Natural Language–Based Conceptual Modelling Frameworks: State of the Art and Future Opportunities

BAYZID ASHIK HOSSAIN, School of Computing, Mathematics and Engineering, Charles Sturt University, Australia
MD. SADDAM HOSSAIN MUKTA, Department of Computer Science and Engineering, United International University, Bangladesh
MD ADNANUL ISLAM, Action Lab, Faculty of Information Technology, Monash University, Australia
AKIB ZAMAN, Department of Computer Science and Engineering, United International University, Bangladesh
ROLF SCHWITTER, School of Computing, Macquarie University, Australia

Identifying requirements for an information system is an important task and conceptual modelling is the first step in this process. Conceptual modelling plays a critical role in the information system design process and usually involves domain experts and knowledge engineers who brainstorm together to identify the required knowledge to build an information system. The conceptual modelling process starts with the collection of necessary information from the domain experts by the knowledge engineers. Afterwards, the knowledge engineers use traditional model driven engineering techniques to design the system based on the collected information. Natural language–based conceptual modelling frameworks or systems are used to help domain experts and knowledge engineers in eliciting requirements and building conceptual models from a natural language text. In this article, we discuss the state of the art of some recent conceptual modelling frameworks that are based on natural language. We take a closer look at how these frameworks are built, in particular at the underlying motivation, architecture, types of natural language used (e.g., restricted vs. unrestricted), types of the conceptual model generated, verification support of the requirements specifications as well as the conceptual models, and underlying knowledge representation formalism. We also discuss some future research opportunities that these frameworks offer.

CCS Concepts: • **Software and its engineering** → **Requirements analysis**; • **General and reference** → **Surveys and overviews**; • **Information systems** → **Relational database model**; • **Theory of computation** → **Logic;**

Additional Key Words and Phrases: Natural language processing, information extraction, conceptual modelling, knowledge representation, semantic round-tripping

Authors' addresses: B. A. Hossain, School of Computing, Mathematics and Engineering, Charles Sturt University, Panorama Ave, Bathurst 2795, Australia; email: bhossain@csu.edu.au; Md. S. H. Mukta and A. Zaman, Department of Computer Science and Engineering, United International University, Madani Avenue, Dhaka 1212, Bangladesh; Md. A. Islam, Action Lab, Faculty of Information Technology, Monash University, Clayton 3800, Victoria, Australia; R. Schwitter, School of Computing, Macquarie University, Balaclava Rd., Macquarie Park, Sydney 2109, Australia.

0360-0300/2023/08-ART12 $15.00
https://doi.org/10.1145/3596597

ACM Reference format:
Bayzid Ashik Hossain, Md. Saddam Hossain Mukta, Md Adnanul Islam, Akib Zaman, and Rolf Schwitter. 2023. Natural Language–Based Conceptual Modelling Frameworks: State of the Art and Future Opportunities. *ACM Comput. Surv.* 56, 1, Article 12 (August 2023), 26 pages.
https://doi.org/10.1145/3596597

1 INTRODUCTION

Conceptual modelling is the activity that elicits and describes the relevant knowledge to build an information system. Conceptual modelling involves different stakeholders (e.g., domain experts and knowledge engineers) who brainstorm together to identify the necessary information for building the information system [26]. Identifying necessary information is the most important part for the design of an information system and is known as requirements elicitation [43]. After identifying the required information, knowledge engineers build a conceptual model of the information system by using model driven engineering techniques such as **entity relationship modelling** (**ERM**) [3, 15], object-oriented modelling (i.e., **Unified Modelling Language (UML)**) [47], or **object-role modelling** (**ORM**) [19, 20]. Model driven engineering is a software development technique that focuses on creating conceptual models to represent the problem domain. Therefore, model driven engineering aims at developing an abstract representation of knowledge and activities that governs the target application domain. One of the problems with these models is that they are constructed graphically and as a result they are often hard to understand for domain experts [31]. A properly designed graphical model can be understood without any prior knowledge but when graphical models represent complex situations they lose their intuitive understandability [34]. To overcome this problem, some conceptual modelling techniques use verbalisation to describe the graphical representation [20].

Another problem with these conceptual modelling techniques is that they have no formal semantics; therefore, they are not machine comprehensible, do not support automatic verification and validation nor automatic reasoning [7]. To overcome these problems, previous works used logic representation languages in parallel with traditional conceptual modelling techniques [5, 14, 40]. There exist a number of tools [13, 37, 38] that allow knowledge engineers to draw a conceptual model and then translate the model constructs automatically into a logical representation. This logical representation is then used to verify the model and can help to validate the model. However, using logic with traditional conceptual modelling techniques also introduces some problems like the difficulty to generate logical representations. Furthermore, it is not easy to understand these logical representations for domain experts, since most of them are not trained in formal languages and no well-established methodology is available to represent conceptual models in logic representations [7].

Recent research on conceptual modelling showed that using a **controlled natural language** (**CNL**)[1] for specification and verbalisation of conceptual models can overcome the problems introduced by using a logic representation language in the conceptual modelling process [26]. A CNL can be defined as a subset of a natural language that is obtained by constraining the grammar and vocabulary in order to eliminate the ambiguity as well as the complexity of a natural language. A CNL can be designed in such a way that it has well-defined computational properties and therefore can be translated unambiguously into a formal representation [27]. Using a CNL in a conceptual modelling framework helps the domain experts to better understand the conceptual models as the framework can support semantic round-tripping from a CNL specification to a

[1]Also known as **restricted natural language** (**RNL**).

conceptual model and from a conceptual model to a CNL specification. Using a CNL also allows the machine to understand the models and therefore supports automated reasoning (e.g., verification). But the main problem using a CNL is that although it is easy to read, it is difficult to write. Recent research [56] on conceptual models showed that there exist three modes of conceptual modelling: (1) informal modelling for cognition and communication; (2) semi-formal modelling for planning and documentation; and (3) formal modelling for generation and contracts. In the software industry 70–79% of the modelling is done informally [56]. Using a CNL for conceptual modelling can make this modelling process formal in a seemingly informal way; if a suitable authoring tool is provided. An authoring tool helps the user to write a specification in CNL and to translate the CNL specification into a formal representation.

Natural language–based **conceptual modelling** (**CM**) frameworks were introduced during the 1990s [41, 51]. The main motivation behind these frameworks is to support domain experts and knowledge engineers in requirements elicitation. In this article, we will discuss some popular as well as some newer frameworks or systems. The rest of the article is organized as follows: Section 2 discusses the paper selection strategy; Section 3 explains the review criteria; and Sections 4–15 describe the natural language–based conceptual modelling frameworks or systems: In Section 4, a framework for software development from natural language specification by Saeki et al. [51]; in Section 5, Mich and Garigliano's **natural language–object-oriented production system** (**NL–OOPS**) [41, 42]; in Section 6, Harmain and Gaizauskas's framework for building class models (CM-Builder) [22]; in Section 7, Ilieva and Ormandjieva's framework for deriving models by automatically analysing user requirements [30]; in Section 8, Ambriola and Gervasi's framework for requirement analysis with CIRCE [2]; in Section 9, Deeptimahanti and Babar's UML model generator [10]; in Section 10, Ibrahim and Ahmad's Race framework [29]; in Section 11, the framework for transforming user stories into UML use-case diagrams by Elallaoui et al. [11]; in Section 12, the cognitive grammar-based framework by Selway et al. [54, 55]; in Section 13, **automatic builder of class diagrams** (**ABCD**) by Karaa et al. [4]; in Section 14, the bi-directional grammar-based framework by Hossain et al. [24, 26, 27]; and in Section 15, Omar and Baryannis's **semi-automated conceptual model extraction system** (**SACMES**) [44] are discussed. After that, Section 16 provides a brief comparison of these frameworks, Section 17 discusses the future research opportunities, and finally, Section 18 concludes the article.

2 PAPER SELECTION STRATEGY

To select relevant papers for our review, we looked for systems and frameworks that support automation in building conceptual models by leveraging NL processing. For our article, we mainly considered the systems that follow a model driven engineering practice. To find relevant systems or frameworks, we searched Google Scholar with the predefined search string "natural language based conceptual modelling frameworks" and "generating conceptual models from natural language specifications." Our other strategy was to read the retrieved papers and explore their references to find additional papers. During the selection process of the appropriate papers, we applied the following criteria:

— Studies that are peer-reviewed.
— Studies that are written in English.
— Studies that use NL or controlled NL specification as an input to derive conceptual models or programs.
— Studies where NL processing is done automatically or semi-automatically.
— Studies that use knowledge representation techniques (e.g., first-order logic, description logic, and/or modal logic) to store conceptual models and therefore support verification.

In some cases when some of the criteria (i.e., verification support) were not fulfilled, we considered those papers that are highly relevant and address the same problem. For example, most of the discussed systems or frameworks do not support formal verification since they do not use a logic representation language for conceptual modelling.

