

From Natural Language to Domain-Specific Languages – A Use Case of Xtext Platform



By:

Amina Zafar

(Registration No.: MS-CSE-21-361404)

Supervisor:

Dr. Farooque Azam

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING,
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING,
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,
ISLAMABAD

December 20, 2023

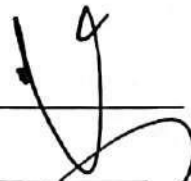
THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis written by NS Amlna Zafar Registration No. 00000361404, of College of E&ME has been vetted by undersigned, found complete in all respects as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS/MPhil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the thesis.

Signature : _____ 

Name of Supervisor: **Dr Farooque Azam**

Date: 20-12-2023

Signature of HOD: _____ 

(Dr Usman Qamar)

Date: 20-12-2023

Signature of Dean: _____ 

(Brig Dr Nasir Rashid)

Date: 20 DEC 2023

From Natural Language to Domain-Specific Languages – A Use Case of Xtext Platform

By
Amina Zafar
(Registration No.: 00000361404)

A thesis submitted to National University of Sciences and Technology,
Islamabad
in partial fulfillment of the requirements for the degree of
Master of Sciences in Software Engineering

Supervisor
Dr. Farooque Azam

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING,
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING,
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY
ISLAMABAD
December 20, 2023

*Dedicated to my mother, whose
tremendous continuous support and
endless prayers led me to this
accomplishment.*

Acknowledgment

First and foremost, I would like to express my gratitude to Allah Almighty, the most merciful and the most kind, for bestowing His blessings upon me and granting me the strength to complete this work.

I extend my heartfelt appreciation to the Higher Education Commission (HEC) Pakistan for funding this research project under the MRED project series.

I am deeply indebted to my supervisor, Brig. Dr. Farooque Azam, for his unwavering intellectual support and valuable ideas throughout this research journey. He has been an inspirational instructor, providing me with technical guidance and moral encouragement. His insightful feedback has honed my skills and critical thinking, enabling me to achieve my research goals. Without his continuous support, completing this dissertation would not have been possible. I am immensely grateful for the motivation he has instilled in me throughout this challenging expedition. Concluding my research study under his supervision has been a great honor.

I would also like to express my appreciation to my external GEC member, Dr. M. Waseem Anwar, for his valuable suggestions and guidance. I am thankful to Ms. Afshan Latif for their assistance and direction in helping me achieve my research objectives.

I dedicate this work to my supervisor, mentor, and teacher, Brig. Dr. Farooque Azam, and my beloved mother, Aisha Nazir, who has supported and stood by me unwaveringly throughout this journey. Without her love and encouragement, I would not have been able to accomplish this work. To my dear "Mama," thank you for everything. I also extend my gratitude to my sister for her support.

Abstract

Development of the domain-specific language (DSL), i.e., the Xtext framework, supports the development of consistent requirements of software systems, but their development complexities are crucial in MDE. At the core of the Xtext framework, we find its grammar, which requires propitious knowledge regarding its technical concepts. The software development life cycle (SDLC) involves a complex requirement elicitation phase that entails collaboration among multiple stakeholders. The problem arises when non-technical stakeholders encounter diverse challenges to grasp the development intricacies of the Xtext grammar. Hence, they cannot communicate their requirements to technical stakeholders. Consequently, there is a need for such a framework that can simplify the DSL development of the Xtext to facilitate collaboration among multiple stakeholders. An extensive analysis of 44 prior studies is conducted related to both NLP techniques and MDE approaches. It is analyzed that a few of the existing studies have focused on modeling with NLP techniques and the Xtext framework to support the generation of consistent requirements. However, it is important to mention that a prominent research gap still exists, as a framework for auto-generated Xtext grammar is not proposed.

Therefore, this thesis presents a research work where a framework is developed to automatically generate the Xtext grammar from the natural language requirements using natural language processing (NLP) techniques. Particularly, a rule-based approach is incorporated to extract the primary DSL elements comprising the Xtext, such as the root element, relationship element, and attributes from the textual requirements. Furthermore, a comprehensive algorithm is devised to systematically apply the NLP rules, facilitating the generation of desired Xtext grammar. Based on this approach, the tool Natural-Language To Domain-Specific Language (NL2DSL) is developed. The proposed approach is validated through two case studies, i.e., the timing model and the diabetic manager. Our generated results prove that the proposed framework generates the Xtext grammar from the textual requirements with a satisfactory degree of accuracy.

Keywords: Domain-Specific Languages, DSLs, Xtext, East-ADL, model-driven engineering, Natural Language Processing.

Contents

Chapter 1: Introduction	1
Background	1
DSL environment	2
Importance of Collaboration in Requirements Engineering	2
Importance of automation for the modeling environment	4
Natural Language Processing (NLP)	4
Goals and Objectives	5
Motivation	6
Problem	6
Proposed Solution	7
Thesis Organization	9
Chapter 2: Preliminaries	10
Literature Review	10
Natural Language Processing in MDE	11
Natural Language Processing in UML	14
Natural Language Processing in SysML	17
Natural Language Processing in DSM	19
Xtext in MDE	21
Blended Modeling	21
Verification/Validation	22
Domain's Perspective	23
Artificial Intelligence	25
Research Gap	26
Contributions	27
Chapter 3: Methodology	28
Proposed Algorithm	29
Transformation Rules For the Identification of Xtext Elements	35
Chapter 4: Implementation	57
Tools and Languages	57
Tool Interface	59
Xtext Grammar Generation Details	60
Chapter 5: Validation	62
Dataset Collection	62
Case Study 01	63
Case Study 02	64
Results	64

Results of Case Study 01	65
Results of Case Study 02	72
Chapter 6: Discussion and Limitations	78
Discussions	78
Limitations	80
Chapter 7: Conclusion and Future Directions	81
7.1 Conclusion.....	81
References	

List of Figures

Figure 1.1	Research Review	8
Figure 1.2	Overview of the Research Study	8
Figure 3.1	High Level of the Proposed Framework.	29
Figure 3.2	Proposed Algorithm.....	30
Figure 3.3	Graphical Description of Rules for the Root Element Identification.	38
Figure 3.4	Graphical Description of Rules for the Relationship Element Identification.	40
Figure 3.5	Graphical Description of Rules for the Multiple Attribute Element Identification.....	47
Figure 3.6	Graphical Description of Rules for the Multiple Attribute Element Identification.....	50
Figure 3.7	Graphical representation of rules for Single Attribute Element Identification.	52
Figure 3.8	Graphical Description of Rules for Single & Optional Attribute Element Identification.....	53
Figure 4.1	Interface of the Eclipse Platform.	58
Figure 4.2	Interface of the NL2DSL Tool.	59
Figure 4.3	NL2DSL's Output comprising Xtext Grammar.....	60
Figure 4.4	Generated Xtext Grammar Save in .xtext.....	61
Figure 5.1	Timing Model Case Study	63
Figure 5.2	Diabetic Manager Case Study	64
Figure 5.3	Generated Xtext Timing Model Grammar	68
Figure 5.4	Generated Xtext Diabetic Manager	72

List of Tables

Table 1.1	Overview of primary DSL elements comprising the Xtext Framework.	3
Table 2.1	Natural Language Processing (NLP) in MDE.....	12
Table 2.2	Natural Language Processing (NLP) in UML.....	15
Table 2.3	Natural Language Processing (NLP) in SysML.....	18
Table 2.4	Natural Language Processing (NLP) in DSM.....	19
Table 2.5	Blended Modeling in Xtext.....	22
Table 2.6	Verification with Xtext.	23
Table 2.7	Domains Perspectives within Xtext.....	24
Table 2.8	AI within Xtext.....	25
Table 2.9	Overview of the Limitations analyzed from the Literature Review	26
Table 3.1	List of POS Tags.....	36
Table 5.1	List of Actual Xtext DSL Elements for Timing Model Case Study	66
Table 5.2	List of Xtext DSL Elements for Timing Model Case Study Identified by NL2DSL tool.	67
Table 5.3	Calculation of NL2DSL Effectiveness for Timing Model Case Study.	68
Table 5.4	List of Actual Xtext DSL Elements for Variated Timing Model Case Study	69
Table 5.5	List of Xtext DSL Elements for Variated Timing Model Case Study Identified by NL2DSL tool.	70
Table 5.6	Calculation of NL2DSL Effectiveness for Variated Timing Model Case Study.....	71
Table 5.7	List of Actual Xtext DSL Elements for the Diabetic Manager Case Study	73
Table 5.8	List of Xtext DSL Elements for the Diabetic Manager Case Study Identified by NL2DSL Tool.	74
Table 5.9	Calculation of NL2DSL Effectiveness for Diabetic Manager Case Study	74
Table 5.10	List of Actual Xtext DSL Elements for Variated Diabetic Manager Case Study.....	75
Table 5.11	List of Xtext DSL Elements for Variated Diabetic Manager Case Study Identified by the NL2DSL tool.	76
Table 5.12	Calculation of NL2DSL Effectiveness for variated Diabetic Manager Case Study	76

Chapter 1

Introduction

The primary purpose of this chapter is to give an overview of the terms utilized throughout this thesis. It is organized into six sections. **Section 1.1** presents the background study of the concepts used throughout this research work. **Section 1.2** presents the goals and objectives of our research study. The motivation behind our research study is presented in **Section 1.3**. The problem statement of our research study and the proposed solution are presented in **Section 1.4** and **Section 1.5**, respectively. **Section 1.6** presents the thesis organization.

Background

This section presents a background study of our research study by exploring which concepts have been utilized. The concepts include:

- DSL Environment
- Importance of Collaboration in Requirement Engineering
- Importance of automation in the modeling environment
- Natural Language Processing (NLP)

DSL environment

Model-driven engineering offers two modeling approaches, i) textual modeling ii) graphical modeling. Throughout this thesis, our research work focuses on the textual modeling approach provided by different frameworks, including Xtext. Particularly, Xtext is a domain-specific modeling (DSL) framework that provides a textual model with reliance on keywords and syntax of the programming language. It is powered by the Itemis AG, released in 2008 under the Eclipse Public License. Xtext framework aims to develop consistent requirements to support complex systems development. It is vital in enhancing usability and imposing validation checks to ensure the requirement's consistency [1].

It is comprised of two components, i) DSL Definition ii) Runtime behavior. Particularly, the DSL definition permits the design of customized Xtext grammar. On the contrary, the runtime behavior assists in the validation of the designed Xtext grammar. Our research study focuses on the DSL definition of the Xtext framework. Table 1.1 presents an overview of the primary DSL elements of the Xtext.

Importance of Collaboration in Requirements Engineering

The requirement engineering is the initial phase of the software development life cycle (SDLC). Moreover, requirement elicitation is a critical phase to elicit the entire requirements of the software system being developed [2]. The requirements are elicited through collaboration among multiple stakeholders. The stakeholders are classified into two categories: i) technical stakeholders and ii) non-technical stakeholders. The requirements elicitation is influenced by the desired needs and expectations, along with each stakeholder's experience that must be satisfied. A detailed understanding of the requirements is required to develop an efficient software system. It is observed that both categories of stakeholders hold distinct perceptions in the model-based development of software systems. Like, technical stakeholders focus on technical perceptions, whereas non-technical stakeholders lack technical expertise and merely focus on the business requirements.

Table 1.1: Overview of primary DSL elements comprising the Xtext Framework.

Sr.#	Primary Element	Sub-Element	Representation in Xtext
1	Root Element	Parser Rule Keyword	ParserRule appears with a colon followed by a keyword.
2	Relationship Element	Parser Rule Keyword	ParserRule appears with a colon followed by a keyword.
		Association Name	Containment Relationship (Association Name, Association Operator, Child Element, Association Constraint)
		Child Element	
		Association Operator	Reference Relationship (Association Name, Association Operator, Cross Reference (Child Element), Association Constraint)
Association Constraint			
3	Attributes	Parser Rule Keyword	ParserRule appears with a colon followed by a keyword.
		Attribute Name	Multiple Attributes Attribute1 Name = Data Type Attribute2 Name = Data Type Or (Attribute1 Name? = Keyword)? (Attribute2 Name? = Keyword)?
		Data Type Or Keyword	Single Attributes Attribute Name = Data Type (Attribute Name? = Keyword)?
			Optional Attributes Or (Attribute Name = Data Type)?

Due to this technical barrier, the communication gap is introduced between the technical and non-technical stakeholders, which impacts the quality of requirements [3]. It is estimated that 56% of the failures in system development are caused by poor communication, which requires high cost and 86% of the staff time for correction [4]. Therefore, a collaborative environment should be developed by devising the identical language among both stakeholder categories [5]. Thus, the technical stakeholders can elicit accurate requirements through effective collaboration with non-technical stakeholders. Several organizations adapt the collaborative environments in their workplace to support the high-quality standard of software systems. Such a collaborative environment supports a variety of advantages, such as system development with non-conflicted requirements, proper stake-

holder engagement, and fewer communication failures.

Importance of automation for the modeling environment

Development of the Domain-specific languages (DSLs) through the Xtext framework is crucial in model-driven engineering. Automation is a process of generating models from the textual requirements. Currently, the manual development of the domain-specific language using the Xtext framework is conducted in various ways, such as blended modeling. Particularly, the manual development of DSLs through the Xtext framework makes it difficult to grasp the technical concepts by the non-technical stakeholders.

In contrast, the technical stakeholders can understand the technical concepts of the Xtext grammar. Due to such complex scenarios, misunderstandings and conflicting requirements between these two categories of stakeholders occur, which can impact their collaboration environment. With the occurrence of these challenges, an auto-generated Xtext grammar is required. To employ this automation process, artificial intelligence technologies such as natural language processing can be applied to auto-generate the Xtext grammar.

Natural Language Processing (NLP)

Natural language processing (NLP) is a branch of artificial intelligence that can comprehend the structure and syntax of human language. Generally, it takes textual data, processes the unstructured text, and then extracts the key information [6]. NLP is divided into two main phases, described in the below sub-sections.

- i) **Data Preprocessing** Data preprocessing is an integral and initial phase of NLP that removes unnecessary information from the unstructured text to create the structured textual dataset. Some techniques, like stemming, lemmatization, etc., can be used based on customized preferences.
- ii) **Algorithm application** Results of the data preprocessing are attained to execute the

applied algorithm. The applied algorithm extracts the desired information from the preprocessed results. The algorithms are classified as machine learning classifiers and customized algorithms with the rule-based approach of NLP, etc.

- (a) **Advanced Artificial Intelligence** Some advanced artificial intelligence techniques, such as Machine learning and deep learning, are utilized. This sub-field of artificial intelligence consists of multiple algorithms such as Random Forest, Naive Bayes, etc. Further, deep learning (DL) comprises different algorithms like CNN (convolutional neural network), which can classify textual requirements. These algorithms attained the results of the preprocessed dataset to identify the desired elements later used for the categories classification.
- (b) **Rule-Based Approach** In the initial days of natural language processing, the rule-based approach was widely utilized, and it is still used today. It is a practical approach that comprises a set of regular expressions or heuristic rules that can apply to the textual dataset, extracting the desired information. In a comparison view, the rule-based approach is straightforward, and the advanced algorithms of Machine learning or deep learning require training the large textual datasets, which is a bit complex.

Goals and Objectives

The main objective of this research study is to support a collaborative environment among diverse stakeholders. The Xtext framework is comprised of various technical concepts conforming to the creation of Xtext grammar. Due to the development complexity of the Xtext grammar, its technical concepts appear different for the technical and non-technical stakeholders. The non-technical stakeholders have an inproficient understanding of the Xtext framework. On the other hand, the technical stakeholders can easily comprehend the technical concepts of Xtext grammar.

Consequently, there is a need for such a framework that supports the language of non-technical stakeholders. In this thesis, our goal is to present a framework in order to provide an automated Xtext grammar. The techniques of natural language processing (NLP) are utilized, paving the way to auto-generate the Xtext grammar from the natural-language requirements which become easily comprehensible by the non-technical stakeholders. The natural-language requirements also support the technical experts by including some reserved technical words such as string type, optional, etc.

Motivation

The requirement specification is the central pillar of software systems development. However, the technical experts elicit the requirements through collaboration with non-technical stakeholders. The non-technical experts cannot grasp the technical concepts of the Xtext grammar, so they cannot share their requirements and collaborate with the technical parties. Although current studies did not play a prominent contribution to the automation of the Xtext grammar. Therefore, there is a need for a framework that provides automated Xtext grammar, and offers several benefits to the public:

- i) It brings simplicity to the DSL development of the Xtext grammar.
- ii) It benefits several organizations by integrating into their workflow process seamlessly.
- iii) It reduces the development burden to support the technical experts.

Problem

The requirement engineering is an initial software development life cycle (SDLC) phase to support the development of software systems. The collaboration is observed to be highly effective in speeding up the requirement elicitation process. Requirements are elicited through the collaborative participation of diverse stakeholders. The emergence of Model-

driven engineering (MDE) brings simplification to the development of complex software systems across multiple domains. Developing Xtext-based DSLs is challenging due to its inherent complexity, which requires extensive knowledge to comprehend. From the perspective of technical stakeholders, the technical concepts comprising the Xtext grammar are easily interpreted by them. On the contrary, the same concepts are hard to understand by non-technical stakeholders because each stakeholder has its own understanding of grammatical notations. This problem negatively impacts the collaborative environment of stakeholders. Although various state-of-art studies have worked on the automation of MDE models, those studies have two-fold:

1. Overall, the models, i.e., UML, SysML, and metamodels, have been proposed with automation. However, the integrated Xtext and NLP approach has merely focused on the generation of formal or unambiguous requirements.
2. A restricted natural-language template has been utilized to write the requirement specifications. Still, the existing studies did not contribute to the automation of the Xtext grammar. Consequently, there is a need for such a framework that provides an auto-generated Xtext grammar through the utilization of NLP techniques to support the collaborative environment of the stakeholders.

Proposed Solution

Throughout this thesis, our research work consists of multiple activities, which are structured as follows: Initially, the main problem is identified. Then, an initial solution is proposed to resolve the identified problem. Then, the literature review is conducted to analyze the related papers through which the research gap and optimum solution are recognized. After, the primary approach is determined by analyzing the related studies corresponding to the proposed solution. Afterward, the proposed solution is implemented using tools and languages. Validation is presented with case studies to prove the feasibility of the proposed solution. Afterward, discussions and significant limitations of the proposed

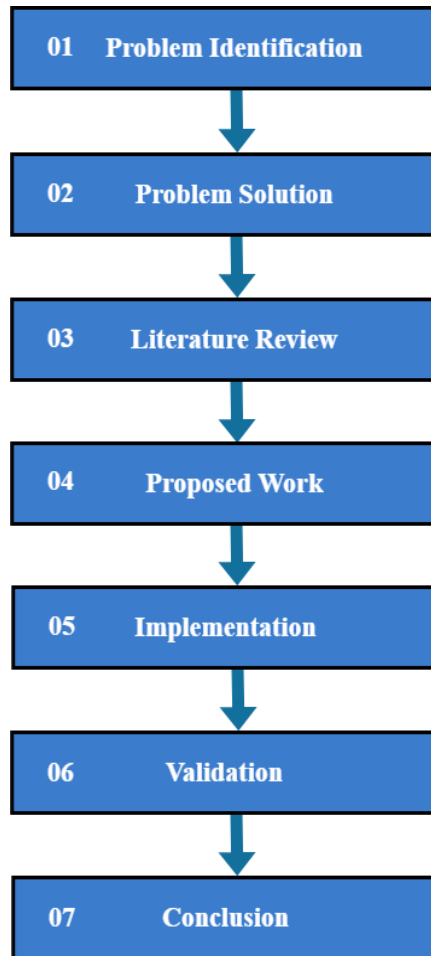


Figure 1.1: Research Review.

solution are presented. Finally, the research study is concluded by suggesting a few improvements. Figure 1.1 presents the outline of our research work. Our research work is presented with a proposed solution that automatically generates the Xtext grammar from the natural-language requirements using the NLP techniques. Figure 1.2 presents the overview of the research work.

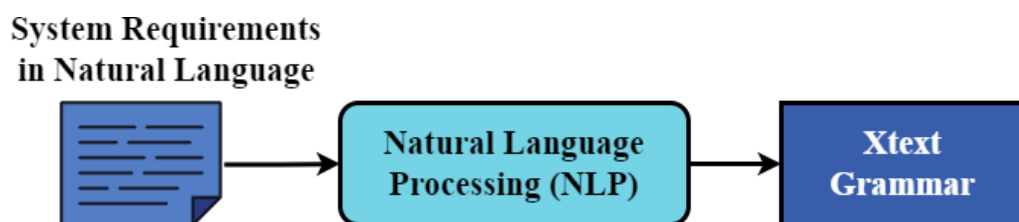


Figure 1.2: Overview of the Research Study.

