 10. Based on the displayed section of the metamodel, the equivalent constraint for the rule should be written as shown in Figure 11.
The process of extracting target information and integrating it into an OCL constraint is demonstrated through an example, clarifying the logic behind automated constraint generation. Figure 12 illustrates the required information based on the equivalent OCL constraint syntax. An algorithm is designed to extract target information from sentences based on their position, order, and assigned labels, ensuring proper placement within the equivalent OCL constraint framework. The logic governing this extraction and mapping process is outlined as follows.
  - The first word or set of words in each rule specifies the context for the corresponding OCL constraint.
  - If a word with the label ENUM exists, the value assigned to this attribute in the sentence must be found, and the corresponding OCL expression should be written according to the correct syntax.
  - If numbers or adjectives indicating quantity are used, the use of the size() or count() operations must be specified, and the equivalent numeric value should be added with the appropriate syntax.
  - The use of Boolean expression operations is also determined based on the equivalent words for them in the text. Due to the consistent structure of the rules throughout the document, the number of such words is very limited.

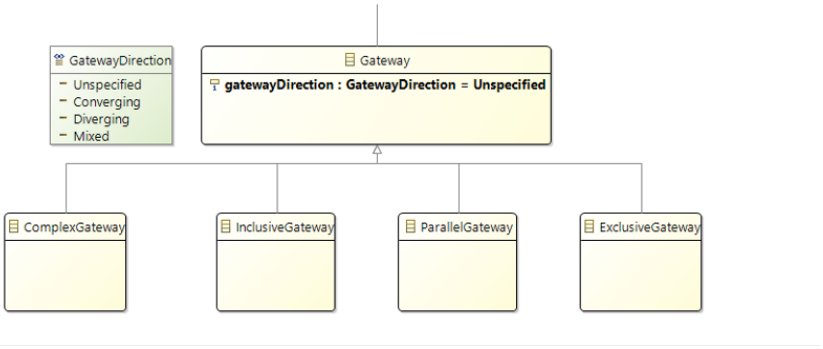

Figure 10: Metamodel of the motivated example

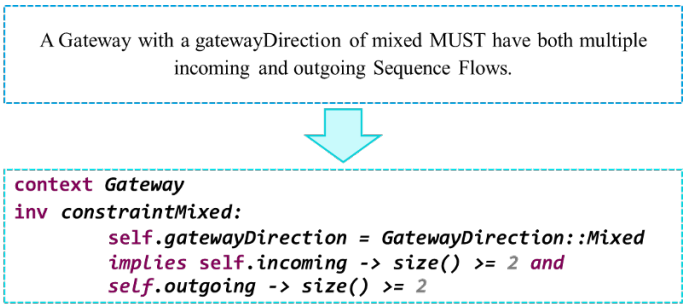

Figure 11: The corresponding OCL constraint for the extracted rule

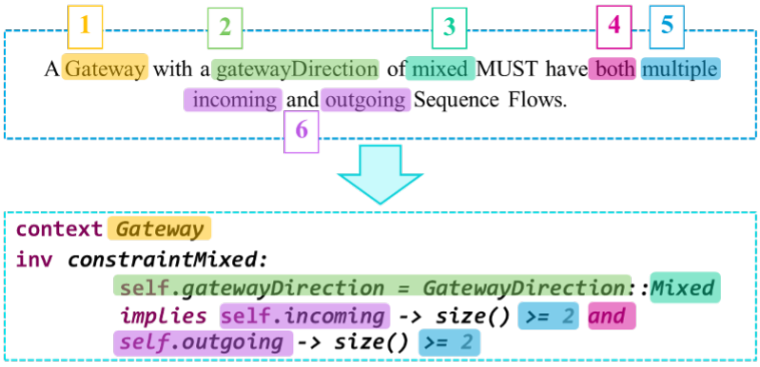

Figure 12: Sample of extracted target information

The pseudocode for this section of the algorithm is presented in Figure 13.

4. Evaluation

Evaluation of the proposed algorithm in terms of sentence structure coverage is investigated in this part. The final output of the proposed algorithm is presented in two formats: a comprehensive table containing all generated constraints in an Excel file and separate text files categorised based on constraint domains. An analysis of the Excel file indicates that the algorithm successfully generated 65 constraints out of a total of 77 mandatory rules, demonstrating an 84.4% coverage in identifying rule structures and generating the corresponding OCL constraints. A sample of the generated constraints is illustrated

```
function FIND_COTEXT
    //search among sentence tokens
    if token tag is "BPE"
        context <- first sequence of "BPE"s

function SET_ATTRIBUTE
    //search among sentence tokens
    if there is one capital letter in token
    attribute <- select that token
    new_tag <- "ATTR"
    if the token tag is "ENUM"
    enum_ref <- Captalize first letter of token

    //check next token after "ENUM"
    if next token is "of"
    enum_value <- next NN or JJ

    //check token before "ATTR"
    if the tokenis about multiplicity
    size <- defined number of multiplicity
```