3 REVIEW CRITERIA

Since the conceptual modelling process usually starts from scratch, it cannot rely on existing data that would make this process immediately suitable for machine learning techniques. In this article, we have considered frameworks or systems that use NL processing techniques for conceptual modelling from scratch. Each framework is described in terms of its motivation, architecture, and characteristics. Furthermore, we based this study on the following key criteria in order to make these frameworks or systems easily comparable.

— ***Diagram Type:*** The type of the diagram (or model) generated by a conceptual modeling framework (i.e., ER, UML Class, ORM, Dataflow, or Use-Case diagram).
— ***Type of NL:*** Whether the conceptual modeling framework uses a restricted or unrestricted NL as a requirements specification language.
— ***Knowledge Representation Formalism:*** The type of knowledge representation formalism used by the frameworks for verification; for example, first-order logic, description logic, or modal logic.
— ***Semantic Round-tripping:*** Whether the CM framework supports semantic round-tripping or not between a NL specification and a conceptual model.
— ***Verification:*** Whether the CM framework supports verification for a requirement specification and a conceptual model in order to identify inconsistencies.

For each of the introduced frameworks or systems, we discuss the motivation behind the authors' work (e.g., the problem that they intended to solve), how the framework has been built (e.g., types of NL processing techniques used), and finally, in the characteristics section, we discuss the properties that make them different from other frameworks or systems.

4 SOFTWARE DEVELOPMENT PROCESS FROM NL SPECIFICATION

4.1 Motivation

The motivation behind the work of Saeki et al. [51] is to find a methodology that constructs a prototype program or a formal specification from an informal NL specification. Describing requirements for software in a complete and consistent way during the early stage of the development phase is not an easy task. Many researchers have proposed methods for prototyping and verification that achieved promising results but these methods are useful only after the prototype or formal specification is constructed. The work of Saeki et al. makes an effort to fill the gap of creating a prototype program or formal specification from a software requirements specification.

4.2 Architecture

The architecture in Figure 1 adopted from [51] shows the process of deriving a formal specification incrementally from an informal specification written in NL. This process consists of two major activities: (1) a design activity and (2) an elaborate activity. The design activity is responsible for extracting an object-oriented model structure of the software modules interactively from an informal English description. Every content word (e.g., nouns and verbs) in the NL description is associated with a particular software concept such as class, attribute, and method. Special attention is given to the patterns of verbs that occur in the sentences to make the process applicable for dynamic systems. The elaborate activity is responsible for refining and rewriting the informal

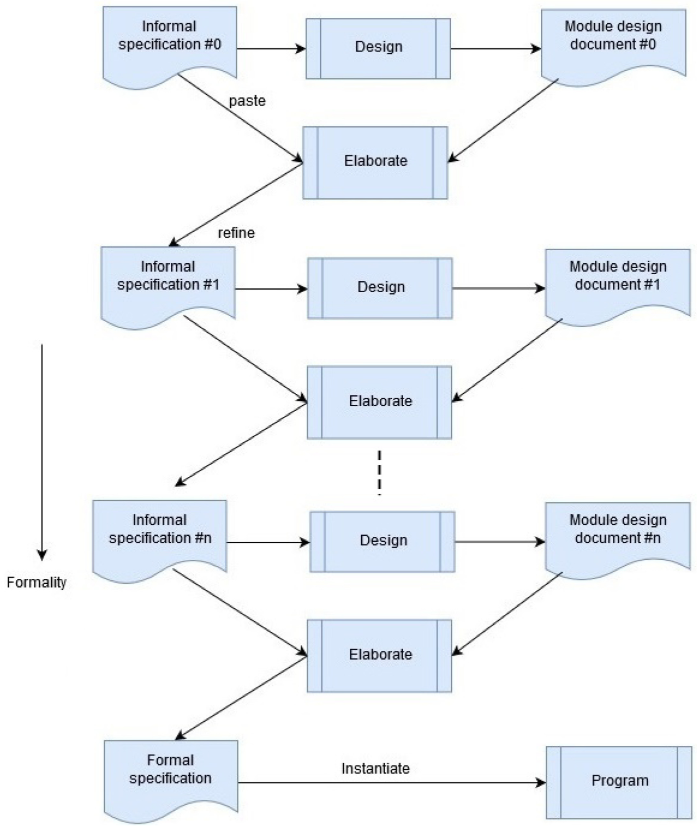
Informal specification #0
Design
Module design document #0
paste
Elaborate
refine
Informal specification #1
Design
Module design document #1
Elaborate
Informal specification #n
Design
Module design document #n
Formality
Elaborate
Formal specification
Instantiate
Program

Fig. 1. Software development process from natural language specification.

specification precisely according to a module design document. The final output of this design and elaborate activity cycle is a formal specification of the system that is used to generate a program.

4.3 Characteristics

Saeki et al. showed a process that builds software modules incrementally from an object-oriented specification. The object-oriented specification is obtained through the design activity from an informal NL specification. The described system can analyse one sentence at a time and can extract required software concepts but is unable to determine the relevant words that are needed for the formal specification. Hence, a human analyst does the elaborate activity and reviews as well as refines the relevant software concepts derived from each sentence.

5 NL–OOPS

5.1 Motivation

The motivation behind Mich and Garigliano's work [41, 42] is to support requirements elicitation and definition during the **software development lifecycle** (**SDLC**) process. In an SDLC, the requirement analysis phase plays a crucial role because errors and omissions in this phase may cause failure to the entire software system. Formalisation can be used to specify requirements but cannot help to extract them. Requirements elicitation highly depends on the analyst's experience and intuition. The goal of the NL–OOPS project was to build a **computer-aided software engineering** (**CASE**) tool that supports requirement analysis by generating object-oriented conceptual models

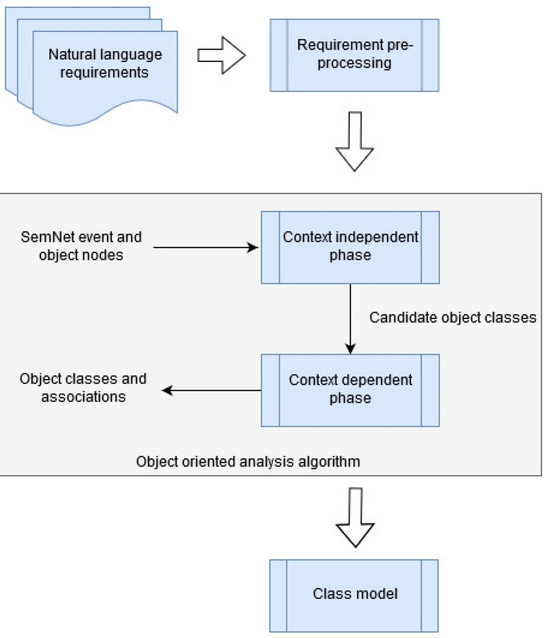
Natural language requirements
Requirement pre-processing
SemNet event and object nodes
Context independent phase
Candidate object classes
Object classes and associations
Context dependent phase
Object oriented analysis algorithm
Class model

Fig. 2. Natural language–object-oriented production system (NL–OOPS).

from requirements documents written in NL. For this purpose, NL–OOPS uses a general-purpose NLP system called **LOLITA** (**Large-scale Object based Linguistic Interactor, Translator and Analyser**) [17] that can process informal NL.

5.2 Architecture

At first the informal NL requirements text is analysed by LOLITA (Figure 2) and then new nodes are added to its information storage that contains around 100,000 interlinked concepts merged with WordNet [17]. Information stored in LOLITA is kept in the form of a knowledge graph, which is also known as a **semantic network** (**SemNet**). This information can be accessed, modified, and also extended by using NL input. SemNet contains two types of nodes: entity nodes and event nodes. Simple relationships between the concepts are represented by arcs whereas complex relationships between concepts are represented using event nodes. Requirements texts are processed morphologically, syntactically, semantically, and pragmatically. Thus, the information stored in SemNet becomes independent from the grammatical form of the NL requirements. The latest version of NL–OOPS supports traceability between the original requirements texts, their representation in SemNet, and the class models generated from the text as the final output. To obtain models at different levels of detail, a user can interactively change some portion of the text or the representation of SemNet at different steps of processing.

NL–OOPS uses an algorithm for the extraction of the objects and their associations during the object-oriented modelling activity. The object-oriented analysis algorithm consists of a context-independent phase and a context-dependent phase. In the context-independent phase, a list of candidate classes is generated. The context-dependent phase performs an analysis and uses LOLITA's event classification facility to extract classes, associations, attributes, values of attributes, and operations with arguments from the list obtained in the first phase of the algorithm. Entity nodes become final classes in the class model. SemNet has four types of event nodes: static, cyclic, dynamic, and instantaneous nodes. Static and cyclic event nodes are related to class associations or attributes whereas dynamic and instantaneous nodes are related to the operation of the classes.

A system analyst can use an interface to SemNet to check both the final output of LOLITA for a given text and to inspect all the nodes LOLITA created for the analysed text. The system analyst can distinguish between old nodes already stored in SemNet and new nodes as NL–OOPS uses different colors to represent them. The traceability function allows the analyst to check which sentence is responsible for a node creation by selecting a node in SemNet as the corresponding sentence is highlighted by the system.