To demonstrate the proposed solution, a tool named Natural-Language To Domain-Specific

Language, abbreviated as NL2DSL is developed. It has a user interface design to load the PDF file containing the textual requirements of the system. Particularly, the rule-based approach comprising the regular expressions is developed to extract the primary DSL elements of the Xtext. Further, the extracted DSL elements of the Xtext are saved in arrays to represent a DSL file of the .xtext extension. The feasibility of the proposed framework is evaluated through two case studies, i.e., the timing model is associated with the Volvo industry, and the diabetic manager is associated with the health system.

Thesis Organization

The thesis is organized as follows:

- Chapter 1 presents the introductory overview of the proposed approach, motivation, problem, and objectives.
- Chapter 2 presents the state-of-art by exploring the NLP approaches for the development of model-based software systems and overall utilization of the Xtext framework.
- Chapter 3 presented an approach to resolve the state-of-the-art problems.
- Chapter 4 discusses the implementation details of the proposed framework.
- Chapter 5 presents the validation of the proposed framework with case studies.
- Chapter 6 presents discussions and a few limitations of the proposed framework.
- Chapter 7 provides a brief conclusion with some enhancements for the proposed framework, which can be implemented in the future. .

Chapter 2

Preliminaries

This chapter is organized into three sections to determine the existing research work. **Section 2.1** describes the research work of previous studies while **Section 2.2** presents the research gap in the context of analyzed studies. **Section 2.3** presents the contributions of this master thesis.

Literature Review

This section presents the analysis of the previous studies in the context of model-driven engineering (MDE). In the MDE context, several existing studies have used natural-language processing techniques to automate the generation of SDLC phases, i.e., verification, testing, etc. Further, natural-language processing techniques have been employed to generate the UML and SysML architectural diagrams. Similarly, a few other studies have employed natural-language processing techniques in the context of metamodeling and Xtext framework. Further, we analyzed several previous studies related to the scenarios in which the Xtext framework is generally employed.

Natural Language Processing in MDE

This subsection presents the analysis of various studies in which the integrated utilization of MDE and NLP techniques is considered. After reviewing existing studies, we identified that several studies proposed their valuable contribution to the SDLC phases. Further, this section explored the modeling languages related to the MDE standards, such as Interaction Flow Modeling Language (IFML). Table 2.1. summarises these analyzed studies to understand the underlying terminologies.

For example, Sonbol et al. [7] proposed a framework to generate the business process modeling notation (BPMN) diagram. The proposed framework has two modules, including natural language analysis and modeling language generation. The natural language analysis module generated the concept map using NLP techniques, such as morphological analysis, lexical analysis, etc. Then, the other module translated the concept map to generate a text graph. Then, the BPMN diagram is translated from the text graph with the inclusion of syntactic and morphological refinements. The study of Sholiq et al. [8] proposed a framework to generate the business process modeling notation (BPMN) diagram from the textual requirements. The proposed methodology is tested with the requirements of various sentence structures, such as simple, complex, etc. Firstly, a heuristic rule-based approach with dependency parsing is adapted to extract the fact types from the requirements. Then, several mapping rules are defined to transform the extracted fact types into the BPMN modeling elements, i.e. activities, pool, datastore, etc. Furthermore, the generated BPMN-based elements are represented by the spreadsheet-based description. The study of Tangkawarow et al. [9] focused on developing a tool of ID2SBVR (Informal document to SBVR) to generate the operational rules of the SBVR (Semantics of Business and Vocabulary Rules) model from the informal interviewing documents. The tool utilized dependency parsing to determine the syntactic relations between the sentences. The tool performed the identification of candidate fact types using word patterns and triplet extraction. Then, the fact types, such as simple, compound, etc, are identified from the

Table 2.1: Natural Language Processing (NLP) in MDE.

Work	Approach	Domain	Purpose	Libraries \Tool	Input	Results
[7]	Semantic, Syntactic, & Morphological Manipulations	Process Management System i.e., Friedrich Dataset	Automation	Stanford CoreNLP	Textual Requirements	BPMN Diagram
[8]	Heuristic Rule-Based Approach with dependency parsing technique & Mapping Rules	Process Management System i.e., Registration Process	Functional Software Size Estimation	Stanza	Textual Requirements	BPMN Diagram
[9]	Candidate Fact Types Identification using Triplet Method Extraction	Process Management System i.e., University Library Dataset	Comprehensive Business Process	NLTK	Informal Interviewing Documents	SBVR Model
[10]	Applying the POS tags with ATL approach	General i.e., Undergraduate-based Project	Automation	Stanford CoreNLP	User Stories Requirements	Test cases
[11]	Semantic approach & Graph Coverage Criteria	Automotive System, i.e., Vehicle Braking System	Unrestricted Requirement Formalization to create the blended generation type	Spacy	Text documents comprising functional requirements	Test cases in the form of sequence diagrams
[12]	Semantic-Role Labeling	Automotive System, i.e., BodySense	System-Level Acceptance Tests	Gate Work-bench	Template-based Use case specifications document	Test cases
[13]	Rule-Based Approach comprising regular expressions	Automotive System i.e., Car Collision Avoidance System	Simplified Verification Process	SharpNLP	Textual Design Requirements comprising action and conditions	Verification assertions
[14]	Rule-Based Approach comprising regular expressions	Web Interfaces i.e., Movie Manager	Automated Validation of Web Interfaces	SharpNLP	Non-Template Natural Language Requirements	IFML Model
[15]	Enhanced Simple Sentence Generation, Deep Syntactic, Semantic Analysis & Temporal logic Approach	Safety-Critical Systems	Formal Notation of Requirements	Stanford CoreNLP	Textual System Requirements comprising triggers, etc	Formalized Requirements

candidate fact types to generate the operational rules of the SBVR. Then, the XML process definition language is used to view the generated SBVR model.

Allala et al. [10] proposed a framework to provide auto-generated test cases from the user story documents. Particularly, a metamodel is designed to represent the requirements of user stories. Then, the designed metamodel is transformed into the test case model using the ATL by accessing the results of POS tags. Gropler et al. [11] proposed a framework to generate the test cases from the natural-language requirements. The study identified the syntactic entities using dependency parsing, which were then mapped to semantic entities. Then, the UML state machine diagram is generated using the customized algorithm of the

rule-based approach that is transformed into the PetriNet model using the graph coverage criteria. This step is directed to generate abstract test cases in the form of sequence diagrams. Wang et al. [12] proposed a framework to automatically generate the test cases from the template-based natural-language requirements. This approach utilized the Restricted Use Case Modeling (RUCM) template to elicit the use case specifications in the embedded systems domain. This study aimed to create use-case models for the acceptance testing of the systems. Using the Gate Workbench, the approach utilized semantic role labeling to generate the OCL constraints from the requirements that capture the detection of use case steps. Then, the approach also employed an algorithm of alloy-based constraint-solving pattern to generate test input data. It led to the creation of a mapping table to support the generation of executable test cases. In study [13], a framework is presented to support the automatic verification of embedded systems. This study provided an AR2AA (automated requirement to assertion analyzer) tool, developed with a rule-based approach comprising regular expressions to extract the actions and conditions scenarios. This step determined the classification of requirements as verifiable or unverifiable.

Hamdani et al. [14] implemented an approach to generate the IFML model from the natural-language requirements for the simplified and automated validation of requirements in the context of web interfaces. Using the rule-based approach comprising the regular expressions, the various elements of the IFML model are extracted, such as view container, view component, actions, and events. Then, an XMI model is created where the extracted elements of the IFML model are viewed. In study [15], a framework is proposed to support the automated extraction and formalization of the requirements in the context of safety-critical systems. The study developed a tool named RCM-Extractor, where the elements such as action, trigger, and conditions are extracted from the textual requirements using the enhanced component extraction algorithm. Further, the study also extracted the sub-components using deep syntactic and semantic analysis, such as valid time, which proceeded to the augmentation of the extracted elements. Then, the temporal logic is applied to formalize the extracted requirements using the model-to-model

transformation of ATL.

Natural Language Processing in UML

This section comprises numerous studies where analysis focuses on the automated extraction of unified modeling languages (UML). Table 2.2. presents an overview of UML-based existing studies. The working directions of UML-based existing studies are defined below.

For example, the study [16] has proposed an integrated framework using natural language processing (NLP) and machine learning (ML) techniques to generate the UML use case diagram from the user stories-based requirements. The authors used the grammatical error correction model and dependency parsing to analyze the grammatical relations between the words of each sentence. The NLP model stores the extracted dataset in a data structure format. The ML model retrieved the dataset from the NLP model, which utilized the Naive Bayes algorithm that computed the classification for use-case identification. Then, the authors used the PlantUML tool to visualize the use case diagram from the generated output of the ML model. Similarly, the study [17] presented an approach to auto-generate the UML use case diagram from textual requirements. In this approach, the authors utilized the Open Information Extraction technique (OpenIE) to create the triplet model. Then, an Ego network graph is created to recognize the use cases, actors, and their relationships.

The study [18] introduced the automatic generation of UML diagrams such as sequence and class diagrams from the scenario-based user requirements. This research study utilized the noun-phrase technique to determine the validity of the sentences. Then, the authors performed the syntactical and lexical analysis to determine the relationship within the requirements. Then, the authors implemented the heuristic rule-based approach by accessing the results of POS tags to extract the information from the requirements. Then, the Plant UML tool is utilized to visualize the extracted information. Z. A. Hamza et al. [19] introduced a methodology to automatically extract the use-case diagrams from the func-

Table 2.2: Natural Language Processing (NLP) in UML.

Work	Approach	Domain	Purpose	Libraries Tool	Input	Results
[16]	Naive Bayes Classifier	General, i.e., Customer Appointment	Automation	Stanford CoreNLP	User-Story Documents	Use-Case Diagram
[17]	Triplet Method & Ego Network Extraction	General, i.e., ATM System	Automation	Stanford CoreNLP	Requirements in Unrestricted Natural- Language template	UML Diagrams i.e., use-cases, actors, & relationships
[18]	Heuristic Rule-Based Approach	General, i.e., Course Registration System	Automation	Stanford CoreNLP	Scenario- Based Requirements	UML Diagrams i.e., Class & Sequence Diagrams
[19]	Heuristic Rule-Based Approach	General, i.e. E-Store System	Automation	-	System Requirements from SRS Document	Use-Case Diagram
[20]	Semantic-Role Labeling extraction	General, i.e. Banking System	Unambiguous Requirements	OpenNLP	Textual Requirements	Class Diagram
[21]	Naive Bayes Classifier & Grammatical Patterns & Greedy Algorithm	Customized Dataset comprising system requirements	Automation	Spacy	Textual Requirements	Class Diagram
[22]	Heuristic Rule-Based Approach	General, i.e., Library System	Unambiguous Requirements	Stanford CoreNLP	Use-Case Specification Document	UML Diagrams, i.e., Use-Case & Activity
[23]	Prolog Rules & Ontology Created	General, i.e., Course Details	Redundancy Elimination	Stanford CoreNLP	User-Story Documents	Use-Case Diagram
[24]	Apply the Word2Vec & Hierarchical Agglomerative Clus- tering Algorithm & Heuristic Rule-Based Approach	General, i.e., Page Rankings	Semantic- Similarity between the requirements	Spacy	User-Story Documents	Use-Case Diagram
[25]	Parse Tree Analyzation	General, i.e., ATM system	Collaborative Environment	Tree Tagger	User Story Document	Use-Case Diagram
[26]	Rule-Based Approach comprising regular expressions & requirement traceability	General, i.e., Money Exchange Service	Unambiguous Requirements		Textual Requirements	Class Diagram
[27]	Heuristic Rule-Based Approach	General, i.e., Product Mechanism Service	Automated generation from various use-case templates	Stanford CoreNLP	Textual Requirements	Class Diagram

tional requirements of the systems. Initially, the authors utilized the ginger spell checker and grammar knowledge pattern technique for the requirements preprocessing. Further, the authors employed the heuristic rule-based approach to identify the elements of UML use-case diagrams, such as actors and use cases. The relationship elements between the actors and use cases are determined using the assigned tags of grammar knowledge pat-

terns. Alharbia et al. [20] focused on the auto-generated UML class diagrams from the textual requirements. The study performed the preprocessing, like removing stopwords, and a semantic-role labeling technique was applied to extract the desired elements of the UML class diagram, such as classes, attributes, methods, etc. Yang et al. [21] designed a tool with the utilization of NLP techniques to support the generation of UML class diagrams. In this study, the customized requirement dataset is created in the English language with the participation of outsourcing volunteers. Initially, performed the data preprocessing activities involving pronoun substitutions and sentence fragmentations. Then, the requirements are processed using the term frequency and inverse document frequencies (TF-IDF) techniques. Then, the preprocessed results are mapped to auto-generate the UML class fragments using the Naive Bayes Classifier. Lastly, the composition of the UML class fragments into a UML class diagram is employed using the greedy algorithm to visualize in the Plant UML tool. The study [22] proposed a framework to automatically generate the use-case and activity diagrams from the natural-language requirements. Initially, the syntactical rules are proposed to normalize the informal requirements that direct to the data preprocessing engine. Furthermore, multiple heuristic NLP rules are proposed to recognize multiple UML elements such as class, attributes, aggregation, etc. Then, the generated results are parsed with a few refinement rules to generate the refined UML class diagram. Nasiri et al. [23] proposed a framework by integrating the techniques of NLP and ontology to automatically generate UML architectural diagrams (use case, class, and package) from the user story documents. Then, the Prolog language is utilized with the pos tagging technique for the identification of the relationship elements, which is directed to generate the ontology model. The Plant UML editor is used to view the generated results.

Kochbati et al. [24] proposed an approach to support the automatic generation of the UML case diagram from the user story documents. The proposed approach initially computed the similarity scores between each sentence of processed textual requirements of user story documents. Then, the HAC (Hierarchical Agglomerative Clustering) algorithm

is utilized to compute the clustered labels for the text summarization. Those generated results are passed to the execution of heuristic NLP rules to extract the use case diagram elements. Similarly, the study of Elallaouia et al. [25] proposed an NLP-based transformation methodology to generate the UML use case diagrams from the requirements. The authors extracted the information related to the UML use case diagrams by applying the heuristic rule-based approach with the POS tags extraction technique and generated results visualized by the Visual Paradigm editor. In the study [26], a framework is proposed to generate the UML class diagram with the preprocessing of the textual requirements. Then, a set of NLP rules comprising regular expressions is applied to extract the elements of the UML class diagram with their traceability matrix to track each requirement approval. The study of Shewta et al. [27] proposed an approach that contributes to generating the UML class diagrams from the use-case templates. Particularly, the study devised conversion rules that generate the intermediate use-case template from the use-case template. Then, the approach rephrased the requirements by converting negative sentences to positive sentences. Then, the entities are identified using heuristic NLP rules with universal dependency relations to extract the elements of the UML class diagram. The performance of the proposed approach validates with the reference models created by the technical experts.

Natural Language Processing in SysML

From the analytical point of view, we identified numerous existing studies that performed the automated generation of the SysML model. Table 2.3 presents an overview of the terminologies used in the related studies. Below are examples of several existing studies explained in a descriptive summary.

Zhong et al. [28] presented the framework to automatically generate the SysML diagrams from the corpus of natural-language requirements. Therefore, the framework used the term-frequency and inverse document frequencies (tf-Idf) techniques to generate the key

Table 2.3: Natural Language Processing (NLP) in SysML.

Work	Approach	Domain	Purpose	Libraries Tool	Input	Results
[28]	Used the TF-IDF technique, Augmenting the Key-phrases & relationships using the OpenIE technique	General, i.e., Wikipedia, Patents	Comprehensive Diagrams	NLTK	Textual Documents, e.g., Manuals	Sysml Diagram, i.e., Block-based Diagrams
[29]	Semantic Role Labelling using Bert Model & create the requirement model using Hash-map	General, i.e., Wikipedia, Patents	Traceability Management	Hanlp	Textual Documents in the Chinese Language,	SysML Requirement Diagram
[30]	Heuristic Rule-Based Approach	Embedded i.e., Engine System	System Control Traceability Management	OpenNLP	textual requirements	SysML Requirement Diagram
[31]	Train with the Named Entity Recognition Model Train-Test Splitting Method	Embedded i.e., Railway System	System, Optimized modeling artifacts	Spacy	User-Story Documents	SysML Modeling Entities
[32]	Train using Convolutional Neural Network & Ontology model	Embedded i.e., Aviation System	System, Control Automation & Efficient Diagrams	Stanford CoreNLP	Textual Requirements	SysML Diagram, i.e., Block-Based Diagram

nouns from the requirements. Similarly, the relationship elements are extracted from the requirements by employing open information extraction (OpenIE) techniques such as semantic role labeling, relational nouns, etc. The key phrases and relationships are selected on the scores of candidate phrases that direct the selection of key relationships. The selected key phrases and relationships are augmented to extract the blocks and relationships, which were further organized into the SysML diagrams using the Plant UML. The study of Chen et al. [29] proposed a framework to support the automated generation of SysML diagrams using natural language processing (NLP) techniques from the textual requirements of the Chinese language. Firstly, word separation and POS tagging are performed to create the domain lexicon, then combine the domain thesaurus with regular expressions to eliminate redundancy. Secondly, semantic role labeling using the Bert model is performed to support the identification of desired elements of the SysML requirement diagram. Hwang et al. [30] proposed a framework that assists in the automated generation of SysML diagrams from the requirements. Initially, this proposed framework split the statements into the system needs and system requirements, directed to the implementation of a heuristic rule-based approach with the pos tags. Then, the generated SysML diagrams are executed in the Cameo System Modeler. Chami et al. [31] proposed a framework integrated with natural language processing and machine learning techniques to extract

the desired entities of SysML models. The proposed framework used the train-test split method to split the dataset into chunks. Then, the SysML model’s desired entities are extracted using the named entity recognition. Qie et al. [32] proposed an integrated natural language processing and deep learning framework to auto-generate the SysML models. Particularly, the framework encountered the named entity recognition methodology to identify elements such as subject, object, etc., from the textual requirements. Then, the convolutional neural network with SGD optimizer is used to determine the semantic relationship between the words of each sentence. Then, the generated results are mapped to the SysML model creation with the Rhapsody tool. There, web ontology language (OWL) is utilized to build the ontology for the result verification process.

Natural Language Processing in DSM

In the context of existing studies, Domain-specific modeling (DSM), i.e., graphical modeling and textual modeling, have a few contributions by integrating with NLP methodologies. Table 2.4 presents an overview of the terminologies used in the related studies. The contributions of DSM with a blend of NLP methodologies are concisely explained below.

Table 2.4: Natural Language Processing (NLP) in DSM.

Work	Approach	Domain	Purpose	Libraries Tool	Input	Results
[33]	Various heuristic Patterns are identified to extract the implicit and explicit clauses and to recognize the semantic formulations	General, i.e., Car Rental System	Comprehensible SBVR Model	Stanford CoreNLP	Textual Requirements comprising business rules	SBVR Model
[34]	Rule-Based Approach comprising regular expressions to Map each natural language Requirement & Apply Xtend Language	General, i.e., Birthday Book	Formal Requirements	-	Formal-Method Based Requirements	Z-Notation Textual Artifact
[35]	Rule-Based Approach comprising regular expressions & Apply Acceleo Language	General, i.e., PSS	Blended Modeling	Stanford CoreNLP	Template-Based Natural-Language Requirements	Ecore Metamodel & MPS DSL
[36]	CNL-based grammar rules proposed in EBNF notation using the WordNet & VerbNet Technique	Financial Domain	Formal Requirements	NLTK	Requirements extracted from SRS documents	Manual Xtext Grammar
[37]	Xtext Grammar was created with the utilization of named entity recognition & recurrent neural networks	Embedded system	Transition of Formal requirements aspects	-	Context-free grammar	Formalized requirements

Haj et al. [33] aimed to automatically transform the textual requirements into meta-model elements related to semantics and vocabulary of business rules (SBVR). This study performed the extraction of the implicit and explicit clauses from the statements by analyzing them with dependency parsing. Then, the study proposed two different algorithms to extract the dictionary terminologies of the business vocabulary, such as synonyms, abbreviations, and noun concepts. Further, the study extracted the business rules of logical formulations with the heuristic rules-based approach. Lastly, they saved the generated content into an XML editor. The study [34] provides a framework to support the generation of formal requirements with integrated techniques of natural-language techniques and formal methods. Firstly, the study utilized the terminologies of the Z-notations for the requirement specifications. Then, the Xtext grammar is created using the rule-based approach comprising the regular expressions to map the textual requirements with their identified POS tags. Secondly, they applied the model-to-text transformation using the Xtend to generate the z-notation-based textual artifact.