Figure 13: Pseudo-code of a part of the proposed algorithm

in Figure 14. The syntactic and semantic validity of these constraints will be further examined in the subsequent evaluation of this study. The columns in this figure are labeled according to the table

	OCL_Constraints	MyTags	Chunks	POSTags	Tokens	NLExtractedRules	Column1	
32	context EndEventinv:	[('If', 'IN'), ('th	(S If/IN (NP	[('If', 'IN'), ('the',	['If', 'the', 'En	If the End Event is not use	32	34
33	context EndEventinv: sel	[('An', 'DT'), (	(S (NP An/D	[('An', 'DT'), ('Er	['An', 'End', 'E	An End Event MUST be a	33	35
34	context EndEventinv: sel	[('An', 'DT'), (	(S (NP An/D	[('An', 'DT'), ('Er	['An', 'End', 'E	An End Event MUST NOT	34	36
35	context IntermediateEver	[('An', 'DT'), (	(S (NP An/D	[('An', 'DT'), ('Er	['An', 'End', 'E	An End Event MUST NOT	35	37
36	context IntermediateEver	[('The', 'DT'), (	S The/DT	[('The', 'DT'), ('F	['The', 'Resu	The Result attribute of the	36	38
37	context IntermediateEver	[('The', 'DT'), (	S The/DT	[('The', 'DT'), ('F	['The', 'Resu	The Result attribute of the	37	39
38	context IntermediateEver	[('To', 'TO'), (	S To/TO b	[('To', 'TO'), ('be	['To', 'be', 'at	To be attached to the bou	38	40
39	context IntermediateEver	[('If', 'IN'), ('th	(S If/IN the/	[('If', 'IN'), ('the',	['If', 'the', 'Inte	If the Intermediate Event is	39	41
40	context IntermediateEver	[('The', 'DT'), (	S The/DT	[('The', 'DT'), ('I	['The', 'Intern	The Intermediate Event M	40	42
41	context IntermediateEver	[('The', 'DT'), (	S The/DT	[('The', 'DT'), ('I	['The', 'Intern	The Intermediate Event M	41	43
42	context IntermediateEver	[('If', 'IN'), ('th	(S If/IN the/	[('If', 'IN'), ('the',	['If', 'the', 'Inte	If the Intermediate Event is	42	44
43	context IntermediateEver	[('Intermediat	(S Intermedi	['Intermediate',	['Intermediat	Intermediate Events MUS	43	45
44	context IntermediateEver	[('An', 'DT'), (	(S (NP An/D	[('An', 'DT'), ('In	['An', 'Interme	An Intermediate Event MU	44	46
45	context LinkIntermediate	[('An', 'DT'), (	(S (NP An/D	[('An', 'DT'), ('ex	['An', 'except	An exception to this: a sou	45	47
46	context LinkIntermediate	[('A', 'DT'), ('L	(S A/DT Lin	[('A', 'DT'), ('Link	['A', 'Link', 'In	A Link Intermediate Event	46	48
47	context EventSub-Proce	[('If', 'IN'), ('th	(S If/IN ther	[('If', 'IN'), ('there	['If', 'there', 'is	If there is a source Link, th	47	49
48	context EventSub-Proce	[('There', 'EX	(S There/E>	[('There', 'EX'),	['There', 'MU	There MUST NOT be mu	48	50
49	context EventSub-Proce	[('If', 'IN'), ('th	(S If/IN the/	[('If', 'IN'), ('the',	['If', 'the', 'inli	If the inline Event Sub-Pro	49	51
50	context EventSub-Proce	[('If', 'IN'), ('th	(S If/IN the/	[('If', 'IN'), ('the',	['If', 'the', 'inli	If the inline Event Sub-Pro	50	52
51	context Gatewayinv: if se	[('If', 'IN'), ('th	(S If/IN the/	[('If', 'IN'), ('the',	['If', 'the', 'Ga	If the Gateway does not h	51	53
52	context Gatewayinv: if se	[('A', 'DT'), ('(	(S A/DT Ga	[('A', 'DT'), ('Ga	['A', 'Gatewa	A Gateway MUST have e	52	54
53	context Gatewayinv: if se	[('A', 'DT'), ('(	(S A/DT Ga	[('A', 'DT'), ('Ga	['A', 'Gatewa	A Gateway with a gatewa	53	55
54	context Gatewayinv: if se	[('A', 'DT'), ('(	(S A/DT Ga	[('A', 'DT'), ('Ga	['A', 'Gatewa	A Gateway with a gatewa	54	56
55	context Gatewayinv: if se	[('A', 'DT'), ('(	(S A/DT Ga	[('A', 'DT'), ('Ga	['A', 'Gatewa	A Gateway with a gatewa	55	57

Figure 14: Final table of constraints generated by the algorithm

structure defined in the proposed algorithm (Figure 15). Each row presents an extracted mandatory rule, outputs from preprocessing stages (tokenisation, POS tagging, segmentation), the refined POS tags, and the corresponding OCL constraint.