5.3 Characteristics

NL–OOPS is an NLP–based CASE tool that supports NL analysis by extracting the objects and their associations for creating object models. Using LOLITA as a core NLP system for the CASE tool makes it possible to help the analyst both in eliciting and modelling user requirements. The object-oriented analysis algorithm filters and translates the object and event nodes from LOLITA's SemNet.

6 CLASS MODEL BUILDER

6.1 Motivation

Harmain's **class model builder** (CM-Builder) [22] aims to support the requirement analysis phase of an object-oriented software development framework. Graphical CASE tools provide significant help in documenting and analysing the output of the requirement analysis phase, and can also assist in detecting inconsistency and incompleteness in the same stage. However, these tools are unable to help during the initial yet most difficult stage of the analysis phase, that of identifying the object classes, attributes, and relationships that are necessary in modelling the problem domain. To contribute to this part of the analysis phase, a CASE tool needs to deal with human language, which knowledge engineers use as a medium to understand the problems they are solving. CM-Builder analyses software requirements texts written in informal NL by using a robust NL processing system called **LaSIE** (**Large Scale Information Extraction**) [16, 28] and constructs an initial UML class model representing the object classes mentioned in the text and the relationships among them. CM-Builder can generate models in two ways: automatically without an analyst and interactively with an analyst. The initial model can then be used as input to a graphical CASE tool [22] for further refinement.

6.2 Architecture

CM-Builder is a modular CASE tool where the modelling process consists of four stages (Figure 3). These stages can be summarised as follows:

(1) Acquire a set of informal software requirements or problem descriptions in NL.
(2) Use an NLP engine (Figure 3) that syntactically and semantically analyses the informal software requirements and then stores all the intermediate analysis results in the form of declarative knowledge for additional analysis.
(3) Extract object classes, their attributes, and the relationships among them from the results produced by the NLP engine.
(4) Produce an initial static structure model of the system from the extracted objects in a standard format (CDIF[2]) and use a graphical CASE tool to manually refine the initial model.

[2]CASE date interchange format.

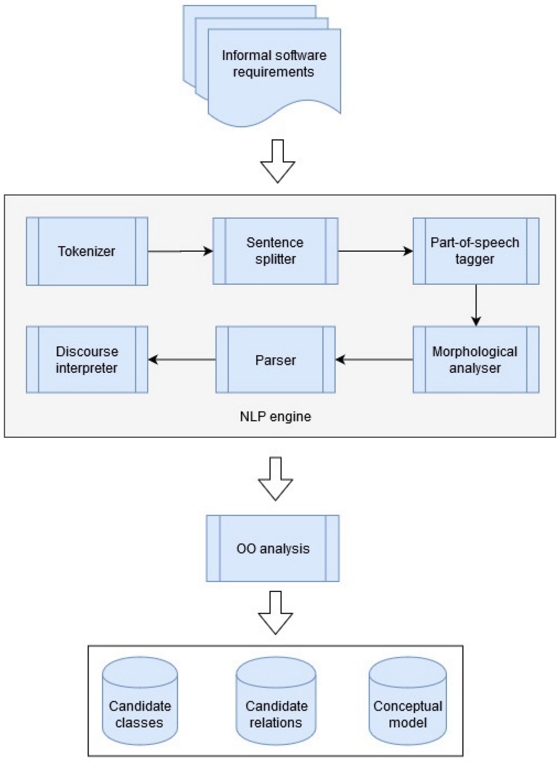

Fig. 3. Class model builder (CM-Builder).

CM-Builder's NLP engine processes the text in three main processing stages: (1) lexical pre-processing, (2) parsing, and (3) discourse interpretation. The lexical pre-processor is composed of four modules: a tokeniser, a sentence splitter, a tagger, and a morphological analyser. The lexical pre-processor takes a plain ASCII file containing a description of the informal requirements as input and produces a set of charts, one per sentence, written as Prolog terms as output. Later these Prolog terms are used directly by a bottom-up chart parser written in Prolog. The chart is an internal structure that is suitable to deal with ambiguous sentences and supports a form of dynamic programming. The parser takes the output of the lexical pre-processor as input to build a syntactic tree and to generate a semantic representation for every sentence in the text in parallel using the grammar rules. The parser is a partial parser because it produces correct but not necessarily complete syntactic structures and semantic representations. After that, the discourse interpretation module reads the semantic representation of every sentence and adds it to a predefined "world" model to construct a discourse model.

6.3 Characteristics

CM-Builder comes in two versions (CM-Builder 1 and CM-Builder 2). In CM-Builder 1, all the associations between attributes and classes, and all the relationships between classes are specified manually by the user. The tool helps to find the most likely classes and the relationships between these classes. In contrast to CM-Builder 1, there is no user interaction with CM-Builder 2 where an initial conceptual model is constructed automatically from an informal requirements text. It is most likely that this model will not be a complete model and may contain errors or omissions.

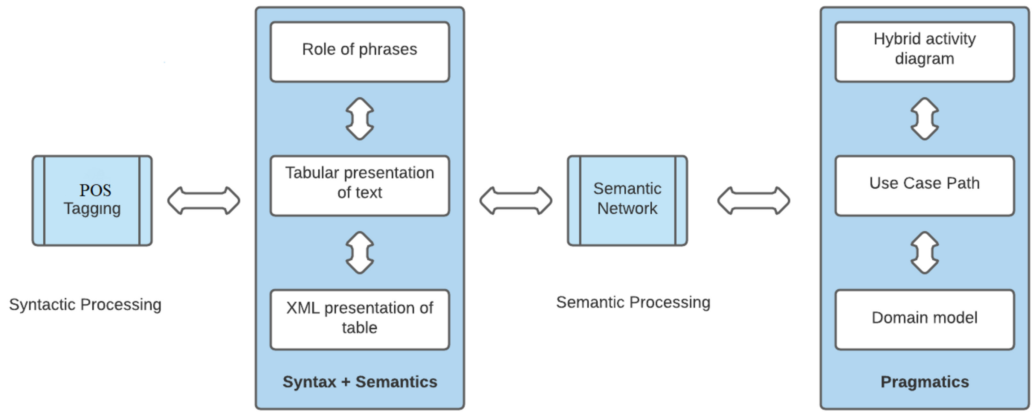

Fig. 4. MDAATUR architecture.

Therefore, the basic idea here is that the output file in the **CASE date interchange format** (**CDIF**) can be used as the input to a CASE tool that will enable the user to further refine and extend the initial model.

Linguistic analysis is still limited in CM-Builder as intransitive verbs are ignored; attachment of postmodifiers such as prepositional phrases and relative clauses are limited. Since the system knowledge bases (lexicons, grammars) are not dynamic and cannot be easily updated by the user, the amount of generic knowledge that is useful for interpreting a range of software requirements texts is limited. Dynamic aspects of the system are not extracted from requirements texts. The tool is not integrated with a graphical CASE tool into a single analysis framework.

7 MODELS DERIVED FROM AUTOMATICALLY ANALYSED TEXTUAL USER REQUIREMENTS

7.1 Motivation

While a domain expert tries to express his/her knowledge in NL, a knowledge engineer tries to express the knowledge in a formal way using a graphical model often without knowing the domain. **Models derived from automatically analysed textual user requirements** (**MDAATUR**'s) [30] objective is to fill in this gap between the different kinds of graphical representation of the same knowledge—those of the user written requirements in NL and those of requirements engineering illustrated using diagrams (i.e., UML). MDAATUR uses unrestricted NL to express requirements text and uses tabular text as well as a semantic network as universal intermediate models. MDAATUR extracts element knowledge using NL processing over the NL description and then matches it with element presentation of a target model.

7.2 Architecture

Textual analysis and model construction in MDAATUR are conducted in four steps (Figure 4). The first step is known as syntax categorisation. In this step, the syntactic category for each word of the text is obtained using a POS tagger. The second step is known as tabular text model building where the information extracted in the first step is used to identify the roles of the words in the sentences and then arranged in a table. Three roles are considered in a sentence: a subject, a predicate, and an object. To label the roles in a sentence, MDAATUR leverages the output from the POS tagger. For identifying roles in a compound sentence, MDAATUR uses grammatical rules and heuristics. The table is composed of four columns, each representing a particular role, which appear in the identical order as the roles appear in a sentence. There is also a column for storing the connection words between simple sentences that make up a compound sentence. This connection is stored to identify the role of the sub-sentence in the main sentence. The third step is known as semantic processing of the text where all the words are presented in a semantic network to discover the interconnection within the whole text. The basic elements of the semantic network are relations (e.g., predicate, prepositional, structural, attributive, possessive, and enumerative). After this step, MDAATUR obtains a complete but compact picture of each element in the network. The fourth and final step is known as interpretation of the text for diagrammatic modelling. The objective of this step is to arrange the information from the previous steps and to model it in the form of a diagram.