In study [35], a blended modeling framework is proposed to generate the Ecore meta-model and DSL of JetBrains MPS. The authors utilized the rule-based approach comprising the regular expressions to automatically generate the basic metamodeling elements such as relationships, attributes, etc. Then, the generated content is saved in an XML file. Further, the XML translator is applied to transform the saved content to the Xcore DSL. Then, the model-to-text transformation is applied with Acceleo language to generate the Ecore meta-model and JetBrains MPS DSL. Veizaga et al. [36] utilized the controlled natural language (CNL) to propose a Rimay requirement editor to support the clarity within the requirements of the financial domain. In this study, the requirements are extracted from the software requirement specification (SRS) document. Then, the authors analyzed the extracted requirements by applying WordNet and VerbNet techniques to propose the grammar rules in EBNF notation. Then, the Xtext grammar is created to express the unambiguous requirements using the proposed grammar rules. Further, the study [37] presented a requirement formalization framework in the domain of embedded

systems. This study presented the implementation of context-free grammar in the Xtext framework, where pseudo-English is transformed into the formal representation of temporal logic properties. Firstly, the named entity recognition model is utilized to produce the correct syntactic terminals, and then the recurrent neural network is used for the transition process.

Xtext in MDE

This subsection presents an overview of the existing studies where the Xtext framework is manually utilized in various ways i.e., blended modeling, verification or validation purposes, etc. The description of the research work of the existing studies is presented in subsequent sections.

Blended Modeling

Predoaia et al. [38] developed a hybrid modeling editor named Graphite to hold both graphical and textual modeling. The hybrid development of Sirius and Xtext models is performed with annotated metamodels. Further, model-to-text (M2T) transformation with Xtext language is performed for the code generation to support the delegation of API calls and global scoping to link the textual and graphical components. The study of Latifaj [39] developed a blended modeling framework where synchronization is performed between the two modeling languages, i.e., graphical and textual. Particularly, the blended modeling framework has two working contributions. i) Developed an Ecore-based mapping modeling language where the mapping rules are identified between two domain-specific modeling languages, i.e., graphical and textual. ii) The specifications of identified mapping rules are implemented using the Xtext framework. iii) With the scenario of mapping rules, the higher-order transformations are implemented using the Xtend to generate the synchronized model transformation within the syntax of the operational QVT language. In study [40], a blended modeling framework is proposed integrating the Xtext and ecore metamodeling languages. The framework utilized Aceleo and Java languages to gen-

erate the textual and graphical modeling notation from the input models. Due to this functionality, the EBNF grammar is implemented using Java CC to map the synchronized modeling elements. The synchronized model switching across the textual and graphical modeling notations within the Sirius editor is performed using the model-to-text transformations and vice versa. The study of [41] designed a blended modeling framework to support collaborative and cross-platform environments in the context of the UML state machine. Therefore, this study proposed an editor that performs the integrated modeling of Xtext, the graphical modeling framework (GMF), and JetBrains MPS. Further, the editor is also enabled with another component of Angular JS to generate a custom web app. The required changes in modeling are propagated to all participants using the emf. Cloud. Below, Table 2.5 presents an overview of these analyzed studies.

Table 2.5: Blended Modeling in Xtext.

Work	Approach	Domain	Purpose	Input	Results
[38]	Modeling of Xtext & Sirius & Apply the M2T with Xtend Language	Cloud Application	Linkage b/w elements	Emfactic Notation DSL	Manual Xtext Grammar & Automated code
[39]	Translation Mapping of two ecore models with Xtext & apply the M2T with the Xtend Language in the QVT-Operational syntax	Embedded System, i.e., UML-RT	Synchronizations	Two samples of Ecore meta-models	Synchronized Model Transformation across multiple notations
[40]	Generation of EBNF Grammar within the JavaCC platform & Apply the M2T & T2M transformation using Aceleo & Java language	General, i.e., PSS	Blended Modeling	Ecore Meta-model Xtext Grammar	Textual & Graphical Notations within Sirius Editor
[41]	Modeling of Xtext, MPS & GMF with a server of EMF.Cloud	Cloud Application	Cross-Platform Collaboration	Software Architecture Descriptions	Angular JS Web Application

Verification/Validation

Liu et al. [42] introduced a methodology for the verification of requirement documents. The proposed approach utilized the Xtext framework to create a DSL with the viewpoint of states and modes. Furthermore, the terminologies of temporal logic properties are

Table 2.6: Verification with Xtext.

Work	Approach	Domain	Purpose	Input	Results
[42]	Utilized the Xtext Framework with the temporal logic properties	General, i.e., Finite State Chart Models	Verification of requirements	Requirements documents on State and modes views	Verified Requirements
[43]	Created the Xtext Grammar & Apply the Henshin rules	Embedded System, i.e., Control systems	Verification of Deadlock's absence	System's Discrete Operations	Grafcet Model
[44]	Created the Xtext Grammar With Event-B specifications & apply the Xtend Language	Data-Driven Application i.e., AI planning problems	Validation of Formal methods	Event-B specification details	PDDL-based textual artifact

defined within the created Xtext grammar. Furthermore, the requirements are dynamically checked using the Nusmv modeling transformation using the Xtend language.

The study [43] has presented a model-driven approach to verify that the system has no existing deadlocks for the embedded system. Therefore, an Xtext domain-specific language is designed within the syntax of the UML state machine. Then, the Henshin rules are applied to transform the designed UML state chart models into graph models to trace the system behaviors. Further, the generated result was utilized for the Grafcet model to view the generated model of the programmable logic controller (PLC). F. Fourati et al. [44] provided a domain-specific language to validate the Event-B models. To realize this behavior, the proposed editor initially created the Xtext grammar, which captured the semantics of Event-B models. To further validate the Xtext grammar, the Planning Domain Definition Language (PDDL) is utilized by the model-to-text transformation of the Event-B models. Table 2.6 presents an overview of these analyzed studies.

Domain's Perspective

Brabra et al. [45] aim to orchestrate cloud resources with high-level elasticity resource management. To attain the objective, this study presented a domain-specific language

by leveraging Xtext and Sirius for the textual and graphical model representations. The study also defines the model-to-model transformation with a set of QVT-operational rules to transform the cloud resource descriptions into the Docker compose model. Further, the Xtend language is utilized for the model-to-text transformation to generate the YAML code from the generated docker model. Lastly, the results are forwarded to the cloud resource orchestration for the execution of elastic policies. Mzid et al. [46] introduced

Table 2.7: Domains Perspectives within Xtext.

Work	Approach	Domain	Purpose	Input	Results
[45]	Created the modeling using Xtext & Sirius & apply the M2M & M2T transformations using QVT-O & Xtend	Cloud Application	Optimum Elastic Cloud Resources	Cloud-Resource Entities	Docker model & YAML textual artifact
[46]	Created the Xtext grammar & Apply the Xtend language & conduct the M2M transformation using Papyrus.	Embedded systems, i.e., IoT Case Study	Reverse engineering	Source code	UML activity diagrams
[47]	Utilized the Xtext Framework with the terminologies of Mape-K loop patterns & ECA rules	Cloud Application, i.e., E-Learning	Automated Deployment	Cloud Deployment Concepts	Cloud Deployment Application with elasticity configuration

an approach to support behavioral reverse engineering. Therefore, this study consists of two phases which are the compiling and modeling phase. In the compiling phase, a programming language code is converted to an intermediate representation of the Gimple code. In the modeling phase, the Xtext grammar is created conforming to the syntax of Gimple code. Then, the study conducted the model-to-text transformation using the Xtend language to generate the ALF (Action language for foundational UML) code. Then, the Papyrus is utilized to create the UML activity diagrams from the generated ALF code. The study of Yangui et al. [47] proposed a domain-specific language editor named AutoCaDep to support the simplified execution of cloud deployment applications across SAAS and PASS platforms. The editor defines the Xtext grammar by adapting the ECA (event, condition, and action) rules and mape-k loop patterns to attain this objective. Further, the model-to-text transformation is performed to deploy the cloud services. Table 2.7. presents an overview of these analyzed studies.

Artificial Intelligence

Mohin et al. [48] proposed an editor with the utilization of various machine-learning classifiers to support prediction analytics in the domain of IoT systems. The study created a domain-specific language editor of ML-quadrant implemented using the Xtext framework and generated the Python code using the Xtend Language.

Table 2.8: AI within Xtext.

Work	Approach	Domain	Purpose	Input	Results
[48]	Utilized the Xtext Framework with the Machine Learning Classifiers & M2T with Xtend language	Iot System, i.e., Ping Pong	Prediction	Behavioral Patterns of IOTs	Auto-generated Python code
[49]	Designed a DSL using the Xtext framework with the natural-language analysis by targeting the concrete Platforms of Discord, Slack	Chat-Bot Application	Cross-platform chatBot Application	User messages	chatBot Application
[50]	Utilized the Xtext Framework with the Neural Network Classifier & M2T with Xtend language	General, i.e., Benchmark Dataset	Structured Requirements formulation	Designed metamodel for requirement specifications	Synthetic Data Generator

The study of Daniel et al. [49] introduced the Xatkit framework to design chatbot applications. In this study, the proposed framework implemented the Xtext grammar by integrating the deployment configuration with Slack and Discord and also recognized the user’s intentions with NLP techniques. The study of Jahic et al. [50] devised a DSL approach to develop the requirements specifications of the deep neural networks. Particularly, this study leveraged the Xtext framework that developed a structured metamodel to simplify the customer requirements for the dataset selection criteria and desired key properties of deep neural networks. Further, the Xtext DSL and model-to-text transformation using Xtend are performed to conduct dataset augmentation for the prediction analytics. Table 2.8. presents an overview of these analyzed studies.

Research Gap

This subsection concludes that most of the existing studies have performed the automated generation in the context of MDE, i.e., verification aspects, test cases, UML, and SysML architectural diagrams. However, a few studies have focused on the automated generation of metamodels. The current studies have also focused on the generation of formal requirements using the NLP techniques within the context of the Xtext framework. Moreover, it is analyzed that the existing studies have used the Xtext framework in multiple ways, i.e., blended modeling, verification, validation, etc., across various domains.

Table 2.9: Overview of the Limitations analyzed from the Literature Review.

Work	Purpose	Approach	Input \Context	Result	Limitations
[35]	Formal Requirements	Mapping of NLP Rules & apply the Xtend Language	Formal Method-based textual Requirements	Z-Notation Textual Artifact	Manual Xtext Grammar
[36]	Unambiguous Requirements	CNL grammar rules defined in EBNF notation using the WordNet & VerbNet Technique	Requirements extracted from the SRS documents	Xtext Grammar	Manual Xtext Grammar
[37]	Formal requirements	Named Entity Recognition & Recurrent Neural Network	Pseudo-English Text	Formal Requirements	Manual Xtext Grammar

We explored existing studies to determine the utilization of the NLP techniques in the context of the Xtext framework. The research study of [35] created the manual Xtext grammar by mapping the requirements with NLP rules. The study of [36] created the manual Xtext grammar by proposing the CNL grammar rules in EBNF notation from the textual requirements of the financial domain using the wordNet and verbNet NLP technique. Similarly, the research study [37], utilized the NLP technique of named entity recognition and deep learning of recurrent neural network, both are utilized to generate the formal requirements from the texts of pseudo-English.

However, current studies utilized the Xtext framework in various ways but did not propose

a methodology that automatically generates the Xtext grammar. The summary of the limitations through the analysis of the existing studies is presented in Table 2.9.

Contributions

It is analyzed from the literature review that several studies have proposed automatic generation of the MDE models, i.e., test cases, UML and SysML architectural models, etc., using natural language processing (NLP) techniques. It is also analyzed that the NLP techniques and Xtext framework are utilized to support the formal or unambiguous requirements. However, our approach is different from the approaches of the existing studies in the following aspects:

a) automatically generate the Xtext grammar from the textual requirements in natural language without relying on any natural language template. This step documents the natural-language requirements in any English language style. b) The tool is implemented with the rule-based approach to extract the primary DSL elements of the Xtext from the natural-language requirements. This step helps to reduce the development complexity of the Xtext grammar. c) Extracted DSL elements conforming to the Xtext grammar, save in the DSL file of the .xtext extension. d) Validation is performed with two case studies. Hence, such a framework where the Xtext grammar is automatically generated is hard to find in the literature review.

Chapter 3

Methodology

In this chapter, we will delve into the methodology of the proposed framework, organized into two sections 3.1 and 3.2. **Section 3.1** provides the working details of the proposed algorithm, and **Section 3.2** provides the underlying details of the transformation engine.

Figure ?? represents the high-level view of the proposed framework's methodology. The elicitation of requirements from diverse stakeholders to support the development of efficient software systems is a challenging process. In the context of the MDE, the underlying terminologies to develop the Xtext grammar are hard to understand by the non-technical stakeholders. Due to the existence of this technical barrier, non-technical experts cannot share their requirements with the technical team. Due to this issue, mutual consensus among the stakeholders is difficult to manage, reflecting a negative impact on their collaboration environment which directly delays the development of software systems. Therefore, natural-language requirements are required to utilize and it is impossible to manage the different styles of requirements. The proposed methodology specifies some rules on the basis of an earlier research study [14], which supports the requirements to be written in English. The description of those rules is given below:

- The requirements should be clearly described in short sentences. While long sentences can create ambiguities.

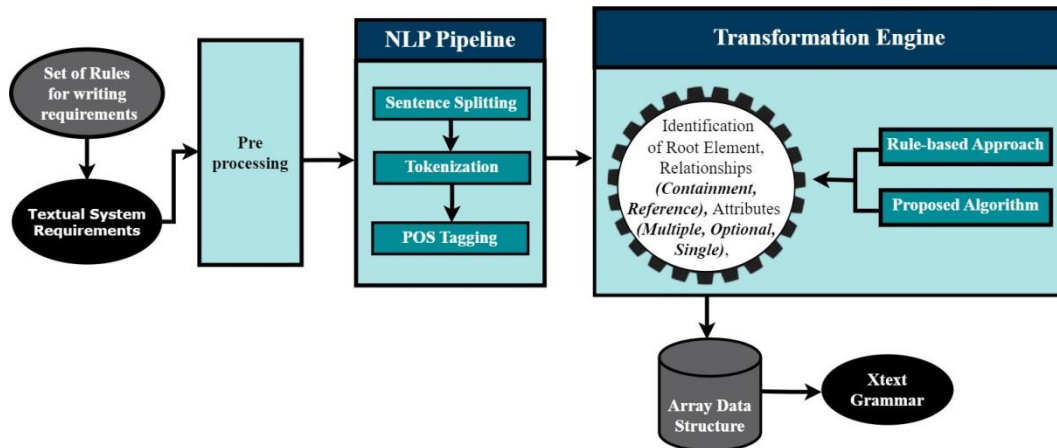


Figure 3.1: High Level of the Proposed Framework.

- The requirements should be simple and defined according to the user’s perspective.
- The requirement should be described in active voice sentences.
- All the requirements should be free from non-functional concepts and negative sentences.

Firstly, the natural-language requirements are written in the form of plain text according to the aforementioned rules. Secondly, a proposed algorithm is developed to support the extraction of the Xtext DSL elements from the textual requirements. Particularly, some processing tasks and NLP techniques are required to apply to the textual requirements. Furthermore, certain transformation rules are applied to the processed textual requirements for the automated generation of the Xtext grammar. The description of the proposed algorithm is given in the subsequent subsection.

Proposed Algorithm

The proposed algorithm comprises various steps that support the extraction of the primary DSL elements of the Xtext. After the execution of the proposed algorithm, the extracted DSL elements conforming to the Xtext grammar, such as the root element, relationships, and attributes are added into a DSL file of .xtext extension. A few activities are performed before the execution of the proposed algorithm which are defined as follows:

Step a. Input a PDF document containing the textual requirements of the system, which are based on the rules described previously for the requirement specification.

Step b. Preprocessing is applied to the textual requirements to remove the punctuation marks.

Step c. Apply the NLP techniques within the NLP pipeline, including sentence splitting, tokenization, and POS (Parts of Speech) tagging.

Step d. Conversion of plural and proper nouns (NNS & NNP) to singular nouns (NN).

After the execution of the above-defined activities, the text is fed into the proposed algorithm. There, the text is in the form of tagged sentences as the splitting method of the tagged sentences is based on the full-stop delimiter. Then, the proposed NLP rules are systematically applied to the preprocessed requirements within the proposed algorithm. The rules are composed of regular expressions to match the relevant tags through the Java Regex library. The workflow of the proposed algorithm is presented in Figure 3.2.

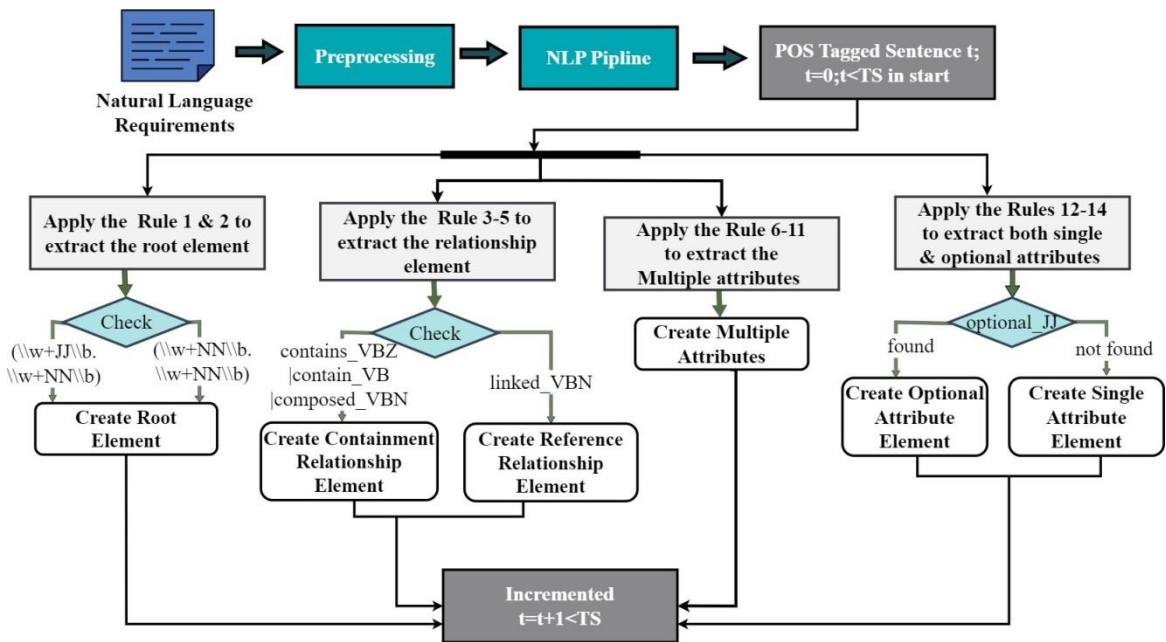


Figure 3.2: Proposed Algorithm.

Steps of the algorithm: The execution of the proposed algorithm has a dependency on the nested for-loop to support the extraction process.

Initially, a main for-loop is defined where the tagged sentences are set to an initial value of $t=0$, which means that the first tagged sentence is checked to the total number of tagged sentences named TS. Then, an if-condition is defined to check whether the tagged sentences have a ‘where_WRB’ tag. If this condition returns false, then subsequently apply the NLP rules on the first tagged sentence. Similarly, if the condition returns true, then subsequently tokenize the tagged sentences based on the ‘where_WRB’ tag. After the tokenization, an inner for-loop is defined to check the whole tokenized sentences with a value of $u=0$, which means the first tokenized sentence is checked to the total number of tokenized sentences named US. Then, the entire NLP rules are applied to the first tokenized sentence. In summary, if the tagged sentences have any ‘where_WRB’ tag, tokenization is performed and NLP rules are applied to the tokenized sentences for the identification of the Xtext DSL elements, i.e., root Element, relationship Element, and attributes. Similarly, suppose the tagged sentences do not have the ‘where_WRB’ tag. In that case, the NLP rules apply to the tagged sentences for the identification process. The execution workflow of the applied NLP rules is described as follows:

Step 1: First, apply the Root Element Rules to the sentence for extracting the root element.

Suppose a sentence i.e., tagged or tokenized is matched with the proposed NLP rules to extract the root element of the entire Xtext grammar. In that case, it is followed by two conditions to retrieve the concept name of the root element and keyword parameter. The expression of both conditions is described below.