Out[124]:

	NLExtractedRules	Tokens	POSTags	Chunks	MyTags	OCL_Constraints
0	If the Activity does not have an incoming Sequ...	[If, the, Activity, does, not, have, an, incom...	[(If, IN), (the, DT), (Activity, NN), (does, V...	[(If, IN), [(the, DT), (Activity, NN)], (does,...	[(If, IN), (the, DT), (Activity, BPE), (does, ...	
1	There are two exceptions to this: Compensation...	[There, are, two, exceptions, to, this, :, Com...	[(There, EX), (are, VBP), (two, CD), (exceptio...	[(There, EX), (are, VBP), (two, CD), (exceptio...	[(There, EX), (are, VBP), (two, CD), (exceptio...	
2	If the Activity does not have an outgoing Sequ...	[If, the, Activity, does, not, have, an, outgo...	[(If, IN), (the, DT), (Activity, NN), (does, V...	[(If, IN), [(the, DT), (Activity, NN)], (does,...	[(If, IN), (the, DT), (Activity, BPE), (does, ...	
3	There are two exceptions to this: Compensation...	[There, are, two, exceptions, to, this, :, Com...	[(There, EX), (are, VBP), (two, CD), (exceptio...	[(There, EX), (are, VBP), (two, CD), (exceptio...	[(There, EX), (are, VBP), (two, CD), (exceptio...	
4	If the multi-instance instances are set to be ...	[If, the, multi-instance, instances, are, set,...	[(If, IN), (the, DT), (multi-instance, NN), (i...	[(If, IN), [(the, DT), (multi-instance, NN)], ...	[(If, IN), (the, DT), (multi-instance, NN), (i...	

Figure 15: Designed table in the proposed algorithm

The OCL constraints' syntactical correctness was verified using the CompleteOCL editor in the Eclipse IDE, chosen for its integration with the BPMN metamodel development. The extracted rules were reviewed, and incompatible constraints were removed, leaving 46 valid rules for 17 metaclasses. These rules were consolidated into a single OCL file using NLP and loaded onto the BPMN metamodel. After validation with CompleteOCL, no syntactical errors were found, confirming the correctness of the constraints.

4.1. Conclusion

In this paper, the proposed approach was examined in detail, outlining its structured pipeline for automating the generation of OCL constraints. The process began with the extraction of syntactic rules from the BPMN specification document, followed by natural language preprocessing using the NLTK tool. Finally, the core contribution of this research, an algorithm for translating informal syntactic rules into formal OCL constraints, was developed and analysed using an example. This approach contributes to Agile Model-Driven Engineering by facilitating automation in constraint generation, enabling seamless integration of formal verification within iterative development cycles. By reducing manual effort and ensuring consistency in model validation, it supports the dynamic and evolving nature of model-driven processes.

Declaration on Generative AI

The authors used ChatGPT (GPT-5) for text editing and refinement. All outputs were reviewed and validated by the authors, who take full responsibility. No proprietary or third-party confidential data were provided to these tools.

References

[1] E. Foster, B. Towle Jr, Software engineering: a methodical approach, Auerbach Publications, 2021.

[2] O. M. Group, Omg business process model and notation revision task force, Available at: https://www.omg.org/spec/BPMN/2.0.2/About-BPMN, 2013.

[3] J. B. Warmer, A. G. Kleppe, The object constraint language: getting your models ready for MDA, Addison-Wesley Professional, 2003.

[4] O. M. Group, Object Constraint Language, version 2.4 ed., 2014. URL: https://www.omg.org/spec/OCL/2.4/PDF.

[5] A. Correia, F. B. e Abreu, Adding preciseness to bpmn models, Procedia Technology 5 (2012) 407–417.

[6] A. Correia, F. B. e Abreu, Enhancing the correctness of bpmn models, in: Sustainable Business: Concepts, Methodologies, Tools, and Applications, IGI Global, 2020, pp. 373–394.

[7] J. Kasse, Supporting compliance verification for collaborative business processes., Ph.D. thesis, Bournemouth University, 2019.

[8] O. M. Kherbouche, A. Ahmad, H. Basson, Using model checking to control the structural errors in bpmn models, in: IEEE 7th International Conference on Research Challenges in Information Science (RCIS), IEEE, 2013, pp. 1–12.

[9] P. M. Nadkarni, L. Ohno-Machado, W. W. Chapman, Natural language processing: an introduction, Journal of the American Medical Informatics Association 18 (2011) 544–551.

[10] E. Loper, S. Bird, Nltk: The natural language toolkit, arXiv preprint cs/0205028 (2002).

[11] K. Chowdhary, Natural language processing, MBM Engineering College, Jodhpur, India, lecture notes (2012).

[12] J. P. Kasse, L. Xu, P. De Vrieze, A comparative assessment of collaborative business process verification approaches, in: Working Conference on Virtual Enterprises, Springer, 2017, pp. 355–367.