7.3 Characteristics

MDAATUR is a framework that automatically analyses the NL requirements and then uses the formal representation of the extracted knowledge to generate graphical models. The MDAATUR system has several advantages over similar systems. For example, MDAATUR analyses unrestricted NL. To bridge the NL model of knowledge and the requirements engineering model of knowledge, MDAATUR uses two internal representations: (1) the tabular representation of text and (2) the semantic network of knowledge. These internal representations can be applied independently in other applications, as they are independent of the style and complexity of the NL. From them, MDAATUR builds three different graphical models (hybrid activity diagrams, use-case path diagrams, and a domain model) of the extracted knowledge.

8 REQUIREMENT ANALYSIS WITH CIRCE

8.1 Motivation

Ambriola's CIRCE system [2] is a framework that aims at supporting the activities of the requirements analysis and modelling phase. Since a carefully conducted requirements analysis is a key factor in the success of a software development project, CIRCE offers a simple but powerful framework for analysing requirements texts expressed in NL. In CIRCE, strong emphasis is given to the comprehensibility of the requirements to facilitate conversation with the domain expert. A modular expert system is responsible for the modelling and analysis activity that ensures the customisability and extendibility of the framework.

8.2 Architecture

CIRCE is an environment that analyses controlled NL requirements unlike the previous NL–based frameworks discussed in this survey. CIRCE applies successive transformations to the requirements text to obtain concrete, rendered views of conceptual models. These include the models of the requirements document itself that is considered as a textual artifact, the models of the system depicted by the requirements text, and the models of the requirements development process by tracing the relevant artifacts [2]. The general architecture of CIRCE is divided into five modules (from NL requirements to conceptual models). Each of these modules can be considered as a function with the input and possible output depicted in Figure 5.

In the first step, the requirements text is parsed and transformed into a forest of parse trees using a domain-based parser named CICO [18]. These parse trees are closely related to the original requirements text, but omitting many surface features to facilitate subsequent analysis. The parse trees acquired from the previous transformation are then encoded as tuples and kept in a common tuple space. These encoded tuples act as a source of the extensional knowledge of the requirements text, and are used as input for the next step. An embedded expert system is then used to refine the extensional knowledge stored in the tuple space. The expert system enriches the

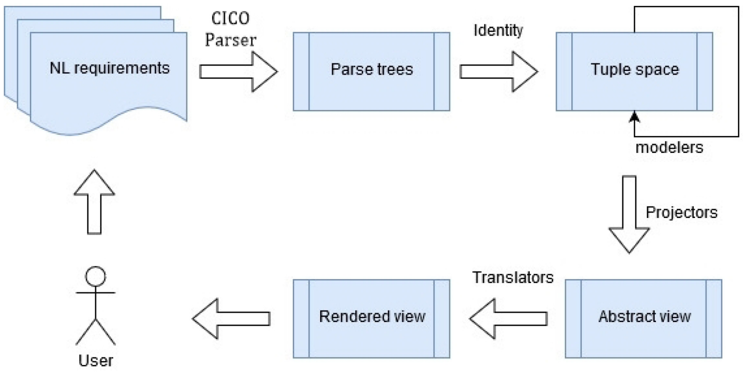

Fig. 5. CIRCE architecture.

tuple space with more tuples. The expert system uses the intensional knowledge about the basic structure and behavior of a software system, and the existing tuples in the tuple space to derive new tuples. The intensional knowledge is provided by a modular component called modelers. The projector is another type of component in CIRCE's architecture that is responsible for providing different views of a requirements text. When a particular view is wanted, projectors extract the necessary information from the tuple space. The extracted information is stored as an internal representation (e.g., using a graph-theoretic notation) in CIRCE and needs to be made tangible for presenting to the user. The translator is the component that is responsible for translating the internal representation into a conceptual model.

CIRCE's architecture has been implemented as a web-based system. The system has a central server that acts as a repository to store requirements documents. A user can use any standard web browser to edit these requirements documents and request specific views. CIRCE performs the necessary transformations to create the requested view and then shows the result as a web page.

8.3 Characteristics

The aim of CIRCE is to provide a basic framework and infrastructure that can accommodate most of the ideas expressed in the area of automated NL requirements analysis. CIRCE also offers a level of enactment, adequate generality, and an initial set of conceptual models and views that make it usable in industry [2]. It has often been observed that a NL is not suitable to represent requirements as NLs are ambiguous, inconsistent, and redundant. CIRCE uses a controlled NL for representing requirements to overcome these problems. Although using NL is the most appropriate way for communication, NL is particularly well suited for stating requirements when it is used in a consistent manner. The effort needed to get familiar with CIRCE was found to be small compared to the effort needed to manually analyse the documents iteratively for each version. The CIRCE framework supports deriving static models such as ER diagrams or UML class diagrams and dynamic models such as use-case diagrams, or event-condition-action rules [2] from the requirements.

9 UML MODEL GENERATOR FROM ANALYSIS OF REQUIREMENTS

9.1 Motivation

Software requirements specifications written in NL are often influenced by geographical, psychological, and sociological factors. They are also incomplete, inconsistent, and ambiguous in nature. **UML Model Generator from Analysis of Requirements** (**UMGAR**) [10] was developed as a domain-independent tool to mitigate these issues of software requirements specifications. Another objective of UMGAR was to overcome the defects caused by overlooking requirements

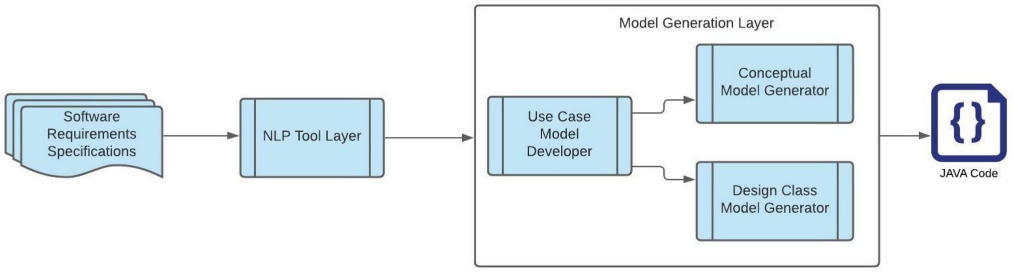
Software Requirements Specifications
NLP Tool Layer
Model Generation Layer
Use Case Model Developer
Conceptual Model Generator
Design Class Model Generator
JAVA Code

Fig. 6. UMGAR architecture.

specification by human reviewers. UMGAR provides complete automated support in developing both static and dynamic models from NL requirements specifications.

9.2 Architecture

UMGAR leverages an object-oriented analysis and design approach to elicit objects from requirements specifications in NL. UMGAR generates analysis and design class models by an approach that combines both the **Rational Unified Process** (**RUP**) [33] and the ICONIX [50] process using NLP tools like the Stanford parser, WordNet, and JavaRAP. UMGAR can process large requirements documents and uses a glossary to avoid communication gaps among team members to create unambiguous requirements. The architecture of UMGAR consists of two layers (Figure 6): (1) NLP tool layer and (2) model generation layer. The NLP tool layer normalises requirements specifications by removing ambiguous requirements and by identifying incomplete requirements. The NLP tool layer performs syntactic reconstruction to extract all possible requirements by splitting complex sentences into simple sentences. UMGAR has eight syntactic rules defined for the syntactic reconstruction purpose. The syntactic reconstruction process is an interactive process because if the NLP tool layer finds any sentence in the requirements specifications that does not satisfy the rules, then UMGAR asks the user to change the sentence structure according to the rules. The model generation layer is responsible for generating different object-oriented models by using normalised requirements from the NLP tool layer. The model generation layer is composed of a use-case model developer, an analysis class model developer, and a design class model developer. To demonstrate traceability between requirements specifications and code, UMGAR generates code models in JAVA by using the Enterprise Architect code generation feature.

9.3 Characteristics

UMGAR is a domain-independent framework for generating UML diagrams that performs syntactic reconstruction of complex sentences by following eight predefined rules. This syntactic reconstruction is an interactive process since it requires users to rewrite complex sentences that are not supported by the predefined syntactic rules. UMGAR can generate use-case models, analysis class models, and design class models. An important feature of UMGAR is the XMI import facility that enables UMGAR to visualise UML diagrams in any UML modelling tool that supports XMI file importation.