- An if- condition is defined as the sentence, i.e., tagged or tokenized, containing a combination of adjective JJ and Noun NN tag. If this specified condition returns true, then the concept name of the root element and keyword parameters are to be extracted.
- Suppose the sentence i.e., tagged or tokenized, has two consecutive nouns NN. If this condition returns true, then both the root element’s concept name and keyword

parameter are to be extracted.

Step 2: The second step executes the Relationship rules to extract relationship elements.

Throughout, this proposed algorithm, we identify the relationships of two different categories, i.e., containment & reference relationship with their association constraint which can be specified as multiple, single, and optional. The proposed NLP rules apply to the given sentence, i.e., tagged or tokenized to support the extraction process of relationship elements. The execution of the proposed NLP rules for both of the relationship categories is defined below.

- Suppose a sentence, i.e., tagged or tokenized either has ‘contains_VBZ’, ‘contain_VB’, or ‘composed_VBN’, then it targets the containment type of relationship category. Similarly, if the sentence, i.e., tagged or tokenized has ‘linked_VBN’, then it targets the reference type of relationship category. We define some conditions to target other elements of these two relationship categories, given below:
 - Suppose the given sentence can have an association constraint of ‘multiple_JJ’. Then, the following elements are defined.
 - i) Association operator is set to ‘+=’.
 - ii) Association constraint is set to ‘*’.
 - Suppose the given sentence can have an association constraint of ‘optional_JJ’. Then, the following elements are defined.
 - i) Association operator is set to ‘=’.
 - ii) Association constraint is set to ‘?’.
 - Suppose the given sentence can have an association constraint of ‘single_JJ’. Then, the following elements are defined.
 - i) Association operator is set to ‘=’.

Additionally, a few post-processing tasks are executed to ensure that the identical root of the relationship element is undefined in the output of the DSL file, then it should be added. Further, the rest of the relationship elements should be added in the Xtext DSL file next to the index of the identified root element.

Step 3: The third step specifies the execution of Multiple Attribute Rules on the given sentence for extracting multiple attributes.

If the sentence, i.e., tagged or tokenized, is matched with the proposed NLP rules to identify the set of multiple attributes, then the extraction process executes. Firstly, check the sentences have an 'it_PRP' tag for extracting the root of multiple attributes. Then, extract the attribute names with their specified datatypes, such as string, id, etc.

Step 4: The fourth and last step of the proposed algorithm has the purpose of extracting the Single and optional Attribute Rules from the given sentence.

Suppose a given sentence, i.e., tagged or tokenized, is matched with the proposed NLP rules to identify the single and optional attributes. Therefore, two conditions are specified to support the valid extraction of both attributes.

- An if-condition is specified to check that the given sentence has an optionalJJ tag that supports the extraction process of the optional attributes. Then, check that the sentence has an 'it_PRP' tag to extract the root element. Then, extract the elements of the optional tag and attribute name with their specified datatypes, i.e., string, etc. Additionally, a few post-processing tasks are executed to ensure that if the identical root element has already been included in the output of the DSL file of the .xtext extension, it should not be added again.
- Similarly, consider that the sentence does not have an optionalJJ tag that supports the extraction process of the single attributes. Then, check that the sentence does not have an 'it_PRP' tag for extracting the root element. Then, extract the attribute names with their specified datatypes, i.e., string, etc. Additionally, a few post-processing tasks are

executed to ensure that if the identical root element has already been included, it should not be added again.

In the absence of the 'where_WRB' tag, all the rules are applied to the single tagged sentence for the desired elements identification, and the value of t is incremented, i.e., $t=t+1$. It means $t=1$, and now the second sentence is passed for matching within the same sequential workflow of the proposed algorithm. Therefore, it is summarized that the proposed algorithm will check all the tagged sentences until the value of t becomes equal to the value of TS . When the 'where_WRB' tag is defined in the tagged sentence, first perform the tokenization based on the 'where_WRB' tag. Then, all the rules are first applied to the single tokenized sentence to identify the desired elements. Then, the value of u is incremented within an inner for-loop, i.e., $u=u+1 < US$. It means $u=1$, and now the second sentence is passed for matching within the same sequential workflow of the proposed algorithm.

Figure 3.2 describes the complete workflow of the proposed algorithm. The extracted results through the implementation of proposed NLP rules are stored in arrays, and then append the results to a DSL file of the .xtext extension. Before the implementation of the proposed NLP rules, the input textual requirements are required to be passed through a few steps. Our approach generates an Xtext grammar to support the collaborative environment among the stakeholders in order to develop critical software systems. The whole process is described below.

1. Pre-Processing of Natural Language (NL)-based text

The textual requirements are defined within a PDF file. The initial state of the text is rough where some unnecessary information is defined, which can change the syntactic meaning of the text. Therefore, it is necessary to clean the input of textual requirements with the removal of punctuation marks. The punctuation marks are removed using the replace function to obtain the text in a structured format.

2. Processing of Natural Language (NL)-based text using NLP pipeline

Stanford CoreNLP [51] is a Java-based NLP library to perform natural language processing activities. It analyzes the natural-language requirements by labeling each word to its corresponding POS (Parts of Speech) tags, such as adjectives, nouns, verbs, etc. Stanford CoreNLP has the following capabilities.

- (a) **Sentence Splitting** The activity of sentence splitting adds simplicity by transforming large textual requirements into sentences. After preprocessing, the preprocessed textual requirements are split into individual sentences with a dot delimiter. Therefore, the textual requirements are converted into a string array, which is later passed to the tokenization activity. The activity of sentence splitting is required before the execution of the tokenization because the proposed NLP rules need to be applied to the individual sentences of the textual requirements.
- (b) **Tokenization** Tokenization is a process of analyzing large textual requirements into sentences, words, or phrases. After performing the activity of sentence splitting, the acquired results are checked based on the words.
- (c) **POS Tagging** The POS (Parts-Of-Speech) tagging is applied to the results acquired by tokenization. The activity of POS tagging is required to operate the execution of the proposed NLP rules. Therefore, we used the Maxent tagger provided by the Stanford CoreNLP library that assigns the POS tags to each word of the given sentence to identify the verbs, nouns, etc. Then, POS tags for each individual sentence are obtained. The list of the POS tags is presented with their descriptions and examples in Table 3.1.

Transformation Rules For the Identification of Xtext Elements

This section presents the details of proposed NLP rules. The transformation engine is the main component of the proposed framework, in which the rule-based approach is

Table 3.1: List of POS Tags.

Sr.#	Tag	Description	Examples
1	NN	Nouns in Singular Form	system, description, constraint
2	NNP	Proper Noun in the singular form	EventFunctionflowport, EventChain
3	NNS	Noun in plural form	concepts, attributes
4	TO	To	To
5	IN	Preposition/Subordinating conjunction	for, of, with, by, like
6	CC	Coordinating conjunction	And
7	VBN	Verb, Past participle	composed, named, preceded
8	VBZ	verb, 3rd person singular, Present participle	is, contains, has
9	VB	verb, the base form	have, contain, define, specify, precede
10	VBP	Verb, non-3rd person singular present	are
11	MD	Modal	may, must, should
12	JJ	Adjective	diabetic, multiple, optional, main
13	VBG	Gerund Verb	Defining, including
14	DT	Determiner	a, an, the
15	WDT	WH-Determiner	that
16	PRP	Possessive pronoun	it
17	RB	Adverb	only
18	CD	Cardinal Number	one

implemented to extract the primary DSL elements of the Xtext. The results attained by the activity of POS tagging are passed to the transformation engine. Particularly, several rules are implemented to attain the DSL elements of the Xtext with the help of NLP techniques. We classified the rules into two categories, i.e., General Purpose Rules and Special Purpose Rules.

General Purpose Rules The general-purpose rules are applied to the results acquired from the POS tagging. These rules are specified to simplify the implementation of special-purpose rules. A few general-purpose rules are defined below:

- **Rule No. 1: Conversion of NNS To NN** The entities are extracted from the tagged sentences identified as NNS (plural noun), which are transformed into the NN (singular noun) tag. For example, concepts in *'It is composed of multiple description concepts'* is tagged as a plural noun (NNS), which is converted to a singular noun

(NN).

- **Rule No. 2: Conversion of NNP To NN** The entities are extracted from the tagged sentences identified as NNP (proper noun), which are transformed into the NN (singular noun) tag. For example, *'The EventChain concept must have a name and category of string type'*. Here, EventChain is tagged as a proper noun (NNP), which is converted to a singular noun (NN).

Special Purpose Rules: To generate the primary DSL elements of the Xtext, we applied the special-purpose rules on all NN POS tags achieved after the application of general-purpose rules. The special-purpose rules are based on a rule-based approach composed of regular expressions. These regular expressions are then applied to the text using the string-matching technique provided by the Java Regex library. Each input sentence is matched by applying these NLP rules to view the pattern of these sentences and select the sentences from which the Xtext grammar can be generated. The pattern is a combination of some POS tags. If the pattern of a sentence is matched with the pattern of NLP rules, then the sentence is selected for the generation of the Xtext grammar. Particularly, we define 14 NLP rules to extract the primary DSL elements, i.e., root element, relationship element, and attribute element. The study [14] suggests that as the complexity of the sentences increases, there is a corresponding rise in the number of rules to consider. To achieve the optimal accuracy, more rules are required to be included. The special-purpose rules are defined below, with descriptions, examples, and diagrammatical representations.

1. **Rules for identification of Root Element** The Xtext grammar is initiated with a parser rule by following its syntax. Particularly, a colon and a keyword parameter appear after the declaration of the parser rule. The keyword is declared within single quotation marks. It is simply known as the root element that is to be defined right after the Xtext Package declaration. **Such as** `timing_model:'timing_model'`. By focusing on such root element identification, we have defined two NLP rules. Figure 3.3 shows the representation of the rules where the Red circle presents the

mandatory feature, and the green circle presents the optional feature, which may or not be present.

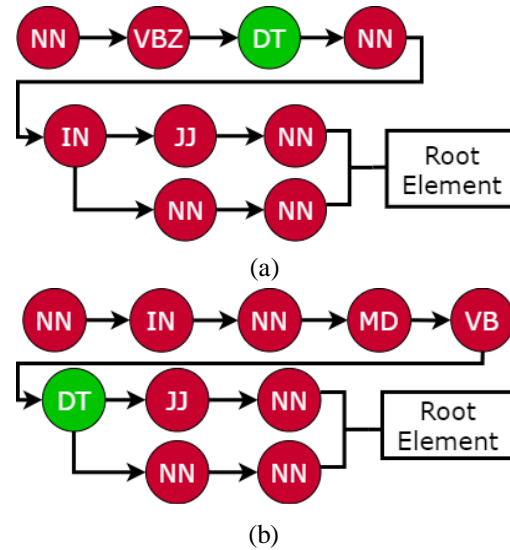


Figure 3.3: Graphical Description of Rules for the Root Element Identification.

- **Rule No. 1** Consider a sentence where the noun NN tag with the verb VBZ appears at the start followed by an optional determiner DT tag, noun tag, and preposition IN tag, as shown in Figure 3.3a. Then, those tags follow one of the two scenarios defined below to form the root element.

- Those tags proceed with an adjective JJ tag and an NN tag.
- Those tags proceed with two consecutive NN tags.

Tagged Output The_DT system_NN presents_VBZ a_DT scenario_NN of_IN timing_NN model_NN.

Regular Expression Transformation “ $(\backslash|w+NN\backslash|b.\backslash|w+VBZ\backslash|b.(?:\backslash|w+DT\backslash|b.)?)\backslash|w+NN\backslash|b.\backslash|w+IN\backslash|b.(\backslash|w+JJ\backslash|b.\backslash|w+NN\backslash|b|\backslash|w+NN\backslash|b.\backslash|w+NN\backslash|b))$ ”

Extraction-Example The input sentence is:

“The system presents a scenario of timing model.”

Here, the ‘timing model’ is tagged as a pattern of two consecutive nouns NN tag. Applying this defined NLP rule to a given sentence, the root element is extracted including two consecutive nouns of the NN tag following the VBZ and IN tag.

- **Rule No. 2** Consider a sentence where the noun NN tag with the preposition IN tag and a noun tag appears at the start of a sentence. Then, look for the combination of MD- VB tag that appears before an optional determiner DT tag, as shown in Figure 3.3b. Then, those tags follow one of the two scenarios defined below.

- i) Those tags proceed with an adjective JJ tag and an NN tag.
- ii) Those tags proceed with two consecutive NN tags.

Tagged Output The_DT scenario_NN of_IN system_NN can_MD be_VB a_DT timing_NN model_NN.

Regular Expression Transformation “(\\w+NN\\b.\\w+IN\\b.\\w+NN\\b.\\w+MD\\b.\\w+VB\\b.(?:\\w+DT\\b.)?)(\\w+JJ\\b.\\w+NN\\b|\\w+NN\\b.\\w+NN\\b)”

Extraction-Example The input sentence is:

“The scenario of system can be a timing model.”

Here, the timing model is tagged as the pattern of two consecutive nouns NN. Applying this NLP rule to the given sentence, the root element is extracted following the VB tag.

2. **Rules for the identification of relationship element** The relationship element starts with a parser rule followed by a colon notation and a keyword parameter in the Xtext grammar. It is known as the root of the relationship element which is undefined in the case of the PRP tag. A relationship element has four components:

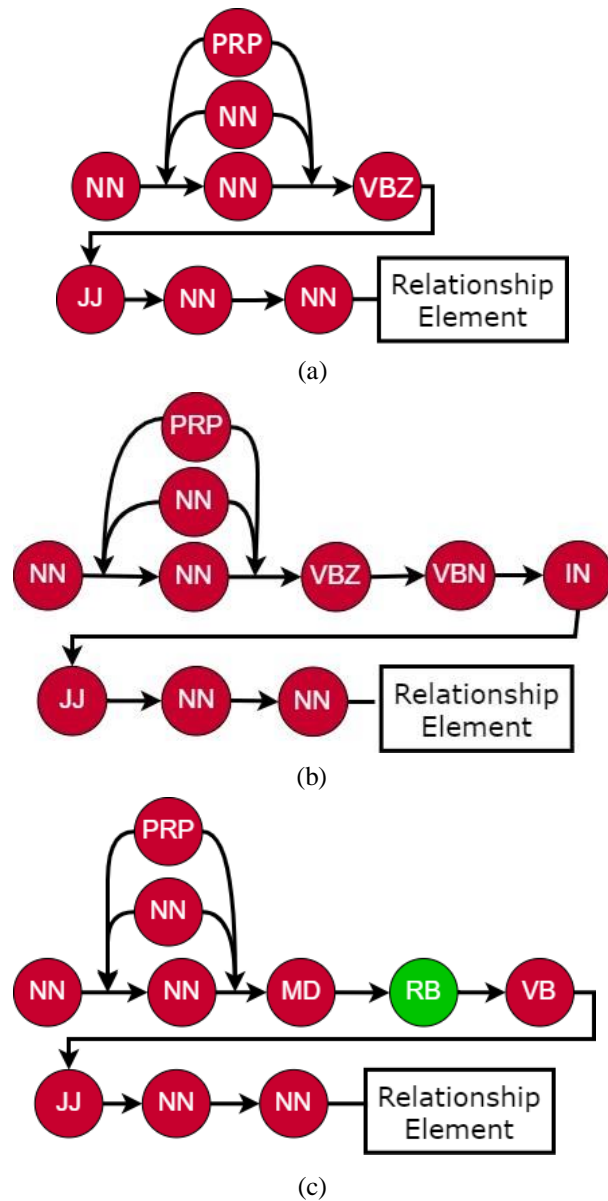


Figure 3.4: Graphical Description of Rules for the Relationship Element Identification.

(i) a root element (ii) an association name, (iii) an association operator, (iv) and the child element with multiplicity constraints. Therefore, the relationship element is represented,

Such as description: ‘description’ (EventFunction+=EventFunction)*

We define three NLP rules for relationship element identification. Figure 3.4 shows the representation of the rules where the Red circle presents the mandatory feature, and the green circle presents the optional feature, which may or not be present.

- **Rule No. 1** Consider a sentence that starts with two consecutive nouns of the NN tag or if a single NN tag appears with a VBZ tag, which identifies the root of the relationship element. A PRP tag might appear instead of an NN tag at the start of a sentence. Then, it is followed by an Adjective JJ tag with two consecutive nouns of the NN tag to identify the rest of the relationship elements. Depicted in Figure 3.4a.

Tagged Output The DT description NN concept NN contains VBZ multiple JJ EventFunction NN concepts NN.

Regular Expression Transformation “((\w+NN\b.\w+NN\b.|\w+NN\b.\w+PRP\b.)(\w+VBZ\b.\w+JJ\b.\w+NN\b.\w+ NN\b))”

Extraction-Example The input sentence is:

“The description concept contains multiple EventFunction concepts.”

Applying this rule to the given sentence, the relationship element is extracted under the following rationals.

- (a) NN/PRP before the VBZ tag refers to the root of the relationship element.
 - (b) Pattern after the adjective JJ refers to the child element.
 - (c) Adjective JJ tag refers to the association constraint of the relationship element.
- **Rule No. 2** Consider a sentence that starts with two consecutive nouns of the NN tag or if a single NN tag appears with a VBZ tag, which identifies the root of the relationship element. A PRP tag might appear at the start of a sentence instead of NN. Then, a combination of the VBN- IN tag is followed by a JJ tag with two consecutive nouns of the NN tag in the same sentence to identify the rest of the relationship elements. Depicted in Figure 3.4b.

Tagged Output The DT description NN concept NN is VBZ composed VBN

of_IN multiple_JJ EventFunction_NN concepts_NN.

Regular Expression Transformation “((\\w+NN\\b.\\w+NN\\b.\\w+NN\\b.\\w+PRP\\b.)(\\w+VBZ\\b.\\w+VBN\\b.\\w+IN\\b.\\w+JJ\\b.\\w+NN\\b.\\w+NN\\b))”

Extraction-Example The input sentence is:

“The description concept is composed of multiple EventFunction concepts.”

Applying this rule to the given sentence, the relationship element is extracted under the following rationals.

- (a) NN/PRP before the VBZ tag refers to the root of the relationship element.
 - (b) Pattern after the adjective JJ refers to the child element.
 - (c) Adjective JJ tag refers to the association constraint of the relationship element.
- **Rule No. 3** Suppose a sentence starts with two consecutive nouns of the NN tag, or if an NN appears with an MD tag, considered as a root of the relationship element. A PRP tag might appear at the start of a sentence instead of NN. Then, those tags are followed by an optional RB tag and a VB tag before the JJ tag associated with two consecutive nouns of the NN tag, which identifies the rest of the relationship elements. Depicted in Figure 3.4c.

Tagged Output

- i) It_PRP should_MD contain_VB multiple_JJ constraint_NN concepts_NN.
- ii) It_PRP should_MD only_RB contain_VB multiple_JJ expression_NN concepts_NN.

Regular Expression Transformation “((\\w+NN\\b.\\w+NN\\b.\\w+NN\\b.)”

$\backslash b. | \backslash w+ PRP \backslash b.) (\backslash w+ MD \backslash b. (? : (\backslash w+ RB \backslash b.) ?) \backslash w+ VB \backslash b. \backslash w+ JJ \backslash b. \backslash w+ NN \backslash b. \backslash w+ NN \backslash b))$ ”

Extraction-Example The input sentence is:

“It should contain multiple expression concepts.”

Applying this rule to the given sentence, the relationship element is extracted under the following rationals.

- (a) NN/PRP tag before the MD tag refers to the root of the relationship element.
- (b) Pattern after the adjective JJ refers to the child element.
- (c) Adjective JJ tag refers to the association constraint of the relationship element.

3. **Rules for the identification of Multiple Attribute elements** The Xtext grammar is also comprised of multiple attributes. Therefore, the attribute element of the multiple category starts with a parser rule followed by a colon notation and a keyword parameter in the Xtext grammar. It is known as the root of the multiple attribute element which is undefined in the case of PRP tag. After the declaration of this root element, the multiple (two) attributes with their datatypes are defined, i.e., STRING, etc.

Such as EventFunction: ‘EventFunction’

name=STRING

category=STRING

We define a set of six NLP rules for the identification of multiple attributes. Figure and Figure 3.6 depict the description of rules, where the red circle defines the mandatory feature and the green circle represents the optional feature, which may or may not be present.

- **Rule No. 1** Suppose a sentence starts with two consecutive nouns NN tag, or if a single noun NN is followed by the combination of MD-VB tag, considered as a root of the multiple attributes. A PRP tag might appear instead of an NN tag at the start of a sentence. Then, those tags proceed with the tags of an optional determiner DT and a noun NN with a combination of the VBN- IN tag followed by the optional Determiner DT tag and a noun NN tag. This combination identifies the set of multiple (two) attributes. Depicted in Figure 3.5a.

Regular Expression Transformation “((\|w+NN\|b.\|w+NN\|b.\|w+NN\|b.\|w+PRP\|b.)(\|w+MD\|b.\|w+VB\|b.(?:(\|w+DT\|b.)?)\|w+NN\|b.\|w+VBN\|b.\|w+IN\|b.(?:(\|w+DT\|b.)?)\|w+NN\|b))”

Tagged Output The DT EventFunction NN concept NN must MD specify VB a DT name NN preceded VBN by IN a DT category NN of IN string NN type NN.

Extraction-Example The input sentence is:

“The EventFunction concept must specify a name preceded by a category of string type.”

This defined NLP rule applies to the given sentence under the set of following rationals.

- (a) NN/PRP before the MD tag refers to the root of the multiple attribute element.
- (b) VB tag and combination of VBN-IN tag are used to extract the multiple attribute’s names.
- (c) After the pattern, the IN tag is again defined in a sentence before two consecutive nouns referring to the datatype.

- **Rule No. 2** Suppose a sentence starts with two consecutive nouns NN tag, or if a single noun NN is followed by the combination of MD-VB tag, considered as a root of the multiple attributes. Depicted in Figure 3.5b. A PRP tag might appear at the start of a sentence instead of NN. Then, those previous tags are followed by the tags of optional preposition IN and an optional determiner DT associated with a noun phrase containing a noun NN, CC, and noun NN tag corresponding to the set of multiple (two) attributes.

Tagged Output

- It_PRP must_MD specify_VB by_IN name_NN and_CC category_NN of_IN string_NN type_NN.
- It_PRP must_MD specify_VB have_VB a_DT name_NN and_CC category_NN of_IN string_NN type_NN.

Regular Expression Transformation “ $((\backslash w+NN\backslash b.\backslash w+NN\backslash b.|\backslash w+NN\backslash b.|\backslash w+PRP\backslash b.)\backslash w+MD\backslash b.\backslash w+VB\backslash b.(?:(\backslash w+IN\backslash b.)?)(?:(\backslash w+DT\backslash b.)?)\backslash w+NN\backslash b.\backslash w+CC\backslash b.\backslash w+NN\backslash b)$ ”

Extraction-Example The input sentence is:

“It must specify by name and category of string type.”

Applying this defined NLP rule to a given sentence under the following ratios.

- NN/PRP tag before the MD tag refers to the root of the multiple attribute element.
- Combination of MD and IN tag, and CC tag are used to extract the multiple attributes’ names.
- After the CC tag, the Preposition IN tag exists in a sentence that refers to the datatype of the attribute.

- **Rule No. 3** Consider a sentence that starts with two consecutive nouns of NN tag, or a single noun NN tag appears with a verb VBZ tag, which identifies the root of the multiple attributes. A PRP tag might appear instead of an NN tag at the start of a sentence. Those tags are followed by an optional determiner DT and a Noun NN tag associated with a WDT determiner tag followed by the combination of MD- VB tag together with an optional Determiner DT tag, and then the noun NN tag appears, corresponds to the set of multiple (two) attributes. Depicted in 3.5c.

Tagged Output It_PRP defines_VBZ a_DT name_NN that_WDT should_MD precede_VB a_DT category_NN of_IN string_NN type_NN.

Regular Expression Transformation “((\|w+NN\|b.\|w+NN\|b.\|w+NN\|b.\|w+PRP\|b.) (\|w+VBZ\|b.(?:(\|w+DT\|b.)?)\|w+NN\|b.\|w+WDT\|b.\|w+MD\|b.\|w+VB\|b.(?:(\|w+DT\|b.)?)\|w+NN\|b))”

Extraction-Example The input sentence is:

“It defines a name that should precede a category of string type.”

Applying this NLP rule to the given sentence under the set of following rationals.

- (a) NN/PRP before the VBZ tag refers to the root of the multiple attribute element.
 - (b) VBZ tag, the combined pattern of WDT and VB tag refers to the names of multiple attributes.
 - (c) After this, the IN tag is again defined in a sentence before two consecutive nouns referring to the datatype.
- **Rule No. 4** Consider a sentence that starts with two consecutive nouns NN or a single noun NN tag is defined with a verb VBZ tag, identifies the root of

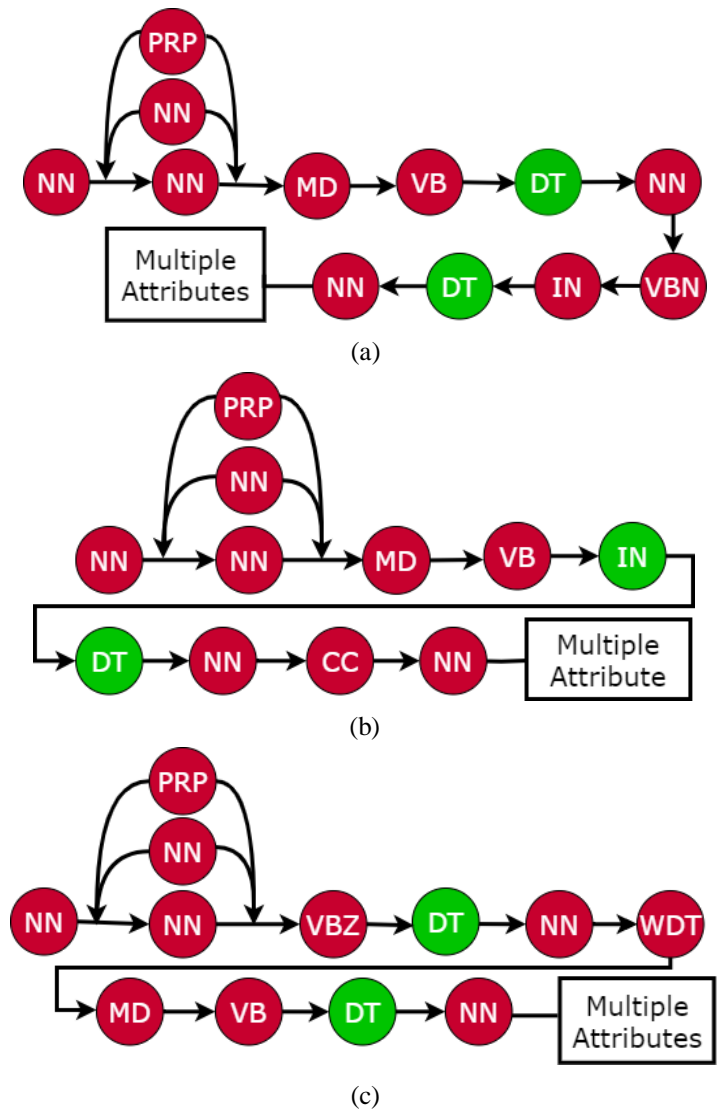


Figure 3.5: Graphical Description of Rules for the Multiple Attribute Element Identification.

the multiple attributes. There can be a PRP tag instead of an NN tag at the start of a sentence. Then those tags proceed with an optional adjective JJ tag, and noun NN tag followed by a VBG (gerund verb) tag or preposition IN tag. Then, those previous tags collectively proceed with a noun NN tag, CC tag, and then again a Noun NN tag, which identifies the multiple (two) attributes element. Shown in Figure 3.6a.

Tagged Output

- i) It_PRP has_VBZ attributes_NN including_VBG name_NN and_CC cate-

gory_NN of_IN string_NN type_NN.

- ii) It_PRP has_VBZ attributes_NN like_IN name_NN and_CC category_NN of_IN string_NN type_NN.

Regular Expression Transformation “((\|w+NN\|b.\|w+NN\|b.\|w+N N\|b.\|w+PRP\|b.)(\|w+VBZ\|b.(?:(\|w+JJ\|b.)?)\|w+NN\|b.(\|w+V BG\|b.\|w+IN\| b.)\|w+ NN\|b.\| w+CC\|b.\|w+NN\|b))”

Extraction Example The input sentence is:

“It has attributes including name and category of string type.”

Applying this defined NLP rule to the given sentence, the multiple attribute element is extracted under the following rationals.

- (a) NN/PRP before the VBZ tag refers to the root of the multiple attribute element.
 - (b) VBG and CC tag refers to the names of multiple attributes.
 - (c) After the pattern, the IN tag is defined in a sentence that refers to the datatypes.
- **Rule No. 5** Suppose the two consecutive nouns NN tag or a single Noun NN tag appears with the verb VBZ tag at the start of a sentence, considered as the root of the multiple attributes. A PRP tag might appear instead of the NN tag at the start of a sentence. Then, the combination of the VBN- IN tag is followed by the tags of an optional determiner DT, noun NN, CC, and two consecutive noun NN tags are defined in the same sentence corresponding to the set of multiple (two) attributes. Depicted in Figure 3.6b.

Tagged Output The DT EventFunction NN concept_NN is_VBZ represented_VBN by_IN name_NN and_CC category_NN attributes_NN of_IN string_NN type_NN.

Regular Expression Transformation “((\|w+NN\|b.\|w+NN\|b.\|w+

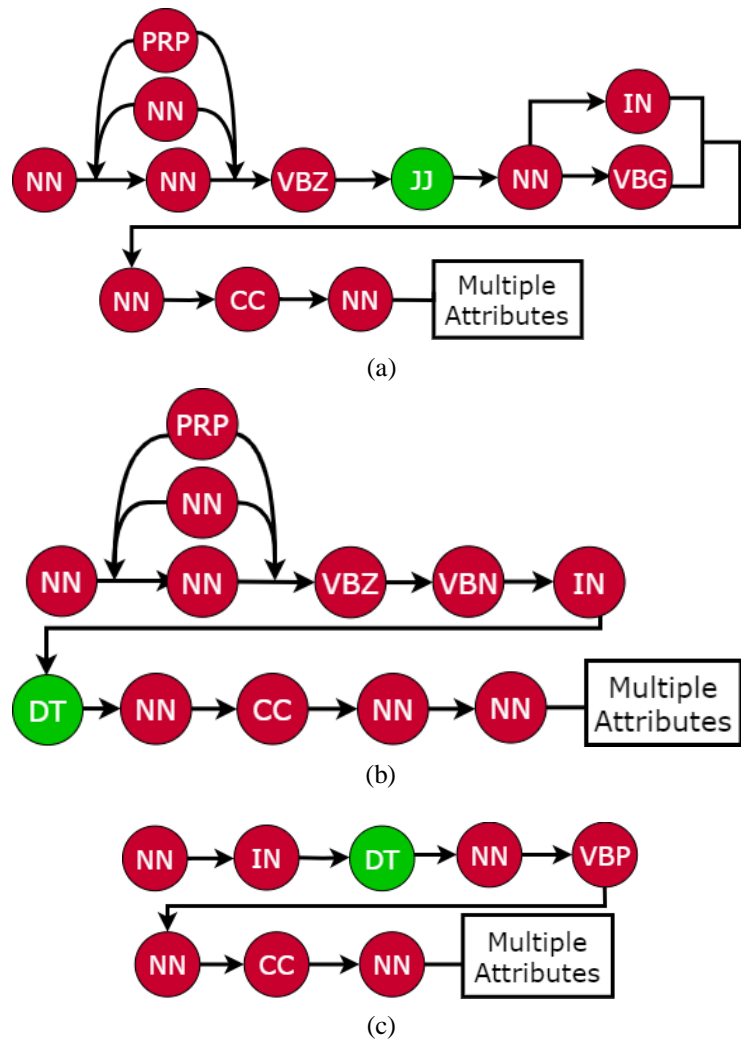


Figure 3.6: Graphical Description of Rules for the Multiple Attribute Element Identification.

Applying this NLP rule to extract the multiple attribute element with consideration of a few following rationals.

- (a) NN before the VBP tag refers to the root of the multiple attribute element.
- (b) VBP and CC tag are used to extract the attribute's names.
- (c) After the pattern, the IN tag is defined in a sentence that refers to the datatypes.

4. Rules for the identification of Single Attribute elements

The Xtext grammar can have single attributes. Therefore, the attribute element of

the single category starts with a parser rule followed by a colon notation, and then a keyword parameter appears. It is considered as the root of the single attribute element which is undefined in the case of PRP tag. After the declaration of this root element, the single (one) attribute with its datatypes is defined, i.e., STRING, etc. The textual example of the target Xtext grammar rule is listed below:

EventFunctionflowport: 'EventFunctionflowport'

FunctionFlowPort=STRING.

We define an NLP rule for the single attribute identification. Figure 3.7 represents the description of rules where the green circle defines the mandatory feature, and the green circle determines the optional feature, which may or may not be present.

- **Rule No. 1** Suppose a sentence starts with two consecutive nouns of the NN tag or a single noun of the NN tag followed by a VBZ and VBN tag, corresponding to the root of the single attribute element. There might be a PRP tag instead of the noun NN that appears at the start of a sentence. Further, those tags must be followed by two scenarios to identify the single attribute, as shown in Figure 3.7.
 - i) The first scenario is where the previous tags are followed by the TO and the VB (verb base form) tag associated with an optional determiner DT and two consecutive nouns NN.
 - ii) In the second scenario, the previous tags are followed by the IN tag and the VBG (gerund verb) tag associated with an optional determiner DT and two consecutive nouns NN.

Tagged Output

- i) It_PRP is_VBZ used_VBN to_TO define_VB a_DT FunctionFlowPort_NN attribute_NN of_IN string_NN type_NN.

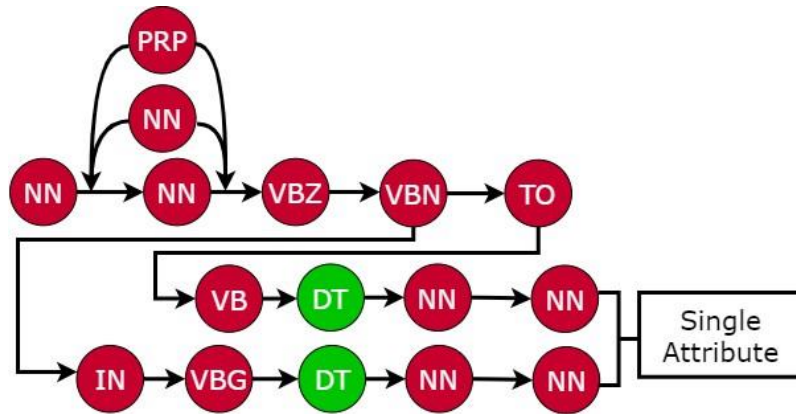


Figure 3.7: Graphical representation of rules for Single Attribute Element Identification.

- ii) It_PRP is_VBZ used_VBN for_IN defining_VBG a_DT FunctionFlowPort_NN attribute_NN of IN string NN type NN.

Regular Expression Transformation “((\\w+NN\\b.\\w+NN\\b.\\w+NN\\b.\\w+PRP\\b.)\\w+VBZ\\b.\\w+VBN\\b.(\\w+TO\\b.\\w+VB\\b.\\w+IN\\b.\\w+VBG\\b.)(?:\\w+DT\\b.)?)\\w+NN\\b.\\w+NN\\b)”

Extraction Example The input sentence is:

“It is used to define a FunctionFlowPort attribute of string type.”

By executing this NLP rule to the given sentence, the multiple attribute element is extracted under a few following rationals.

- NN/PRP before the VBZ tag refers to the root of the single attribute element.
- Combination of VBN and VB tag are used to extract the single attribute’s name.
- After the pattern, an IN tag is defined in a sentence that implies the datatypes.

5. Rules for the identification of Single & Optional Attribute elements

In the context of the Xtext grammar, both single and optional attributes can be

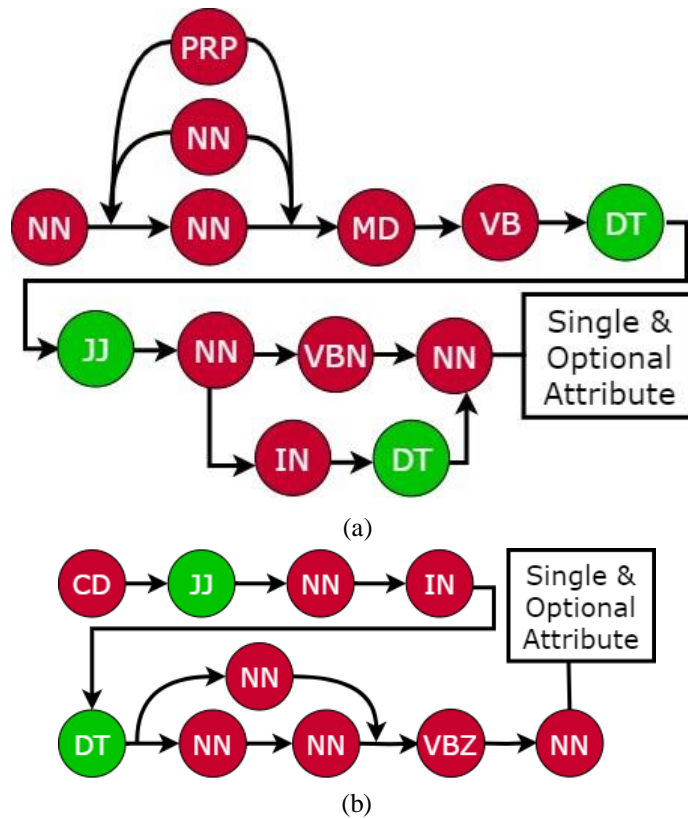


Figure 3.8: Graphical Description of Rules for Single & Optional Attribute Element Identification.

defined. Therefore, both types of attributes can initially start with a parser rule followed by a colon notation and a keyword parameter. It is known as the root of both types of attributes which is undefined in the case of PRP tag. The declaration of both attributes is defined as:

- i) Declaration of the single attribute with its datatype (STRING, etc).

such as FunctionFlowPort=STRING.

- ii) The optional attribute element is declared together with a question-mark notation and its datatype (STRING, etc).

such as (TraceableSpecification=STRING)?.

Two NLP rules are defined for the identification process to target both single and optional attributes. Figure 3.8 represents the description of rules where the red

circle determines the mandatory feature and the green circle determines the optional feature, which may or may not be present.

- **Rule No. 1** Consider a sentence that starts with two consecutive Nouns NN tags or a single NN tag that appears with a modal MD tag, which identifies the root of the attributes. There might be a PRP tag instead of a noun NN tag. Then, those tags are associated with a VB tag followed by an optional determiner DT tag and an optional adjective JJ tag, and then a Noun NN tag appears. Then, proceeds with one of the two scenarios to identify the set of single and optional attributes, as shown in Figure 3.8a.

- i) The first scenario is where those tags proceed with the VBN (past-participle verb) tag and a noun NN tag.
- ii) The second scenario is where those tags proceed with the preposition IN tag and an optional determiner DT tag defined before the noun NN tag.

Tagged Output

- i) It_PRP should_MD define_VB a_DT main_JJ attribute_NN like_IN a_DT FunctionFlowPort_NN of_IN string_NN type_NN.
- ii) It_PRP may_MD have_VB an_DT optional_JJ attribute_NN named_VBN TraceableSpecification_NN of_IN string_NN type_NN.
- iii) It_PRP should_MD define_VB an_DT attribute_NN named_VBN FunctionFlowPort_NN of_IN string_NN type_NN.

Regular Expression Transformation “ $((\backslash w+NN\backslash b.\backslash w+NN\backslash b.\backslash w+NN\backslash b.\backslash w+PRP\backslash b.)\backslash w+MD\backslash b.\backslash w+VB\backslash b.(?:(\backslash w+DT\backslash b.)?)(?:(\backslash w+JJ\backslash b.)?)\backslash w+NN\backslash b.(\backslash w+VBN\backslash b.\backslash w+NN\backslash b.\backslash w+IN\backslash b.(?:(\backslash w+DT\backslash b.)?))\backslash w+NN\backslash b)$ ”

Extraction Example The input sentence is:

“It may have an optional attribute named TraceableSpecification of string type.”

Executing this NLP rule to the given sentence, the optional attribute element is extracted under a few following rationals.

- (a) NN/PRP before the MD tag refers to the root of an optional attribute.
 - (b) Combined patterns of MD and VBN tags are used to extract the name of an optional attribute.
 - (c) JJ tag refers to the optional tag.
 - (d) After the pattern, an IN tag is defined in a sentence that implies the datatypes.
- **Rule No. 2** Consider a sentence that starts with a cardinal number CD tag, as shown in Figure 3.8b, followed by the following tags: Suppose the previously identified tag appears with an optional adjective JJ tag and a Noun NN tag, followed by a preposition IN, an optional determiner DT, and a noun NN tag, which identifies the root element. Then, it appears with a verb VBZ and a noun tag, which targets the identification process of optional and single attributes.