10 RACE, A CLASS DIAGRAM EXTRACTION TOOL USING NLP TECHNIQUES

Ibrahim and Ahmad [29] propose a method and develop a tool called RACE to facilitate the process of requirements analysis and class diagram extraction from NL text. They analyse requirements and extract class diagrams from textual content by using **natural language processing** (**NLP**) and domain ontology techniques. Usually, the requirements are taken from interviews, documents,

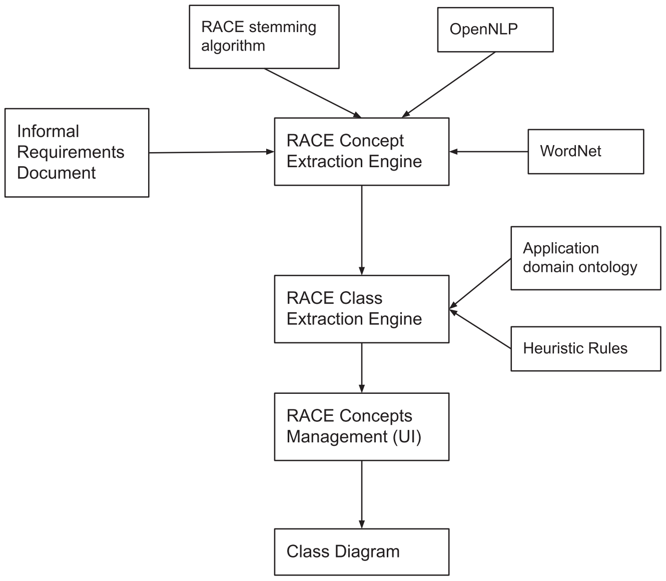
RACE stemming algorithm
OpenNLP
Informal Requirements Document
RACE Concept Extraction Engine
WordNet
Application domain ontology
RACE Class Extraction Engine
Heuristic Rules
RACE Concepts Management (UI)
Class Diagram

Fig. 7. RACE system architecture.

and notes. The RACE tool can generate class diagrams in a fast way and uses **taxonomic class modelling** (**TCM**) for the extraction of class diagrams.

10.1 Architecture

The RACE framework uses NLP and domain ontology techniques to understand the requirements well (Figure 7). First, RACE uses the openNLP [46] parser for POS tagging and chunking (i.e., dividing sentences into phrases). Afterwards, a stemming algorithm is used for the morphological analysis [21] of the requirements. After that, RACE uses WordNet to validate the semantic correctness of the sentences generated during the syntactic analysis. RACE uses the hypernyms from WordNet for selected nouns to verify generalisation relationships where a noun phrase is supposed to be "*a kind of another*" noun phrase. WordNet is also used to find semantically similar terms, and for the acquisition of synonyms. RACE uses the synonyms to extract words that are semantically related to each other. By using the above steps, RACE extracts possible concepts. For identifying the concepts correctly, RACE takes help from a domain ontology. Later, RACE proposes a few rules for class, attribute, and relationship identification. Finally, RACE also provides a system to manually add, create, or delete concepts.

10.2 Characteristics

The RACE tool mainly focuses on producing class diagrams from users' requirements. The authors use NLP techniques for analysing requirements that are written in informal text. The system has both internal (i.e., concept extraction engine and class extraction engine) and external (i.e., concept management) components. They use the concept extraction engine, which generates concepts with the help of a domain ontology. They use WordNet for semantic correctness of the extracted concepts, its attributes, and relationship between the concepts. The goal of the concept extraction engine is to extract concepts from the requirements description. On the other hand, the class extraction engine takes the output from the concept extraction engine to generate class diagrams by using some heuristic rules.

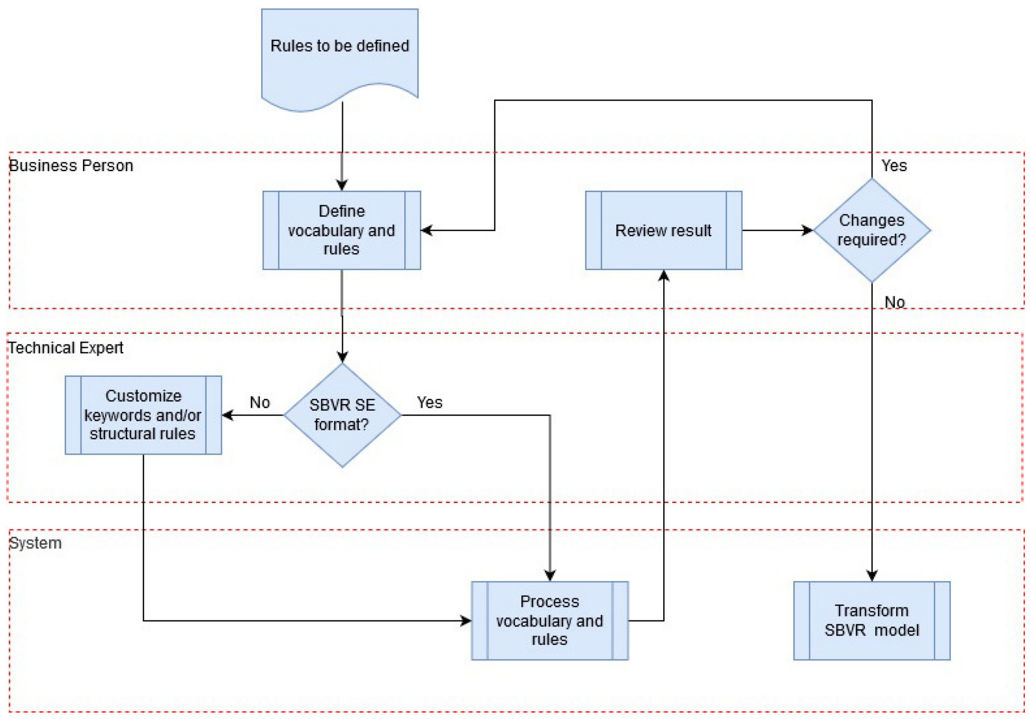

Fig. 8. Cognitive grammar-based approach.

11 COGNITIVE GRAMMAR-BASED FRAMEWORK

11.1 Motivation

The main motivation behind the cognitive grammar-based framework by Selway et al. [54, 55], is to support business users in creating and maintaining their business vocabulary and rules using NL and a flexible document structure. The idea is to reduce the manual effort that is required to formalise the business specifications and transform them into conceptual models suitable for use in and for the development of an information system. Another idea is to allow business users to build well-formed sets of business vocabularies and rules, with consequential benefits in communicating requirements, software quality, launch, and productivity. This approach uses the unique combination of model-driven engineering [53], cognitive linguistics [35], and knowledge-based configuration techniques to implement the framework.

11.2 Architecture

In a cognitive grammar-based approach, the overall analysis process starts whenever an organisation identifies some rules to specify and assigns a business person in the organization to define the business vocabulary and rules following their own style and format (Figure 8). The cognitive grammar-based approach uses SBVR Structured English [39] as the specification language. If the rules defined by the organisation vary too widely with the SBVR Structured English, a technical expert is required to customise the defined rules in SBVR Structured English to identify different elements of the specification. The cognitive grammar-based approach uses a **general architecture for text engineering** (GATE) [9] to process the defined vocabulary and rules. The GATE framework processes a business specification through tokenisation and sentence splitting while the customised rules (keywords and structural rules) from the technical expert identify a document structure. This results in a representation of rules in SBVR Structured English that are displayed to the user for review after processing the specification. The rules may contain errors that are not resolved automatically by the system. The user can then make changes if required and ask the framework to process the document again to update the SBVR model and to get additional feedback. This cycle will continue until the model is complete. Once the business user has resolved all problems with the specification, the technical experts can use the SBVR model to develop an information system.

11.3 Characteristics

The cognitive grammar-based approach uses a theory from the field of cognitive linguistics known as cognitive grammar for NL processing [54]. The cognitive grammar combines syntax, semantics, and pragmatics into a comprehensive view of language and provides consistent treatment for all the linguistic elements that are traditionally treated individually [54]. This research uses a unique combination of techniques from cognitive linguistics, knowledge configuration, and model-driven engineering that semi-automatically transform a business specification into a formal SBVR model. Even though this approach enables the generation of suitable formal models from business specifications, it does not offer any formal verification of the models.

12 BI-DIRECTIONAL GRAMMAR-BASED FRAMEWORK

12.1 Motivation

The main motivation behind Hossain's bi-directional grammar-based framework [24, 26, 27] is to support semantic round-tripping in conceptual modelling and provide verbalisation support for conceptual models. Another motivation behind the bi-directional grammar-based framework is to make the conceptual modelling process formal in a seemingly informal way by using a CNL for writing a specification. The bi-directional grammar-based approach uses description logic as a knowledge representation language. Previous NL–based approaches [2, 22, 29, 41, 51] for conceptual modelling did not constrain the NL enough and did not use description logic to formally represent the conceptual models. Moreover, the idea of semantic round-tripping was not present. The bi-directional grammar-based approach also uses CNL for the conceptual modelling process to overcome the problems that occur using traditional conceptual modelling techniques and tries to make the conceptual modelling process transparent through verbalisation. It offers verbalisation for popular conceptual modelling techniques (e.g., ER and UML class diagrams). These modelling techniques are frequently used in the industry and have no verbalisation support. Existing tools that support creating these models do not provide the facility of writing specifications for conceptual models in a CNL and therefore semantic round-tripping is not possible. Hossain's bi-directional grammar-based conceptual modelling framework offers the following benefits:

— Writing textual specifications in CNL that correspond to conceptual models.
— Description logic-based common formal representation (DL *ALCQI*) for different conceptual models.
— Semantic round-tripping between a textual specification and a graphical representation of conceptual models.
— Verification of the written specifications and the conceptual models.

12.2 Architecture

In a bi-directional grammar-based approach, a specification of the conceptual model is written in CNL first and then the specification is translated automatically with the help of a bi-directional

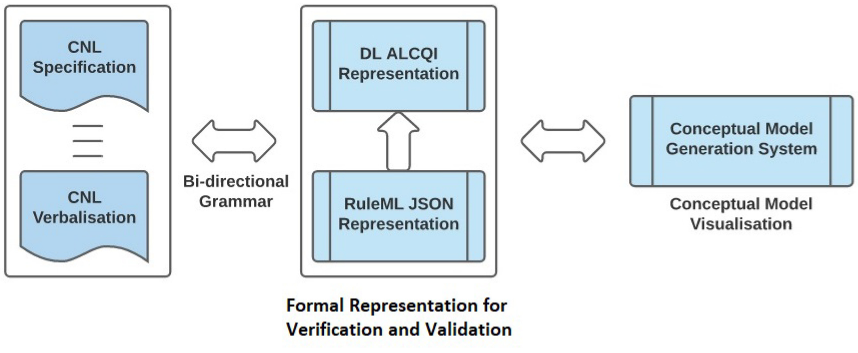
CNL Specification
≡
CNL Verbalisation
Bi-directional Grammar
DL ALCQI Representation
RuleML JSON Representation
Formal Representation for Verification and Validation
Conceptual Model Generation System
Conceptual Model Visualisation

Fig. 9. Bi-directional grammar-based approach.

grammar into a description logic representation. An existing description logic reasoner[3] is then used to check the consistency of the formal representation and after that the corresponding conceptual model is generated from this representation. In this approach, the conceptual model is derived from the specification whereas in conventional logic-based approaches knowledge engineers first draw the model and then use programs to translate the model into a formal notation [13]. Figure 9 shows the system architecture for the bi-directional grammar-based conceptual modelling framework. Figure 9 also shows the underlying architecture of the framework where a bi-directional grammar is responsible to translate the CNL specification into a formal representation (RuleML JSON[4]) and the other way around. The RuleML JSON representation is mapped into a reasoner friendly description logic representation for verification. The RuleML JSON representation is further processed to generate a conceptual model. The framework works in both directions (from a CNL specification to a conceptual model and vice versa).

12.3 Characteristics

The bi-directional grammar-based conceptual modelling framework has several advantages. Firstly, it uses a formal representation to generate conceptual models. Secondly, it makes the conceptual modelling process easy to understand by providing a framework for writing specifications, generating visualisations (with the help of a case tool or a conceptual model generation system [49]), and verbalisations of these visualisations. Thirdly, it makes the conceptual models machine processable like other logical approaches, and therefore supports verification. Furthermore, the support for verbalisation facilitates better understanding of the modelling process, which is only available in limited form in other conceptual modelling frameworks and helps users to manipulate the models in a round-tripping fashion.

13 TRANSFORMATION OF USER STORIES INTO UML USE-CASE DIAGRAMS

13.1 Motivation

For writing functional requirements, understanding user stories is an important task. Meryem's framework [11] proposes a technique of transforming users' stories into corresponding UML use-case diagrams. The framework of Elallaoui et al. uses a **model-driven architecture** (**MDA**) for all the transformations. These models represent the highest level of abstraction, which show the requirements understandably by the domain experts.

[3]http://www.hermit-reasoner.com/.
[4]https://wiki.ruleml.org/index.php/RuleML_in_JSON.

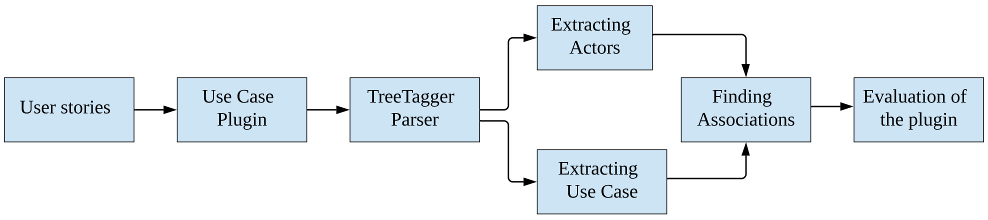

Fig. 10. Transformation of user stories to UML Use-Case diagrams.

13.2 Architecture

The framework of Elallaoui et al. mainly constructs the **computational independent model** (**CIM**) by automatically transforming a set of user stories into UML use-case diagrams. The main benefit of user stories as well as the transformation work is already carried out in the MDA approach. The framework uses a TreeTagger parser along with other NLP techniques for transforming user stories. This parse tree helps in finding actors, use cases, and their association/relationship based on the plugin that the framework developed. Figure 10 presents the steps for conversion of user stories to UML use-case diagrams. For extracting UML use-case diagrams, the algorithm takes a set of user stories and puts them into a text file. The technique applies POS tags for categorising terms into proper nouns, verbs, and so forth. Each term of the user stories has been implemented by a few JAVA classes. For example, the *ElementFactory* class creates actors and their use cases. The *ModelFactory* class creates a model by adding elements such as actors and their association.

13.3 Characteristics

The framework of Elallaoui et al. is an NLP-based use-case building framework that analyses user stories. The framework automatically converts user stories into corresponding use-case diagrams by extracting actors and their relationships. The advantage of generating UML use-case diagrams is that they are easy to understand and designers can also understand requirements without any difficulties. The framework has been evaluated based on the generated models by comparing the accuracy between automatically and manually detected actors, use cases, and respective relationships. Finally, for a specific case study, recall and precision scores of 98% have been reported.

14 SEMI-AUTOMATED CONCEPTUAL MODEL EXTRACTION SYSTEM

14.1 Motivation

Semi-Automated Conceptual Model Extraction System (**SACMES**) [44] has been developed through the integration of a linguistic-based approach with an ontology-based approach supported by a minimum level of human intervention to resolve NL ambiguities. The need for a domain-independent ontology has been emphasised in this study. To ease the difficulty and save time, the recursive update of the domain-independent ontology has been proposed in this literature through human intervention. The system has been called semi-automated as human intervention has been integrated with the system. However, the evaluation of the system has shown that the need for human intervention will reduce as the behaviour database flourishes continuously.

14.2 Architecture

The developed system integrates five tools that include Stanford CoreNLP, WordNet Ontology (WordNet 3.1), **Linguistic Rules** (**LR**) as human intervention, **Conceptual Model Ontology** (**CMO**), and **User History Knowledge Base** (**UHKB**) using Microsoft SQL Server consisting of two sectors named (a) **Entities History Knowledge Base** (**EHKB**) and (b) **Relationships**

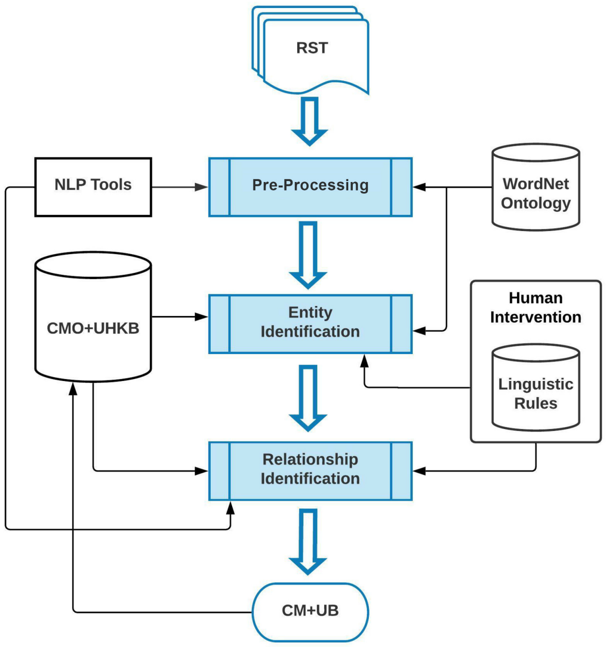

Fig. 11. Semi-automated conceptual model extraction system (SACMES).

History Knowledge Base (**RHKB**). The system architecture of SACMES is displayed in Figure 11. The system takes a **requirements specification text** (**RST**) as input. The processing of the input is executed in three phases: (a) pre-processing phase, (b) entity identification phase, and (c) relationship identification phase. In the pre-processing phase the NLP tool identifies potential noun phrases. After that, the NLP tool and WordNet work together to eliminate nouns and noun phrases that are unlikely to be entities. The final output of this phase is the potential entities named as candidate entities. In the entity identification, CMO along with WordNet work to map the nouns into entities or to discard them. Additionally, human interventions are used as linguistic rules to apply domain importance and resolve issues related to multiple attributes of entities. If a noun phrase is important within a requirements specification, then it can be mapped into an entity, e.g., a doctor or a patient within a hospital-related specification is an example of domain importance. Applied human interventions are saved in the EHKB for improving the knowledge base of the system. In the relationship identification phase, relationships between the entities along with the cardinality restrictions are being determined using the RHKB, the NLP tool, and the CMO. In this system, human interventions are used to demystify "need to know" rules and ambiguities. The patterns of human interventions are updated in the RHKB. Conceptual models are generated as the final output of this process. Finally, the CMO is being updated with the conceptual models and UHKB is updated with the user behaviour during this process.