Tagged Output

- i) One_CD optional_JJ attribute_NN of_IN the_DT timing_NN model_NN is_VBZ TraceableSpecification_NN of_IN string_NN type_NN.
- ii) One_CD attribute_NN of_IN EventFunctionflowport_NN is_VBZ FunctionFlowPort_NN of_IN string_NN type_NN.

Regular Expression Transformation “ $(\backslash w+CD\backslash b.(?:\backslash w+JJ\backslash b.)?)\backslash w+NN\backslash b.\backslash w+IN\backslash b.(?:\backslash w+DT\backslash b.)?)\backslash w+NN\backslash b.*\backslash w+VBZ\backslash b.\backslash w+NN\backslash b)$ ”

Extraction Example The input sentence is:

“One attribute of EventFunctionflowport is FunctionFlowPort of string type.”

Executing this NLP rule to the given sentence, the optional attribute element is extracted under a few following rationals.

- (a) Combination of CD and IN tag containing the NN before the VBZ tag refers to the root of the single attribute element.
- (b) VBZ tag implies the name of a single attribute.
- (c) After the pattern, an IN tag is defined in a sentence that implies the datatypes.

Chapter 4

Implementation

This chapter presents the implementation details required to understand the underlying terminologies of the tool. **Section 4.1** provides the details of libraries, tools, and languages used for the implementation. **Section 4.2** presents the interface elements of the implemented tool. **Section 4.3** discusses the Xtext grammar generation in detail.

Tools and Languages

Then, the text is passed for minor preprocessing to attain the structured format. This section describes the underlying implementation details, such as tools, libraries, and languages used to implement the proposed framework. The NL2DSL tool is based on the proposed algorithm implemented using the Java language within the Eclipse IDE framework. Figure 4.1 depicts the interface of the Eclipse platform. The NL2DSL tool takes the input of a PDF document containing the textual requirements of the system. The generated Xtext grammar by the NL2DSL tool is viewed with the .xtext DSL file. The process is summarized as follows:

- The textual requirements of the systems within a PDF file are given as input to the NL2DSL tool. Then, the text is passed for minor preprocessing to attain the

```

import javax.swing.*;

public class GUI_Project
{
    public static void main(String[] args)
    {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                try {
                    createAndShowGUI();
                }
                catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        });
    }

    static void createAndShowGUI() throws IOException
    {
        nlp_rules.nestedclass nestedObj = new nlp_rules.nestedclass();
        Font fontl=new Font("Arial", Font.BOLD,15);
        Font fontc=new Font("Arial", Font.LAYOUT_LEFT_TO_RIGHT, 13);

        FlatLightLaf.setup(); //setting the look and feel
        JFrame.setDefaultLookAndFeelDecorated(true);

        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}

```

```

GUI_Project (1) [Java Application] C:\Users\AminaSoftware\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.7
Expression:: system presents a scenario of timing model
Matched Rule: (\w+NN\b.\w+VBZ\b.\w+DT\b.\w+NN\b.\w+IN\b.(\w+JJ\b.\w+NN\b)|\w+NN\b.\w+NN\b)
Matched statement: java.util.regex.Matcher[pattern=\w+NN\b.\w+NN\b region=0,64 lastmatch=]
Matched statement: java.util.regex.Matcher[pattern=\w+JJ\b.\w+NN\b region=0,64 lastmatch=]
System Has:timing model
#####
added in xtext aGGS:timing_model:'timing_model'

```

Figure 4.1: Interface of the Eclipse Platform.

structured format by removing punctuation marks.

- Maxnet Tagger by the Stanford CoreNLP library [51] is employed to identify the phrases of nouns, verbs, adjectives, etc from the requirements.
- Java Regex API library [52] is employed to implement the proposed NLP rules in the form of regular expressions that support the desired extraction of the Xtext DSL elements with the string-matching technique.
- The series of if-else statements controls the extraction process to match the relevant tags of each word and match the string expressions with the proposed NLP rules to support the desired extraction of the Xtext DSL elements.
- The results of the extracted elements of the Xtext grammar are stored in an array data structure and added to the DSL file of the .xtext extension with a few post-processing operations. The process of the NL2DSL is fully automated. Hence no manual interruption is required.

Tool Interface

Figure 4.2 depicts the user interface of the NL2DSL tool. The tool inputs a PDF file containing the textual requirements of the system to generate the output of the Xtext grammar. From Figure 4.2, it is observed that the tool has multiple buttons that are intended to perform different functions. The main functionalities of the buttons are as follows.

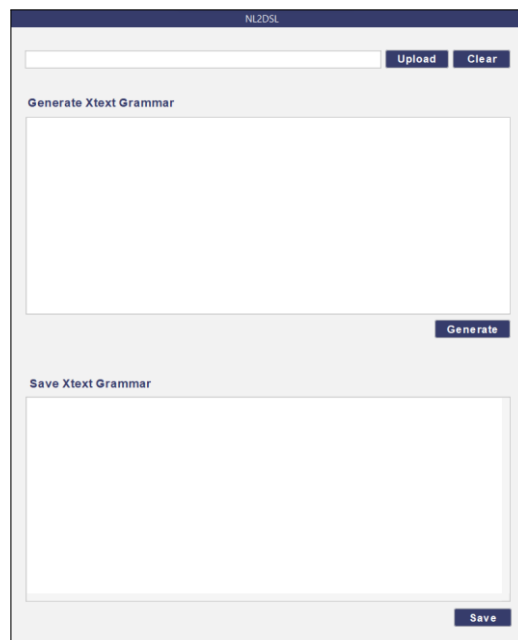


Figure 4.2: Interface of the NL2DSL Tool.

Upload

By pressing the 'upload' button, the tool inputs a PDF file that contains the intended text added to the top textbox. The tool also locates the location of the selected input file added to the top text field.

Generate

Pressing this button generates the Xtext grammar from the texts, and the result is added to the below textbox.

Save

By pressing the save button, the results of the generated Xtext grammar are saved to the user-specified location of the device in a DSL file of the .xtext extension.

Clear

When this button is clicked, the generated result of the Xtext grammar from the below textbox clears.

Xtext Grammar Generation Details

The basic steps required to operate the NL2DSL tool are already discussed at the start of Chapter 4. So, these steps must be understood to generate the Xtext grammar from the requirements. Each button has its own functionality, so its details are provided. The tool has to input a PDF file of textual requirements to generate the Xtext grammar from that file.

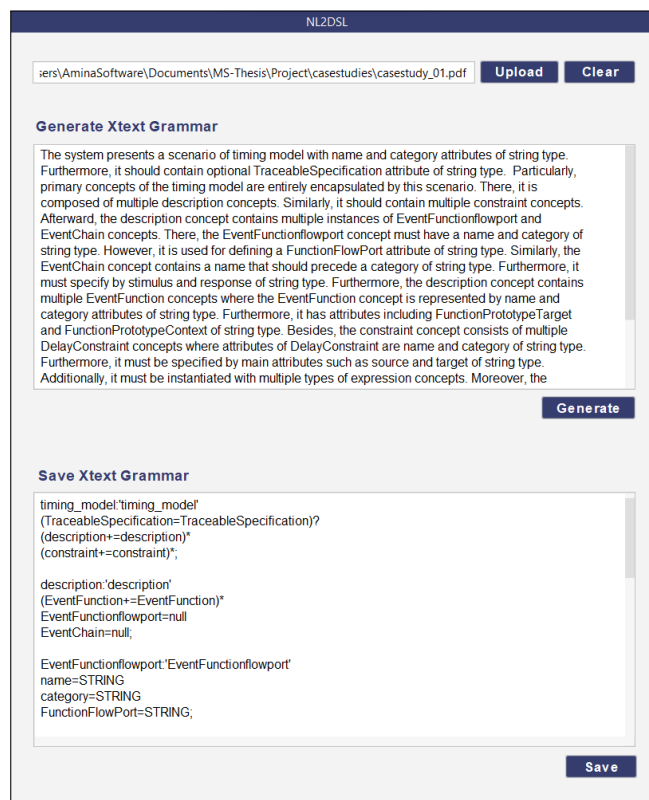


Figure 4.3: NL2DSL's Output comprising Xtext Grammar.

By clicking the upload button, the tool will ask you to input a PDF file from the specific location of your device. When the file is successfully uploaded, then the output is generated by clicking the generate button. Figure 4.3 shows the generated Xtext grammar in the output area of the tool. Now, the Xtext grammar is generated by the tool from the uploaded PDF file. This is the desired output of the NLP rules. The generated Xtext grammar in the output area of the tool can be saved to the desired location of the device for future use. From Figure 4.4, it is observed that the NL2DSL tool has a save button that is used to perform this desired operation. It clearly indicates that by clicking the save button, the desired location of the device is asked at which you want to save the output file with the file extension of .xtext. Therefore, non-technical experts can use this file to understand the requirements of complex systems while technical experts can use it for the efficient development of software systems. Also, it alleviates the additional burden of the technical experts with the auto-generated Xtext grammar.

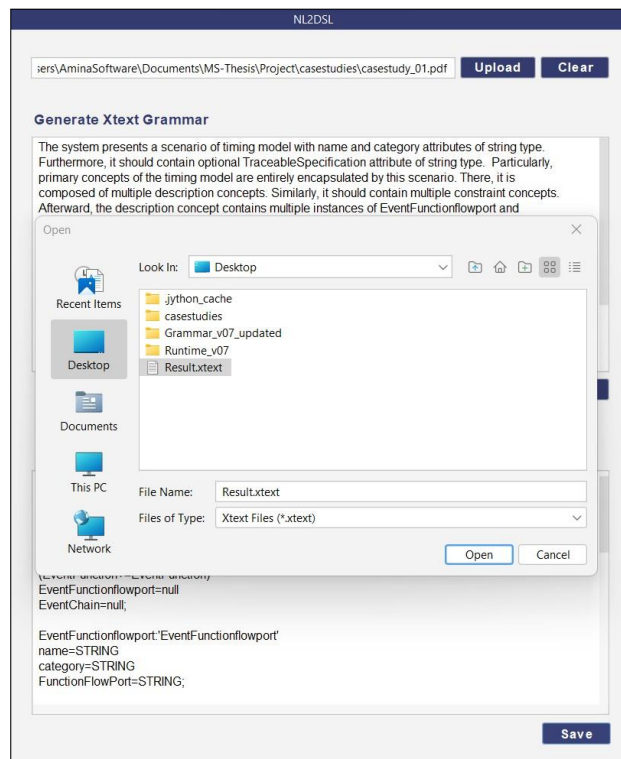


Figure 4.4: Generated Xtext Grammar Save in .xtext.

Chapter 5

Validation

Within this chapter, the validation of the proposed framework is presented in detail with the case studies and generated results. The main aim of this chapter is to provide a proof-of-concept in order to demonstrate the viability of the proposed framework. This chapter is organized into two sections. **Section 5.1** provides the details of case studies. Subsequently, **Section 5.2** presents the results of the generated Xtext grammar.

Dataset Collection

Initially, analytical reasoning is conducted to determine the structural representation of systems in order to define the functional concepts. Thus, the textual requirements consisting of the functional concepts are defined as a case study with the analysis results as per the rules to write the requirements. We defined two case studies: one is the timing model associated with the Volvo industry, and the other one is the diabetic manager associated with the health industry. Further, it is analyzed from the research gap (presented in Chapter 2 that there is an ongoing necessity for a framework of automated Xtext grammar. Therefore, we can utilize these two case studies to prove the efficiency of our proposed framework.

It is noted that the initial state of the textual system requirements requires some preprocessing tasks to make it capable of implementing the NLP rules to generate the Xtext grammar. Preprocessing is required to perform before being fed into the proposed algorithm. The texts may include punctuation marks that can change the syntactic meaning. Before the algorithm's operation, the text must be cleansed to generate the Xtext grammar. This has been discussed in detail in Chapter 3.

Case Study 01

The requirements of the timing model case study are taken from the prior research study [53] containing the EastADL specifications, associated with the Volvo industry. Particularly, the requirements of the 'Timing Model' cover the details of developing reliable embedded systems. The textual requirements of the timing model case study are presented in Figure 5.1.

The system presents a scenario of timing model with name and category attributes of string type. Furthermore, it should contain optional TraceableSpecification attribute of string type. Particularly, primary concepts of the timing model are entirely encapsulated by this scenario. There, it is composed of multiple description concepts. Similarly, it should contain multiple constraint concepts. Afterward, the description concept contains multiple instances of EventFunctionflowport and EventChain concepts. There, the EventFunctionflowport concept must have a name and category of string type. However, it is used for defining a FunctionFlowPort attribute of string type. Similarly, the EventChain concept contains a name that should precede a category of string type. Furthermore, it must specify by stimulus and response of string type. Furthermore, the description concept contains multiple EventFunction concepts where the EventFunction concept is represented by name and category attributes of string type. Furthermore, it has attributes including FunctionPrototypeTarget and FunctionPrototypeContext of string type. Besides, the constraint concept consists of multiple DelayConstraint concepts where attributes of DelayConstraint are name and category of string type. Furthermore, it must be specified by main attributes such as source and target of string type. Additionally, it must be instantiated with multiple types of expression concepts. Moreover, the constraint concept contains AgeConstraint which should be a concept described with multiple instances where the AgeConstraint concept has various elements like name and category attributes of string type. Furthermore, it is used for defining a scope attribute of string type. Additionally, it should only contain multiple expression concepts. Particularly, the expression concept is represented by name and value attributes of string type. Furthermore, one main attribute of expression is type of string type.

Figure 5.1: Timing Model Case Study.

Case Study 02

The feasibility of the proposed framework is proved with the help of another case study, that is based on the general scenario of diabetes associated with the health industry. Particularly, the textual requirements of this case study cover diabetes management through controlling the symptoms. The basic information about diabetes is elicited from the web portal of the National Library of Medicine [54]. The textual requirements of the diabetic manager as a case study are presented in Figure 5.2.

The scenario of system can be a diabetic manager which is presented with a category attribute of string type. Particularly, it is required to define each health concept by the scenario of this system. There, it should contain primary concepts of diabetes, such as the symptom concept with multiple instances. Then, the symptom concept must have a name and severity of string type. Particularly, the common symptom of diabetes are fatigue or hyperglycemia, which are highly linked to developing diabetes. Furthermore, the system consists of multiple patient concepts where each patient concept is used to define a glucose attribute of string type. Furthermore, one optional attribute of the patient is name of string type. Furthermore, it should have an attribute like AnyDisease of a boolean datatype. However, it is linked with multiple symptom concepts. Additionally, the system contains multiple doctor concepts where the doctor concept must have a specialization and category of string type. Moreover, it refers to the multiple instances of patient concepts. Furthermore, it should contain multiple instances of the medicine concept where the medicine concept contains multiple attribute elements like name and dose of string type.

Figure 5.2: Diabetic Manager Case Study.

Results

Various NLP rules have been applied to each case study in order to generate the Xtext grammar. For each case study, the results of the Xtext grammar are separately presented in subsequent sections. The rules have been implemented within a sequence of if-else conditions employed by the for-loop. Therefore, a sentence pattern containing the POS tags synchronizes and matches with the rule pattern, so the DSL element conforming to the Xtext grammar is extracted. The iteration of the loop ends thereafter generating the desired DSL element of the Xtext grammar, and then the next iteration begins to check the next sentence patterns with the rules, and so on. In this sequence, the Xtext grammar is generated from the textual requirement of the case studies. Based on the prior research study [14], we performed the performance evaluation of the NL2DSL tool. The

evaluation of the NL2DSL tool is determined for each case study which is conducted with a few evaluation metrics like Precision, Recall, and Over-Specification based on some underlying parameters.

The description of each underlying parameter is described below:

1. **Correct** represents those Xtext DSL elements correctly identified by the human experts' analysis process and NL2DSL tool.
2. **Incorrect** are those Xtext DSL elements that are correctly identified by the human experts' analysis process, but the NL2DSL tool incorrectly generated them.
3. **Missing** are those Xtext DSL elements that are considered to be a part of the case study, but the NL2DSL tool fails to generate them.
4. **Extra** Extra are such Xtext DSL elements that are additionally generated by the NL2DSL tool beyond the corrected DSL elements of the Xtext.

According to these given parameters, the performance evaluation of the NL2DSL tool is conducted for each case study to determine its effectiveness in terms of generated Xtext grammar. And, the calculation formula for each evaluation metric is defined as follows:

$$\text{Precision} = \frac{\text{Correct}}{\text{Correct} + \text{Incorrect}}$$

$$\text{Recall} = \frac{\text{Correct}}{\text{Correct} + \text{Missing}}$$

$$\text{Over Specification} = \frac{\text{Extra}}{\text{Correct} + \text{Missing}}$$

Results of Case Study 01

The generated Xtext grammar is saved in a DSL file of the .xtext extension, depicted in Figure 5.3 for the timing model case study. Particularly, the primary DSL elements of the Xtext such as root element, relationship, and different types of attributes are generated from the timing model case study. To determine the performance of the NL2DSL tool, firstly we conducted a human experts' analysis of the timing model case study to identify

the actual type of Xtext DSL elements, presented in Table 5.1.

Table 5.1: List of Actual Xtext DSL Elements for Timing Model Case Study.

Req.#	Textual Requirement	Root Element	Relationships	Attributes		
				Multiple	Single	Optional
1	The system presents a scenario of timing model with name and category attributes of string type.	timing model: 'timing_model'		name=STRING category=STRING		
2	Furthermore, it should contain optional TraceableSpecification attribute of string type.					(TraceableSpecification=STRING)?
3	Particularly, primary concepts of the timing model are entirely encapsulated by this scenario.					
4	There, it is composed of multiple description concepts.		(description+=description)*			
5	Similarly, it should contain multiple constraint concepts.		(constraint+=constraint)*			
6	Afterward, the description concept contains multiple instances of EventFunctionflowport and EventChain concepts.		description:'description' (EventFunctionflowport+= EventFunctionflowport)* (EventChain+= EventChain)*			
7	There, the EventFunctionflowport concept must have a name and category of string type.			EventFunctionflowport: 'EventFunctionflowport' name=STRING category=STRING		
8	However, it is used for defining a FunctionFlowPort attribute of string type.				FunctionFlowPort =STRING	
9	Similarly, the EventChain concept contains a name that should precede a category of string type.			EventChain:'EventChain' name=STRING category=STRING		
10	Furthermore, it must specify by stimulus and response of string type.			stimulus=STRING response=STRING		
11	Furthermore, the description concept contains multiple EventFunction concepts where the EventFunction concept is represented by name and category attributes of string type.		(EventFunction+= EventFunction)*	EventFunction:'EventFunction' name=STRING category=STRING		
12	Furthermore, it has attributes including FunctionPrototypeTarget and FunctionPrototypeContext of string type.			FunctionPrototypeTarget =STRING FunctionPrototypeContext =STRING		
13	Besides, the constraint concept consists of multiple DelayConstraint concepts where attributes of DelayConstraint are name and category of string type.		constraint:'constraint' (DelayConstraint+= DelayConstraint)*	DelayConstraint: 'DelayConstraint' name=STRING category=STRING		
14	Furthermore, it must be specified by main attributes such as source and target of string type.			source=STRING target=STRING		
15	Additionally, it must be instantiated with multiple types of expression concepts.		(expression+=expression)*			
16	Moreover, the constraint concept contains AgeConstraint which should be a concept described with multiple instances where the AgeConstraint concept has various elements like name and category attributes of string type.		(AgeConstraint+= AgeConstraint)*	AgeConstraint: 'AgeConstraint' name=STRING category=STRING		
17	Furthermore, it is used for defining a scope attribute of string type.				scope=STRING	
18	Additionally, it should only contain multiple expression concepts.		(expression+=expression)*			
19	Particularly, the expression concept is represented by name and value attributes of string type.			expression:'expression' name=STRING value=STRING		
20	Furthermore, one main attribute of expression is type of string type.				type=STRING	

Then, the DSL elements conforming to the Xtext grammar are identified by the NL2DSL tool, presented in Table 5.2.

Table 5.2: List of Xtext DSL Elements for Timing Model Case Study Identified by NL2DSL tool.