14.3 Characteristics

SACMES has improved the extraction of conceptual models from NL using human intervention. The system mainly focuses on the creation of an **Entity Relationship Diagram** (**ERD**) by identification of entities and relationships. Entity and entity relationships with cardinality restrictions are represented as formal knowledge. Additionally, the user behaviour is stored in the database for continuous improvement of the system. Moreover, semantic round-tripping is not considered

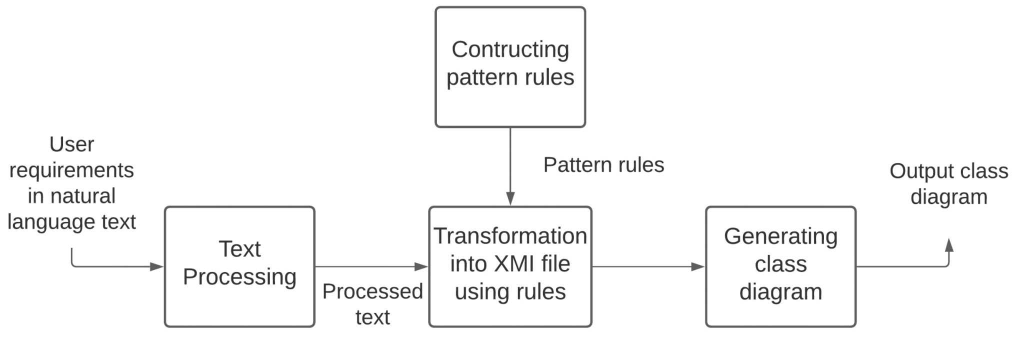
Contructing pattern rules
Pattern rules
User requirements in natural language text
Text Processing
Processed text
Transformation into XMI file using rules
Generating class diagram
Output class diagram

Fig. 12. Methodology of ABCD.

while taking input into the system. The evaluation of the system showed that the performance of the users developing conceptual models can be improved using SACMES. Precision and recall of the users have increased by 6% and 13%, respectively for entity identification and 14% and 23%, respectively, for relationship identification. The performance of the model has also improved by processing more NL requirements and continuously reducing the need of human intervention.

15 ABCD

15.1 Motivation

The ABCD tool [4] was developed with a view to mapping user's requirements specifications to class diagrams automatically. In software engineering, automated class diagram generation from verbose requirements specifications written in different NL texts by the users, is an area of huge interest to software engineers and the research community [4]. However, this remains a quite difficult task, since NLs can be highly ambiguous and unpredictable for a machine to process. As a result, user requirements engineering has been well explored in the literature, where the researchers and developers have proposed different techniques and tools [4]. To this end, the prime goal of this work is to propose a novel technique to enable the automatic extraction of UML class diagrams from the user-specified textual requirements for the development of software. At present, UML class diagrams have emerged as one of the most applicable tools for not only reflecting appropriately on the user's verbose NL–based requirements but also envisioning the software development plan. Precisely, a UML class diagram enhances the interpretation of the designed model throughout the development process of the corresponding software. In the course of generating these class diagrams, the work also aims to identify a diverse range of UML concepts with benchmark accuracy.

15.2 Architecture

The authors present an improved approach for automating the transformation from NL user's requirements to UML class diagrams with the help of the ABCD tool. Motivated by a MDA approach, this work comprises the following three major steps (Figure 12):

— processing the user requirements text (source model) into a processed text (target model),
— defining pattern rules (meta-model transformations) to transform the processed text into an **XML metadata interchange** (**XMI**) file (target meta-model), and
— generating the class diagrams from the XMI file.

The text processing step involves sentence splitting, tokenisation, PoS tagging, and syntactic parsing. To do these tasks, statistical NLP techniques are employed using the open source Stanford NLP toolkit. The next task, pattern rules identification, requires a rule-based technique as it involves semantic analysis of the processed text to extract the relevant information about candidate concepts of the UML class diagrams. More specifically, following human knowledge expertise, a set of patterns/rules is created using regular expressions to match an NL sentence to identify the candidate concepts of the UML class utilising the text processing information (PoS, dependencies). Three types of patterns are recognised in this study (in order of priority): attributes (class name and attributes), relationships (inheritance, aggregation, composition, and association), and methods. These extracted concepts for UML class diagrams are saved in an XMI file. Finally, a CASE tool, ArgoUML, is used to automatically generate the UML class diagram from the XMI file [8].

15.3 Characteristics

ABCD focuses on generating a UML class diagram; particularly on creating pattern rules using regular expressions for generating the XML metadata from the NL text. However, the authors do not consider semantic round-tripping in this study. The ABCD tool is evaluated in terms of three metrics: precision (P), recall (R), and overgeneration (O), where ABCD outperforms three metrics: ($P = 92\%$, $R = 89\%$, $O = 26\%$) of other popular automated class diagram generation tools (e.g., CM-Builder [22], LIDA, DC-Builder, UMLG). Moreover, ABCD can extract more UML concepts (e.g., aggregations, multiplicity, compositions) than many automated tools.

16 DISCUSSION

Most of the existing NL–based CM frameworks derive conceptual models from NL text but cannot generate NL text from conceptual models. A bi-directional grammar-based conceptual modelling framework can make the communication transparent between a domain expert and a knowledge engineer by allowing semantic round-tripping between restricted NL text and conceptual models. But the problem arises when a conceptual modelling framework uses unrestricted NL text for writing specifications because of its ambiguity, inconsistency, and complexity. Domain experts feel comfortable writing a specification in unrestricted NL but computers cannot process this text completely; therefore, semantic round-tripping becomes impossible. Table 1 below shows a comparison of the NL–based CM frameworks or systems that we discussed in this article in terms of their generated diagrams, use of RNL, formal representation, support for round-tripping, and verification.

Diagram Type: Among the discussed frameworks, the framework by Saeki et al. [51] does not support any specific diagram type. The framework by Saeki et al. derives a formal program from a textual specification. UML class diagram is the most common diagram type that is generated by Mich and Garigliano's NL–OOPS [42], Harmain and Gaizauskas's CM-Builder [22], Ambriola and Gervasi's CIRCE [2], Deeptimahanti's UMGAR by Deeptimahanti and Babar [10], RACE by Ibrahim and Ahmad [29], Meryem's the framework by Elallaoui et al. [11], Hossain's bi-directional grammar-based framework [23], and ABCD by Karaa et al. [4]. ER diagram is supported by Omar and Baryannis's SACMES [44], Hossain and Schwitter's bi-directional grammar-based framework [27], and Ambriola and Gervasi's CIRCE [2] framework. ORM^{zero} diagram is only supported by Hossain's bi-directional grammar-based framework [23].

Use of CNL: Among all the discussed frameworks Ambriola and Gervasi's CIRCE [2], Selway's cognitive grammar-based framework [55], and Hossain's bi-directional grammar-based framework [23] use restricted/CNLfor conceptual modelling. Unlike CIRCE and Selway's cognitive grammar-based framework, Hossain's bi-directional grammar-based framework advocates that an authoring tool should be used while writing specifications using a CNL to make the writing process easier for the domain experts and knowledge engineers.

Knowledge Representation Formalism: Among the discussed frameworks, Mich and Garigliano's NL–OOPS [42], Harmain and Gaizauskas's CM-Builder [22], Ilieva and Ormandjieva's

Table 1. Comparison of the NL–Based CM Frameworks

System or Framework Name	*Comparison Criteria*				
	Diagram Type	***Use of RNL***	***KR Formalism***[a]	***Round-Tripping***	***Verification Support***
Saeki et al.'s Framework [51]	N/A	No	N/A	No	No
Mich and Garigliano's NL-OOPS [42]	UML Classs Diagram	No	FOL[b]	No	Yes
Harmain and Gaizauskas's CM-Builder [22]	UML Class Diagram	No	FOL[b]	No	No
Ilieva and Ormandjieva's MDAATUR [30]	Use-Case Path, Hybrid Activity, and Domain Model	No	FOL[b]	No	No
Ambriola and Gervasi's CIRCE [2]	UML Class, ER, Dataflow, and Use-Case Diagram	Yes	FOL[b]	No	Yes
Deeptimahanti and Babar's UMGAR [10]	Use Case, UML Class Diagram	No	N/A	No	No
Ibrahim and Ahmad's RACE [29]	UML Class Diagram	No	N/A	No	Yes
Meryem's Framework [11]	Use-Case Diagram	No	N/A	No	No
Selway's Cognitive Grammar-Based Framework [55]	SBVR Model	Yes	FOL[b] Modal Logic	No	No
Hossain's Bi-directional Grammar-Based Framework [23]	UML Class Diagram, ER Diagram, and ORM^{zero} Diagram	Yes	DL[c]	Yes	Yes
Karaa et al.'s ABCD [4]	UML Class Diagram	No	N/A	No	No
Omar and Baryannis's SACMES [45]	ER Diagram	No	N/A	No	Yes

[a]KR = Knowledge Representation. [b]FOL = First-Order Logic. [c]DL = Description Logic.