Req.#	Textual Requirement	Root Element	Relationships	Attributes		
				Multiple	Single	Optional
1	The system presents a scenario of timing model with name and category attributes of string type.	timing model: 'timing_model'				
2	Furthermore, it should contain optional TraceableSpecification attribute of string type.		TraceableSpecification =TraceableSpecification)?			
3	Particularly, primary concepts of the timing model are entirely encapsulated by this scenario.					
4	There, it is composed of multiple description concepts.		(description+=description)*			
5	Similarly, it should contain multiple constraint concepts.		(constraint+= constraint)*			
6	Afterward, the description concept contains multiple instances of EventFunctionflowport and EventChain concepts.			description:'description' EventFunctionflowport=null EventChain= null		
7	There, the EventFunctionflowport concept must have a name and category of string type.			EventFunctionflowport: 'EventFunctionflowport' name=STRING category=STRING		
8	However, it is used for defining a FunctionFlowPort attribute of string type.				FunctionFlowPort =STRING	
9	Similarly, the EventChain concept contains a name that should precede a category of string type.			EventChain:'EventChain' name=STRING category=STRING		
10	Furthermore, it must specify by stimulus and response of string type.			stimulus=STRING response=STRING		
11	Furthermore, the description concept contains multiple EventFunction concepts where the EventFunction concept is represented by name and category attributes of string type.		(EventFunction+= EventFunction)*	EventFunction:'EventFunction' name=STRING category=STRING		
12	Furthermore, it has attributes including FunctionPrototypeTarget and FunctionPrototypeContext of string type.			FunctionPrototypeTarget =STRING FunctionPrototypeContext =STRING		
13	Besides, the constraint concept consists of multiple DelayConstraint concepts where attributes of DelayConstraint are name and category of string type.			DelayConstraint: 'DelayConstraint' name=STRING category=STRING		
14	Furthermore, it must be specified by main attributes such as source and target of string type.					
15	Additionally, it must be instantiated with multiple types of expression concepts.					
16	Moreover, the constraint concept contains AgeConstraint which should be a concept described with multiple instances where the AgeConstraint concept has various elements like name and category attributes of string type.			constraint:'constraint' AgeConstraint=MULTIP concept=MULTIP AgeConstraint: 'AgeConstraint' name=STRING category=STRING		
17	Furthermore, it is used for defining a scope attribute of string type.				scope=STRING	
18	Additionally, it should only contain multiple expression concepts.		(expression+=expression)*			
19	Particularly, the expression concept is represented by name and value attributes of string type.			expression:'expression' name=STRING value=STRING		
20	Furthermore, one main attribute of expression is type of string type.				type=STRING	

Afterward, a comparative analysis with the actual and generated results by the NL2DSL tool is performed to determine which generated Xtext DSL elements are classified as correct, incorrect, missing, and extra elements.

This step leads to calculating the performance of the NL2DSL tool with various evaluation metrics, such as Precision, Recall, and Over-Specification, presented in Table 5.3 for the timing model case study.

Table 5.3: Calculation of NL2DSL Effectiveness for Timing Model Case Study.

	Root Element	Relationships	Attributes			Total	Precision	Recall	Over Specifications
			Multiple	Single	Optional				
Correct	1	4	8	3	0	16	84.21%	80%	0%
Incorrect	0	1	2	0	0	3			
Missing	0	2	2	0	0	4			
Extra	0	0	0	0	0	0			

```

timing_model: 'timing_model'
(TraceableSpecification=TraceableSpecification)?
(description+=description)*
(constraint+=constraint)*;

description: 'description'
(EventFunction+=EventFunction)*
EventFunctionflowport=null
EventChain=null;

EventFunctionflowport: 'EventFunctionflowport'
name=STRING
category=STRING
FunctionFlowPort=STRING;

EventChain: 'EventChain'
name=STRING
category=STRING
stimulus=STRING
response=STRING;

EventFunction: 'EventFunction'
name=STRING
category=STRING
FunctionPrototypeTarget=STRING
FunctionPrototypeContext=STRING;

DelayConstraint: 'DelayConstraint'
name=STRING
category=STRING;

constraint: 'constraint'
AgeConstraint=MULTIP
concept=MULTIP;

AgeConstraint: 'AgeConstraint'
name=STRING
category=STRING
scope=STRING
(expression+=expression)*;

expression: 'expression'
name=STRING
value=STRING
type=STRING;

```

Figure 5.3: Generated Xtext Timing Model Grammar.

The effectiveness of the proposed framework’s methodology is also determined by inducing some variations within the textual requirements of the timing model case study. This step determines that the NL2DSL tool has the flexibility to incorporate the variated text in order to generate the Xtext grammar. The results of the human experts’ analysis are

presented in Table 5.4 to identify the actual type of Xtext DSL elements from the varied timing model case study. Subsequently, the list of the DSL elements of the Xtext is presented in Table 5.5, which is identified by the NL2DSL tool from the varied requirements of the timing model case study.

Table 5.4: List of Actual Xtext DSL Elements for Varied Timing Model Case Study.

Req.#	Textual Requirement	Root Element	Relationships	Attributes		
				Multiple	Single	Optional
1	The scenario of system can be a timing model, where it must have different concepts with the inclusion of primary attributes.	timing_model: 'timing_model'				
2	Furthermore, it must have a name preceded by a category of string type.			name=STRING category=STRING		
3	Furthermore, it may have an optional attribute named TraceableSpecification of string type.					TraceableSpecification=STRING)?
4	There, it is composed of multiple description concepts.		(description+= description)*			
5	Similarly, it should contain multiple constraint concepts.		(constraint+= constraint)*			
6	Afterward, the description concept contains multiple instances of EventFunctionflowport and EventChain concepts.		description:'description' (EventFunctionflowport+= EventFunctionflowport)* (EventChain+= EventFunction)*			
7	There, the EventFunctionflowport concept must have a name and category of string type.			EventFunctionflowport: 'EventFunctionflowPort' name=STRING category=STRING		
8	However, it is used for defining a FunctionFlowPort attribute of string type.					FunctionFlowPort=STRING
9	Similarly, the EventChain concept contains a name that should precede a category of string type.			EventChain:'EventChain' name=STRING category=STRING		
10	Furthermore, it must specify by stimulus and response of string type.			stimulus=STRING response=STRING		
11	Furthermore, the description concept is composed of multiple EventFunction concepts where the EventFunction concept is represented by name and category attributes of string type.		(EventFunction+= EventFunction)*	EventFunction:'EventFunction' name=STRING category=STRING		
12	Furthermore, it has attributes including FunctionPrototypeTarget and FunctionPrototypeContext of string type.			FunctionPrototypeTarget=STRING FunctionPrototypeContext=STRING		
13	Besides, the constraint concept consists of multiple DelayConstraint concepts where attributes of DelayConstraint are name and category of string type.		constraint:'constraint' (DelayConstraint+= DelayConstraint)*	DelayConstraint: 'DelayConstraint' name=STRING category=STRING		
14	Furthermore, it must have a source preceded by a target of string type.			source=STRING target=STRING		
15	Additionally, it should contain multiple expression concepts.		(expression+= expression)*			
16	Moreover, the constraint concept contains multiple AgeConstraint concepts where the AgeConstraint concept has various elements like name and category attributes of string type.		(AgeConstraint+= AgeConstraint)*	AgeConstraint: 'AgeConstraint' name=STRING category=STRING		
17	Furthermore, it contains a scope attribute of string type.					scope=STRING
18	Additionally, it should contain primary concepts, like an expression with multiple instances.		(expression+= expression)*			
19	Particularly, the expression concept is represented by name and value attributes of string type.			expression:'expression' name=STRING value=STRING		
20	Furthermore, one attribute of expression is type of string type.					type=STRING

Table 5.5: List of Xtext DSL Elements for Variated Timing Model Case Study Identified by NL2DSL tool.

Req.#	Textual Requirement	Root Element	Relationships	Attributes		
				Multiple	Single	Optional
1	The scenario of system can be a timing model, where it must contain different concepts with the inclusion of primary attributes.	timing_model: 'timing_model'			inclusion= null	
2	Furthermore, it must have a name preceded by a category of string type.			name=STRING category=STRING		
3	Furthermore, it may have an optional attribute named TraceableSpecification of string type.					TraceableSpecification =STRING)?
4	There, it is composed of multiple description concepts.		(description+= description)*			
5	Similarly, it should contain multiple constraint concepts.		(constraint+= constraint)*			
6	Afterward, the description concept contains multiple instances of EventFunctionflowport and EventChain concepts.			description:'description' EventFunctionflowport=null EventChain=null		
7	There, the EventFunctionflowport concept must have a name and category of string type.			EventFunctionflowport: 'EventFunctionflowPort' name=STRING category=STRING		
8	However, it is used for defining a FunctionFlowPort attribute of string type.				FunctionFlowPort =STRING	
9	Similarly, the EventChain concept contains a name that should precede a category of string type.			EventChain:'EventChain' name=STRING category=STRING		
10	Furthermore, it must specify by stimulus and response of string type.			stimulus=STRING response=STRING		
11	Furthermore, the description concept is composed of multiple EventFunction concepts where the EventFunction concept is represented by name and category attributes of string type.		(EventFunction+= EventFunction)*	EventFunction:'EventFunction' name=STRING category=STRING		
12	Furthermore, it has attributes including FunctionPrototypeTarget and FunctionPrototypeContext of string type.			FunctionPrototypeTarget =STRING FunctionPrototypeContext =STRING		
13	Besides, the constraint concept consists of multiple DelayConstraint concepts where attributes of DelayConstraint are name and category of string type.			DelayConstraint: 'DelayConstraint' name=STRING category=STRING		
14	Furthermore, it must have a source preceded by a target of string type.			source=STRING target=STRING		
15	Additionally, it should contain multiple expression concepts.		(expression+= expression)*			
16	Moreover, the constraint concept contains multiple AgeConstraint concepts where the AgeConstraint concept has various elements like name and category attributes of string type.		constraint:'constraint' (AgeConstraint+= AgeConstraint)*	AgeConstraint: 'AgeConstraint' name=STRING category=STRING		
17	Furthermore, it contains a scope attribute of string type.					
18	Additionally, it should contain primary concepts, like an expression with multiple instances.				expression=null	
19	Particularly, the expression concept is represented by name and value attributes of string type.			expression:'expression' name=STRING value=STRING		
20	Furthermore, one attribute of expression is type of string type.				type=STRING	

Previously, we calculated the performance of the NL2DSL tool for the timing model case study. Similarly, we calculated the performance of the NL2DSL tool for the varied timing model case study. Therefore, the performance evaluation results of the NL2DSL tool with different metrics like Precision, Recall, and Over-Specification for the varied

timing model case study are presented in Table 5.6.

Table 5.6: Calculation of NL2DSL Effectiveness for Variated Timing Model Case Study.

	Root Element	Relationships	Attributes			Total	Precision	Recall	Over Specifications
			Multiple	Single	Optional				
Correct	1	5	10	2	1	19	90.47%	90.47%	4.76%
Incorrect	0	0	1	1	0	2			
Missing	0	1	0	1	0	2			
Extra	0	0	0	1	0	1			

Afterward, the accumulative performance evaluation of the NL2DSL tool for the timing model case study is calculated, i.e., actual and variated. This step mainly leads to determining the overall performance of the NL2DSL tool for the timing model case study.

Table 5.2 and Table 5.5 shows that total Xtext DSL elements generated by the NL2DSL tool are $23+24 = 47$

The correct type of Xtext DSL elements generated by the NL2DSL tool are $16+19 = 35$

The incorrect type of Xtext DSL elements generated by the NL2DSL tool are $= 3+2 = 5$

Missing Xtext DSL elements generated by the NL2DSL tool are $4+2 = 6$

Extra Xtext DSL elements generated by the NL2DSL tool are $0+1 = 1$

$$\text{Precision} = \frac{\text{Correct}}{\text{Correct} + \text{Incorrect}}$$

$$\text{So, Precision} = \frac{35}{35+5} = 87.5\%$$

$$\text{Recall} = \frac{\text{Correct}}{\text{Correct} + \text{Missing}}$$

$$\text{So, Recall} = \frac{35}{35+6} = 85.36\%$$

$$\text{Over Specification} = \frac{\text{Extra}}{\text{Correct} + \text{Missing}}$$

$$\text{So, Over Specification} = \frac{1}{35+6} = 2.43\%$$

Results of Case Study 02

We specified another case study which is a diabetic manager to prove the validation of the proposed framework. The output of the generated Xtext grammar by the NL2DSL tool for the diabetic manager case study is presented in Figure 5.4. Particularly, the primary DSL elements of the Xtext such as root element, relationship, and different types of attributes are generated from the diabetic manager case study.

```
system: 'diabetic_manager'  
(doctor+=doctor) *  
health=null  
diabetes=null;  
  
symptom: 'symptom'  
name=STRING  
severity=STRING;  
  
diabetes: 'diabetes'  
fatigue=null  
hyperglycemia=null;  
  
patient: 'patient'  
glucose=STRING  
(name=STRING) ?  
(AnyDisease?='AnyDisease') ?  
(symptom+=[symptom]) *;  
  
doctor: 'doctor'  
specialization=STRING  
category=STRING  
medicine=null;  
  
medicine: 'medicine'  
(attribute+=attribute) *;
```

Figure 5.4: Generated Xtext Diabetic Manager.

Recall from the previous subsection, the performance evaluation of the NL2DSL tool with various evaluation metrics, i.e., Precision, Recall, and Over-Specification is determined by comparing the actual results of the human experts' analysis with the results of the generated Xtext grammar by the NL2DSL tool for the timing model case study. Similarly, we conducted a human experts' analysis of the diabetic manager case study to identify the actual types of Xtext DSL elements, presented in Table 5.7.

Then, the DSL elements of the Xtext are identified by the NL2DSL tool, presented in Table 5.8 for the diabetic manager case study. Then, a comparative analysis is performed to determine which generated Xtext DSL elements of the diabetic manager case study are classified as correct, incorrect, missing, and extra.

Table 5.7: List of Actual Xtext DSL Elements for the Diabetic Manager Case Study.

Req.#	Textual Requirement	Root Element	Relationships	Attributes		
				Multiple	Single	Optional
1	The scenario of system can be a diabetic manager which is presented with a category attribute of string type.	system:'diabetic manager'			category=STRING	
2	Particularly, it is required to define each health concept by the scenario of this system.					
3	There, it should contain primary concepts of diabetes, such as the symptom concept with multiple instances.		(symptom+= symptom)*			
4	Then, the symptom concept must have a name and severity of string type.			symptom:'symptom'	name=STRING severity=STRING	
5	Particularly, the common symptom of diabetes are fatigue or hyperglycemia, which are highly linked to developing diabetes.					
6	Furthermore, the system consists of multiple patient concepts where each patient concept is used to define a glucose attribute of string type.		(patient+= patient)*		patient:'patient'	glucose=STRING
7	Furthermore, one optional attribute of the patient is name of string type.					(name=STRING)?
8	Furthermore, it should have an attribute like AnyDisease of a boolean datatype.				(AnyDisease?= 'AnyDisease')?	
9	However, it is linked with multiple symptom concepts.		(symptom+= [symptom])*			
10	Additionally, the system contains multiple doctor concepts where the doctor concept must have a specialization and category of string type.		(doctor+= doctor)*	doctor:'doctor'	specialization=STRING category=STRING	
11	Moreover, it refers to the multiple instances of patient concepts.		(patient+= [patient])*			
12	Furthermore, it should contain multiple instances of the medicine concept where the medicine concept contains multiple attribute elements like name and dose of string type.		(medicine+= medicine)*	medicine:'medicine'	name=STRING dose=STRING	

With the specification of these parameters, the performance of the NL2DSL tool is calculated with various performance evaluation metrics, such as Precision, Recall, and Over-Specification, which is presented in Table 5.9 for the diabetic manager case study.

As we recall from the previous subsection, we have presented some variations within the textual requirements of the timing model case study. Therefore, after determining the performance of the NL2DSL tool for the actual diabetic manager case study, we define some variations within the textual requirements of the diabetic manager case study to prove that the NL2DSL has a flexible methodology in order to generate the Xtext grammar.

Thereafter, the actual DSL elements conforming to the Xtext grammar are identified with

Table 5.8: List of Xtext DSL Elements for the Diabetic Manager Case Study Identified by NL2DSL Tool.

Req.#	Textual Requirement	Root Element	Relationships	Attributes		
				Multiple	Single	Optional
1	The scenario of system can be a diabetic manager which is presented with a category attribute of string type.	system:'diabetic manager'				
2	Particularly, it is required to define each health concept by the scenario of this system.				health=null	
3	There, it should contain primary concepts of diabetes, such as the symptom concept with multiple instances.				diabetes=null	
4	Then, the symptom concept must have a name and severity of string type.			symptom:'symptom' name=STRING severity=STRING		
5	Particularly, the common symptom of diabetes are fatigue or hyperglycemia, which are highly linked to developing diabetes.			diabetes:'diabetes' fatigue=null hyperglycemia=null		
6	Furthermore, the system consists of multiple patient concepts where each patient concept is used to define a glucose attribute of string type.				patient:'patient' glucose=STRING	
7	Furthermore, one optional attribute of the patient is name of string type.					(name=STRING)?
8	Furthermore, it should have an attribute like AnyDisease of a boolean datatype.				(AnyDisease?= 'AnyDisease')?	
9	However, it is linked with multiple symptom concepts.		(symptom+= [symptom])*			
10	Additionally, the system contains multiple doctor concepts where the doctor concept must have a specialization and category of string type.		(doctor+= doctor)*	doctor:'doctor' specialization=STRING category=STRING		
11	Moreover, it refers to the multiple instances of patient concepts.					
12	Furthermore, it should contain multiple instances of the medicine concept where the medicine concept contains multiple attribute elements like name and dose of string type.		medicine:'medicine' (attribute+= attribute)*		medicine=null	

Table 5.9: Calculation of NL2DSL Effectiveness for Diabetic Manager Case Study.

	Root Element	Relationships	Attributes			Total	Precision	Recall	Over Specifications
			Multiple	Single	Optional				
Correct	1	2	2	2	1	8			
Incorrect	0	1	0	2	0	3	72.72%	72.72%	18.18%
Missing	0	2	0	1	0	3			
Extra	0	0	1	1	0	2			

the human experts' analysis process from the varied diabetic manager case study, presented in Table 5.10. Then, the NL2DSL tool identifies the DSL elements of the Xtext from the varied diabetic manager case study, presented in Table 5.11.

Table 5.10: List of Actual Xtext DSL Elements for Variated Diabetic Manager Case Study.

Req.#	Textual Requirement	Root Element	Relationships	Attributes		
				Multiple	Single	Optional
1	The system presents a scenario of diabetic manager with a category attribute of string type.	system:'diabetic manager'			category=STRING	
2	There, one primary aspect of diabetes involves a scenario for controlling its conditions through symptom monitoring.					
3	There, the system is composed of multiple symptom concepts where the symptom can be fatigue and hyperglycemia linked to developing diabetes.		(symptom+= symptom)*			
4	Furthermore, the symptom concept must have a name and severity of string type.			symptom:'symptom' name=STRING severity=STRING		
5	Furthermore, the system contains multiple instances of patient concepts where each patient concept is declared to define a glucose attribute of string type.		(patient+= patient)*		patient:'patient' glucose=STRING	
6	Furthermore, one optional attribute of the patient is name of string type.					(name=STRING)?
7	Furthermore, it is represented to define an AnyDisease attribute with a boolean datatype.				(AnyDisease?= 'AnyDisease')?	
8	However, it is linked with multiple symptom concepts.		(symptom+= [symptom])*			
9	Additionally, the system contains multiple doctor concepts where the doctor concept must have a specialization and category of string type.		(doctor+= doctor)*	doctor:'doctor' specialization=STRING category=STRING		
10	Moreover, it refers to the multiple instances of patient concepts.		(patient+= [patient])*			
11	Furthermore, it should contain multiple instances of the medicine concept where the medicine concept contains multiple attribute elements like name and dose of string type.		(medicine+= medicine)*	medicine:'medicine' name=STRING dose=STRING		

Afterward, a performance evaluation of the NL2DSL tool is conducted by comparing the actual results of the human experts' analysis with the generated results of the NL2DSL tool for the varied requirements of the diabetic manager case study. The performance evaluation results of the NL2DSL tool for the varied diabetic manager case study are presented in Table 5.12.

Now, the accumulative type of performance evaluation result of the tool for the diabetic manager case study is calculated, i.e., actual and varied. From the Table 5.8 and Table 5.11, it has shown total Xtext DSL elements generated by the NL2DSL tool are $16+16 = 32$

Correct Xtext DSL elements generated by the NL2DSL tool are $= 8+9 = 17$

Incorrect Xtext DSL elements generated by the NL2DSL tool are $= 3+2 = 5$

Table 5.11: List of Xtext DSL Elements for Variated Diabetic Manager Case Study Identified by the NL2DSL tool.

Req.#	Textual Requirement	Root Element	Relationships	Attributes		
				Multiple	Single	Optional
1	The system presents a scenario of diabetic manager with a category attribute of string type.	system:'diabetic manager'				
2	There, one primary aspect of diabetes involves a scenario for controlling its conditions through symptom monitoring.				diabetes:'diabetes' scenario=null	
3	There, the system is composed of multiple symptom concepts where the symptom can be fatigue and hyperglycemia linked to developing diabetes.		(symptom+=symptom)*	symptom:'symptom' fatigue=null hyperglycemia=null		
4	Furthermore, the symptom concept must have a name and severity of string type.			name=STRING category=STRING		
5	Furthermore, the system contains multiple instances of patient concepts where each patient concept is declared to define a glucose attribute of string type.				patient:'patient' glucose=STRING	
6	Furthermore, one optional attribute of the patient is name of string type.					(name=STRING)?
7	Furthermore, it is represented to define an AnyDisease attribute with a boolean datatype.					(AnyDisease?='AnyDisease')?
8	However, it is linked with multiple symptom concepts.		(symptom+=symptom)*			
9	Additionally, the system contains multiple doctor concepts where the doctor concept must have a specialization and category of string type.		(doctor+=doctor)*	doctor:'doctor' specialization=STRING category=STRING		
10	Moreover, it refers to the multiple instances of patient concepts.					
11	Furthermore, it should contain multiple instances of the medicine concept where the medicine concept contains multiple attribute elements like name and dose of string type.		medicine:'medicine' (attribute+=attribute)*		medicine=null	

Table 5.12: Calculation of NL2DSL Effectiveness for variated Diabetic Manager Case Study.

	Root Element	Relationships	Attributes			Total	Precision	Recall	Over Specifications
			Multiple	Single	Optional				
Correct	1	3	2	2	1	9	81.8%	75%	16.67%
Incorrect	0	1	0	1	0	2			
Missing	0	2	0	1	0	3			
Extra	0	0	1	1	0	2			

Missing Xtext DSL elements generated by the NL2DSL tool are = 3+3 =6

And, Extra Xtext DSL elements generated by the NL2DSL tool are = 2+2 = 4

$$\text{Precision} = \frac{\text{Correct}}{\text{Correct} + \text{Incorrect}}$$

$$\text{So, Precision} = \frac{17}{17+5} = 77.27\%$$

$$\text{Recall} = \frac{\text{Correct}}{\text{Correct} + \text{Missing}}$$

So, **Recall** = $\frac{17}{17+6} = 73.9\%$

Over Specification = $\frac{\text{Extra}}{\text{Correct} + \text{Missing}}$

So, **Over Specification** = $\frac{4}{17+6} = 17.39\%$

Chapter 6

Discussion and Limitations

This chapter presents the details of two sections 6.1 and 6.2. **Section 6.1** covers a detailed discussion of the proposed framework. **Section 6.2** presents the limitations of our proposed framework.

Discussions

The proposed framework is introduced with an automatic generation of the Xtext grammar from the natural-language requirements. The feasibility of the proposed framework is proved by providing the textual requirements of systems as case studies, including the timing model and the diabetic manager. Throughout this thesis, we specified the inclusion of the functional requirements of the system as the textual requirement. The complete generated results of the Xtext grammar are presented in Figure 5.3-5.4, proving that our approach is capable of generating the Xtext grammar. Furthermore, the feasibility of the tool is also performed by inducing the variations within the textual requirements. The evaluation results of the NL2DSL tool, as presented in Table 5.3, Table 5.6 and Table 5.9, Table 5.12, prove that the proposed framework is capable of generating the Xtext grammar with a satisfactory level of accuracy. The proposed framework is developed with an open-source NL2DSL tool, available at GitHub [55]. Initially, the textual requirements are

presented with unnecessary information that can degrade the accuracy of results. Therefore, a few preprocessing techniques (presented in Chapter 3) are applied to the textual requirements. Then, the preprocessed results are passed to the transformation engine to match the statements and generate an output of the Xtext grammar.

The main objective of our proposed framework is to provide a truly collaborative environment among multiple stakeholders. Consequently, the proposed framework utilized the NLP techniques to introduce the automatic generation of the Xtext grammar from the natural-language requirements. The textual requirements have no reliance on restricted natural-language templates. However, the rules defined in Chapter 3, assist in writing the requirements within the English language. Therefore, the non-technical stakeholders easily comprehend the generated Xtext grammar with written textual requirements. Further, the natural-language requirements have some reserved words such as string type and optional that directly support the technical experts. It benefits the technical experts by eliciting accurate requirements within less time, as natural-language requirements are utilized to build a mutual consensus among the technical and non-technical stakeholders. Moreover, several organizations can easily incorporate it into their workflows to perform seamless integration with other frameworks of DSLs.

A comparative analysis is conducted with baseline research paper [14], showing that the overall development of the IFML model exhibits less complexity. In this study, most of the proposed NLP rules have targeted the sub-clauses of textual requirements for generation purposes. On the contrary, the implementation of Xtext grammar includes various syntax rules, making its development complex. Moreover, our proposed approach has implemented different NLP rules by targeting the whole sentences to generate the Xtext DSL elements. We have also investigated the ChatGPT natural-language model which provides the Xtext grammar similar to the generated Xtext grammar by the NL2DSL tool to a certain extent. For example, it correctly generates a few DSL elements of the Xtext, i.e., the root element, relationship, and attributes but also introduces additional elements

incorrectly. Further, it has limited proficiency and it is unable to generate the attributes of the boolean datatype. Rather than ChatGPT, our NL2DSL is proficient in generating the Xtext grammar from the textual requirements with better accuracy results. The ChatGPT's results of Xtext grammar are publicly accessible from the GitHub site [54].

The proposed framework provides flexibility to incorporate enhancements. Currently, this article focuses on the automated generation of the Xtext grammar, where the DSL elements are the root element, relationships, and attributes. The same approach can capture the enumerations, qualified names, and inheritance, but additional NLP rules might be proposed. Thus, we can say that such type of enhancements can be implemented in future work.

Limitations

The proposed framework is highly supportive of enhancements to define other DSL elements of the Xtext. Although the proposed approach generates the Xtext grammar according to the case studies, it still has some limitations. Currently, it supports the generation of primary DSL elements of the Xtext. However, it lacks the generation of the other DSL elements such as inheritance, enumerations, and qualified names. We believe that the implementation and methodology of our proposed framework can easily incorporate such identified limitations.

Chapter 7

Conclusion and Future Directions

Conclusion

This thesis presented a comprehensive framework to support a collaborative environment among the non-technical and technical stakeholders. Therefore, this thesis presented a combined utilization of both natural-language processing (NLP) techniques and model-driven engineering (MDE). The proposed framework provides an automated generation of the Xtext grammar from the natural-language requirements. Thus, the technical concepts of the Xtext grammar are available in natural language resembling the language of non-technical stakeholders. Similarly, the natural-language requirements support the technical stakeholders by including reserved words such as string type and optional.

Within this thesis, the proposed framework applied preprocessing techniques to the textual requirements. Several NLP rules (14) are specified to extract the primary DSL elements of the Xtext, such as attributes, etc. We defined the textual requirements in the context of functional concepts of the systems. Therefore, we specified two case studies, i.e., the timing model and the diabetic manager, to ensure the applicability of our proposed framework. Thus, we proposed a complete algorithm that enables the execution of this proposed framework. Within this proposed approach, we have created a UI tool named

Natural-Language To Domain-Specific Language (NL2DSL), which allows users to input a PDF file consisting of textual requirements. It automatically generates an Xtext grammar that is saved in a DSL file of the .xtext extension.

Currently, it has a new methodology where a framework is developed to provide an automated Xtext grammar using the NLP techniques. The evaluation results prove that it is capable of generating an automated Xtext grammar from the textual requirements with a satisfactory degree of accuracy. The proposed framework lies in providing the natural-language requirements without the need for any restricted natural-language template. Thus, textual requirements are specified in any style of the English language with the set of rules that are elicited from the earlier research paper [14]. Generally, the proposed framework provides benefits to streamline the requirement elicitation phase. It offers benefits to industrial organizations to integrate with other frameworks of DSLs.

The proposed framework is highly supportive of further enhancements. However, a rule-based approach comprising the regular expressions is applied to extract the primary DSL elements of the Xtext. There is still some extendable way where some aspects of the Xtext grammar are not supported, including inheritance, enumerations, and qualified names. Our proposed framework supports upgrading in the aforementioned extendable ways. The NL2DSL tool can be upgraded by employing the model-transformation approaches to expand its utility in terms of verification aspects.

References

- [1] M. Eysholdt and H. Behrens, “Xtext: implement your language faster than the quick and dirty way,” in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pp. 307–309, 2010.
- [2] C. J. Davis, R. M. Fuller, M. C. Tremblay, and D. J. Berndt, “Communication challenges in requirements elicitation and the use of the repertory grid technique,” *Journal of Computer Information Systems*, vol. 46, no. 5, pp. 78–86, 2006.
- [3] V. Laporti, M. R. Borges, and V. Braganholo, “Athena: A collaborative approach to requirements elicitation,” *Computers in Industry*, vol. 60, no. 6, pp. 367–380, 2009.
- [4] V. Goodrich and L. Olfman, “An experimental evaluation of task and methodology variables for requirements definition phase success,” in *Twenty-Third Annual Hawaii International Conference on System Sciences*, vol. 4, pp. 201–209, IEEE Computer Society, 1990.
- [5] U. Erra and G. Scanniello, “Assessing communication media richness in requirements negotiation,” *IET software*, vol. 4, no. 2, pp. 134–148, 2010.
- [6] E. D. Liddy, “Natural language processing,” 2001.
- [7] R. Sonbol, G. Rebdawi, and N. Ghneim, “A machine translation like approach to generate business process model from textual description,” *SN Computer Science*, vol. 4, no. 3, p. 291, 2023.

- [8] S. Sholiq, R. Sarno, and E. S. Astuti, “Generating bpmn diagram from textual requirements,” *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 10, pp. 10079–10093, 2022.
- [9] I. Tangkawarow, R. Sarno, and D. Siahaan, “Id2sbvr: A method for extracting business vocabulary and rules from an informal document,” *Big Data and Cognitive Computing*, vol. 6, no. 4, p. 119, 2022.
- [10] S. C. Allala, J. P. Sotomayor, D. Santiago, T. M. King, and P. J. Clarke, “Towards transforming user requirements to test cases using mde and nlp,” in *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, pp. 350–355, IEEE, 2019.
- [11] R. Gröpler, V. Sudhi, E. J. C. García, and A. Bergmann, “Nlp-based requirements formalization for automatic test case generation.,” in *CS&P*, vol. 21, pp. 18–30, 2021.
- [12] C. Wang, F. Pastore, A. Goknil, and L. C. Briand, “Automatic generation of acceptance test cases from use case specifications: an nlp-based approach,” *IEEE Transactions on Software Engineering*, vol. 48, no. 2, pp. 585–616, 2020.
- [13] M. W. Anwar, I. Ahsan, F. Azam, W. H. Butt, and M. Rashid, “A natural language processing (nlp) framework for embedded systems to automatically extract verification aspects from textual design requirements,” in *Proceedings of the 2020 12th International Conference on Computer and Automation Engineering*, pp. 7–12, 2020.
- [14] M. Hamdani, W. H. Butt, M. W. Anwar, I. Ahsan, F. Azam, and M. A. Ahmed, “A novel framework to automatically generate ifml models from plain text requirements,” *IEEE Access*, vol. 7, pp. 183489–183513, 2019.
- [15] A. Zaki-Ismail, M. Osama, M. Abdelrazek, J. Grundy, and A. Ibrahim, “Rcm-extractor: an automated nlp-based approach for extracting a semi formal representa-

- tion model from natural language requirements,” *Automated Software Engineering*, vol. 29, no. 1, p. 10, 2022.
- [16] R. P. Dias, C. Vidanapathirana, R. Weerasinghe, A. Manupiya, R. Bandara, and Y. Ranasinghe, “Automated use case diagram generator using nlp and ml,” *arXiv preprint arXiv:2306.06962*, 2023.
- [17] M. Imtiaz Malik, M. Azam Sindhu, and R. Ayaz Abbasi, “Extraction of use case diagram elements using natural language processing and network science,” *Plos one*, vol. 18, no. 6, p. e0287502, 2023.
- [18] A. M. Alashqar, “Automatic generation of uml diagrams from scenario-based user requirements,” *Jordanian Journal of Computers and Information Technology*, vol. 7, no. 2, 2021.
- [19] Z. A. Hamza and M. Hammad, “Generating uml use case models from software requirements using natural language processing,” in *2019 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO)*, pp. 1–6, IEEE, 2019.
- [20] F. Alharbia, S. R. Masadeh, and F. Alshrouf, “A framework for the generation of class diagram from text requirements using natural language processing,” *International Journal*, vol. 10, no. 1, 2021.
- [21] S. Yang and H. Sahraoui, “Towards automatically extracting uml class diagrams from natural language specifications,” in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pp. 396–403, 2022.
- [22] E. A. Abdelnabi, A. M. Maatuk, T. M. Abdelaziz, and S. M. Elakeili, “Generating uml class diagram using nlp techniques and heuristic rules,” in *2020 20th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, pp. 277–282, IEEE, 2020.

- [23] S. Nasiri, Y. Rhazali, M. Lahmer, and A. Adadi, "From user stories to uml diagrams driven by ontological and production model," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 6, 2021.
- [24] T. Kochbati, S. Li, S. Gérard, and C. Mraidha, "From user stories to models: A machine learning empowered automation.," *MODELSWARD*, vol. 10, p. 0010197800280040, 2021.
- [25] M. Elallaoui, K. Nafil, and R. Touahni, "Automatic transformation of user stories into uml use case diagrams using nlp techniques," *Procedia computer science*, vol. 130, pp. 42–49, 2018.
- [26] O. S. Dawood *et al.*, "Toward requirements and design traceability using natural language processing," *European Journal of Engineering and Technology Research*, vol. 3, no. 7, pp. 42–49, 2018.
- [27] Shweta, R. Sanyal, and B. Ghoshal, "Automated class diagram elicitation using intermediate use case template," *IET Software*, vol. 15, no. 1, pp. 25–42, 2021.
- [28] S. Zhong, A. Scarinci, and A. Cicirello, "Natural language processing for systems engineering: Automatic generation of systems modelling language diagrams," *Knowledge-Based Systems*, vol. 259, p. 110071, 2023.
- [29] J. Chen, B. Hu, W. Diao, and Y. Huang, "Automatic generation of sysml requirement models based on chinese natural language requirements," in *Proceedings of the 2022 6th International Conference on Electronic Information Technology and Computer Engineering*, pp. 242–248, 2022.
- [30] G. J. Hwang, T. Mazzuchi, and S. Sarkani, "Generation of mbse models from system requirements," in *Open Architecture/Open Business Model Net-Centric Systems and Defense Transformation 2022*, vol. 12119, pp. 35–49, SPIE, 2022.

- [31] M. Chami, C. Zoghbi, and J.-M. Bruel, “A first step towards ai for mbse: Generating a part of sysml models from text using ai,” *A First Step towards AI*, 2019.
- [32] Y. Qie, H. Shen, and A. Liu, “A deep learning and ontology based framework for textual requirements analysis and conceptual model generation,” in *Complex Systems Design & Management: Proceedings of the 4th International Conference on Complex Systems Design & Management Asia and of the 12th Conference on Complex Systems Design & Management CSD&M 2021*, pp. 3–14, Springer, 2021.
- [33] A. Haj, A. Jarrar, Y. Balouki, and T. Gadir, “The semantic of business vocabulary and business rules: an automatic generation from textual statements,” *IEEE Access*, vol. 9, pp. 56506–56522, 2021.
- [34] M. M. Awan, W. H. Butt, M. W. Anwar, and F. Azam, “Seamless runtime transformations from natural language to formal methods—a usecase of z-notation,” in *2022 17th Annual System of Systems Engineering Conference (SOSE)*, pp. 375–380, IEEE, 2022.
- [35] M. W. Anwar and F. Ciccozzi, “Blended metamodeling for seamless development of domain-specific modeling languages across multiple workbenches,” in *2022 IEEE International Systems Conference (SysCon)*, pp. 1–7, IEEE, 2022.
- [36] A. Veizaga, M. Alferez, D. Torre, M. Sabetzadeh, and L. Briand, “On systematically building a controlled natural language for functional requirements,” *Empirical Software Engineering*, vol. 26, no. 4, p. 79, 2021.
- [37] G. Hains and O. Fenek, “Machine learning pseudo-natural language for temporal logic requirements of embedded systems,” in *2023 15th International Conference on Knowledge and Systems Engineering (KSE)*, pp. 1–4, IEEE, 2023.
- [38] I. Predoiaia, D. Kolovos, M. Lenk, and A. Garc’ia-Dom’inguez, “Streamlining the development of hybrid graphical-textual model editors for domain-specific languages,” *Journal of Object Technology*, vol. 22, no. 2, 2023.

- [39] M. Latifaj, F. Ciccozzi, and M. Mohlin, “Higher-order transformations for the generation of synchronization infrastructures in blended modeling,” *Frontiers in Computer Science*, vol. 4, p. 1008062, 2023.
- [40] M. W. Anwar, M. Latifaj, and F. Ciccozzi, “Blended modeling applied to the portable test and stimulus standard,” in *ITNG 2022 19th International Conference on Information Technology-New Generations*, pp. 39–46, Springer, 2022.
- [41] K. Aslam, Y. Chen, M. Butt, and I. Malavolta, “Cross-platform real-time collaborative modeling: an architecture and a prototype implementation via emf. cloud,” *IEEE Access*, 2023.
- [42] Y. Liu and J.-M. Bruel, “Modelling and verification of natural language requirements based on states and modes,” in *30th International Requirements Engineering Conference Workshops (REW 2022)*, IEEE, 2022.
- [43] M. El Hamlaoui, Y. Laghouaouta, Y. Qamsane, and A. Mishra, “A model based approach for generating modular manufacturing control systems software,”
- [44] F. FOURATI, M. T. BHIRI, and R. ROBBANA, “Validating event-b models using pddl,” *Procedia Computer Science*, vol. 207, pp. 2638–2647, 2022.
- [45] H. Brabra, A. Mtibaa, W. Gaaloul, and B. Benatallah, “Toward higher-level abstractions based on state machine for cloud resources elasticity,” *Information Systems*, vol. 90, p. 101450, 2020.
- [46] R. Mzid, A. Charfi, and N. Etteyeb, “Use of compiler intermediate representation for reverse engineering: A case study for gcc compiler and uml activity diagram,” in *MODELSWARD*, pp. 211–218, 2022.
- [47] S. H. Hiba and M. Belguidoum, “Autocadep: An approach for automatic cloud application deployment,” in *Service-Oriented Computing–ICSOC 2019 Workshops: WE-*

SOACS, ASOCA, ISYCC, TBCE, and STRAPS, Toulouse, France, October 28–31, 2019, Revised Selected Papers 17, pp. 82–94, Springer, 2020.

- [48] A. Moin, M. Challenger, A. Badii, and S. Günemann, “A model-driven approach to machine learning and software modeling for the iot: Generating full source code for smart internet of things (iot) services and cyber-physical systems (cps),” *Software and Systems Modeling*, vol. 21, no. 3, pp. 987–1014, 2022.
- [49] G. Daniel, J. Cabot, L. Deruelle, and M. Derras, “Xatkit: a multimodal low-code chatbot development framework,” *IEEE Access*, vol. 8, pp. 15332–15346, 2020.
- [50] B. Jahić, N. Guelfi, and B. Ries, “Semkis-dsl: A domain-specific language to support requirements engineering of datasets and neural network recognition,” *Information*, vol. 14, no. 4, p. 213, 2023.
- [51] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky, “The stanford corenlp natural language processing toolkit,” in *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pp. 55–60, 2014.
- [52] Z. Zhong, J. Guo, W. Yang, T. Xie, J.-G. Lou, T. Liu, and D. Zhang, “Generating regular expressions from natural language specifications: Are we there yet?,” in *AAAI Workshops*, pp. 791–794, 2018.
- [53] M. W. Anwar, F. Ciccozzi, and A. Bucaioni, “Enabling blended modelling of timing and variability in east-adl,” in *Proceedings of the 16th ACM SIGPLAN International Conference on Software Language Engineering*, pp. 169–180, 2023.
- [54] MedlinePlus, “Diabetes.” <https://medlineplus.gov/ency/article/001214.htm>, 2023.
- [55] “NL2DSL.” <https://github.com/AminaZafar-CEME/NL2DSL>. Accessed: 17-12-2023.