MDAATUR [30], and Ambriola and Gervasi's CIRCE [2] use first-order logic (FOL) [26] as an underlying formalism for the conceptual models. Selway's cognitive grammar-based framework [55] uses both FOL and Modal logic [6] as the underlying formalism whereas Hossain's bi-directional grammar-based framework [23] uses **Description Logic** (**DL**) [26]. DL is a computationally complete and decidable fragment of FOL; therefore, validation and consistency check can be completed at a reasonable time in Hossain's bi-directional grammar-based framework.

Round-Tripping: All CM frameworks that we discussed in this article derive conceptual models or prototype programs from NL text. They have similar motivations and all of them are unidirectional except the bi-directional grammar-based approach [23]. Hossain's bi-directional grammar-based approach supports semantic round-tripping in conceptual modelling by leveraging a CNL [27]. It generates SQL code from the CNL specification that can be used in building a schema for the information system.

Verification Support: Among the frameworks listed in Table 1, Mich and Garigliano's NL–OOPS [42], Ambriola and Gervasi's CIRCE [2], Ibrahim and Ahmad's RACE [29], and Hossain's bi-directional grammar-based framework [23] support verification during conceptual modelling. This verification support mainly includes consistency and redundancy checking of the generated conceptual models.

Saeki's software development prototype is implemented to aid knowledge engineers in eliciting requirements and to derive a formal specification from an informal one. Mich and Garigliano's NL–OOPS tool uses a complex NLP system called LOLITA to extract necessary information from NL and builds a conceptual model from it. NL–OOPS supports verification (e.g., inconsistency and redundancy checking) of the generated model. Harmain and Gaizauskas's CM-Builder follows a similar approach to NL-OOPS but does not have any verification support. Among these CM frameworks, Ambriola and Gervasi's CIRCE system uses a CNL for writing a domain-specific glossary and rules for conceptual modelling. CIRCE also supports verification of the specification. Selway's cognitive grammar-based approach has been specially designed for domain experts and uses a CNL (SBVR Structured English) for writing specifications. But Selway's cognitive grammar-based approach does not support verification. Hossain and Schwitter's bi-directional grammar-based approach supports both verification and semantic round-tripping in conceptual modelling by using a CNL. Unlike CIRCE and Selway's cognitive grammar-based framework, Hossain and Schwitters's bi-directional grammar-based framework advocates that an authoring tool should be used while writing specifications using a CNL to make the writing process easier for the domain experts and knowledge engineers. Whereas, ABCD by Karaa et al. blends the statistical and pattern recognition properties of NLP techniques to generate UML class diagrams from user requirements. Precisely, more than 1,000 pattern rules, capable of extracting complex UML concepts such as multiplicity, aggregations, compositions, and generalisations have been incorporated in ABCD. However, similar to Hermain and Gaizauskas's CM-Builder and Selway's approach, it does not support automatic verification. Unlike the previous approaches, SACMES [44] introduces automation to generate conceptual models based on a domain-independent ontology [25]. One of the features of this system is the use of human intervention aligning with the automated output to minimise ambiguity and thereby improve the performance of the system. Moreover, continuous storage of user behaviour and the output model improves the automation capability with time.

Ilieva and Ormandjieva's MDAATUR [30], Deeptimahanti and Babar's UMGAR [10], Ibrahim and Ahmad's RACE [29], and the framework by Elallaoui et al. [11] generate conceptual models from unrestricted NL text. All these frameworks except MDAATUR do not use any formal representations for the generated conceptual models. MDAATUR uses a semantic network to formally represent the conceptual models. In addition to that, due to the use of unrestricted NL specifications, these frameworks do not support round-tripping like the system of bi-directional grammar-based framework [27]. However, RACE validates conceptual models by cross-checking the number of valid concepts generated using their system and manual human interventions. The advantage of these works is that they perform completely automated analysis of the NL text. No manual translation is required by the user. The parsing performed by these approaches is normally shallow and incomplete because these systems use rules that search for syntactic patterns by ignoring different parts of a sentence to extract elements. Therefore, it becomes difficult for these approaches to provide detailed feedback to the user about possible errors, ambiguities, or inconsistencies. Another vital problem with these approaches is their variable performance. Apart from that, modern NLP techniques are statistical and they require a large number of training data to work correctly. Moreover, for supervised machine learning algorithms, domain expertise is needed to annotate the requirements. These requirements are most often classified assets of companies and domain experts are seldom available to annotate these datasets. To overcome some of the problems of classified assets, federated learning[5] could be used in a NL–based conceptual model generation framework [48].

Furthermore, an algorithm trained on a company-specific requirements text may not work on a different company-specific requirements text. A possible solution has been suggested by Ferrari [12] to build tools that can be trained on the job (which gather data and learn their task from domain experts). Another interesting work [57] is the comparison between a textual description and a process model where conflicts between textual and model-based process descriptions are detected. To make process information accessible to different collaborators, many organisations maintain textual process descriptions alongside graphical process models [57]. However, there is a risk that the model and the text become misaligned when changes are not applied to both of them consistently. The main idea behind the work in [57] is to support organisations in keeping their process descriptions consistent by providing a way to automatically identify inconsistencies between a process model and a corresponding textual specification. The bi-directional grammar-based framework [27] provides a solution to this problem by providing semantic round-tripping between the specifications and conceptual models.

Apart from the model-driven engineering techniques, a recent method of low or no code software development is getting much attention. Low code software development has changed the dynamics of software engineering by introducing the capability of citizen developers to build tools without having a programming background [58]. Users can drag-and-drop reusable actions using visual editors from low code platforms into workflows to speed up development. Direct integrations, low code API accessibility, drag-and-drop workflow designers, and workflow testing/prototyping resources are features that the majority of low code development platforms share [59]. A low code development tool may also contain monitoring, resource management, and sophisticated tools that speed up DevOps [36], depending on the platform. Low code development provides us with several benefits including reduction of software development cost, better business agility [1], rapid iterations [59], better manufacturing capability [52], and so forth. Despite having several advantages, low or no code development techniques suffer from several shortcomings: without professional developers, applications made with drag-and-drop platforms may be difficult to update to satisfy new technical requirements [32]. Moreover, the runtime performance of the application may not be as efficient, and it may be harder to integrate it with larger software systems of an organisation [59]. Likewise, some no code tools that create mobile applications, called online application generators, can have security issues [1]. Similarly, for a citizen developer who does not know coding, debugging would be quite a challenge [32]. While low code and no code tools contribute to the development phase of a software, formalisation and representation of the NL–based requirement specifications of the software still remain as a distinct significant task, which usually comes prior to the software development phase (e.g., using low code or no code).

17 FUTURE WORK

Existing NL–based conceptual modelling frameworks mainly focus on automatic requirements elicitation. Some of them use controlled NL as a specification language and support verification of the specification. However, none of them uses description logic as a formal representation language and none of them addressed semantic round-tripping in conceptual modelling, except the bi-directional grammar-based approach. We envisage an ideal conceptual modelling framework that combines the advantages of the existing frameworks; for example, a conceptual modelling

[5]https://ai.googleblog.com/2017/04/federated-learning-collaborative.html.

framework that can support both requirements elicitation and semantic round-tripping (verbalisation). Additionally, the visualisation of the conceptual models can be improved using more generic frameworks; for example, if the resulting conceptual model is in a generic format that can be transformed into an ER diagram, UML class diagram, or use-case diagram according to the requirements of the system. This can save efforts and resources of developing multiple tools for multiple sets of uses. Moreover, in the case of requirements elicitation, the framework could use open information extraction with a human in the loop who refines and curates the vocabulary, and for verbalisation, it can use a CNL. In this way, it can support both requirements elicitation from unrestricted NL specifications and verbalisation. The verbalisation can then be used as a formal specification to support semantic round-tripping in such a framework. In contrast, the use of machine learning techniques, such as generative models, can be introduced through adaptive modification of the models according to the requirements. In addition, a universal dataset of requirements texts can resolve the prevailing difficulties of different datasets for different instances. To this end, exploring different NLP techniques or designing a blend of NLP technology and human intelligence for the automatic formalisation of requirements text into CNL (i.e., automated or semi-automated preprocessing of data), remains a direction for potential future work. On a similar note, integrating the discussed methodologies in the formal verification, semantic round-tripping, and development of testing frameworks for low code or no code software development techniques can be an interesting research direction to explore.

18 CONCLUSION

This article presents a comparative study about some older and some newer NL–based conceptual modelling frameworks with a particular focus on motivation, architecture, and characteristics that include the major aspects of software requirements specification and formalisation such as different diagram types, use of natural and restricted forms of NL, knowledge representation formalisms, semantic round-tripping, and verification support. We find that the majority of these frameworks ignored the use of CNL in requirements specification, although it could significantly enhance the effectiveness of the frameworks. Moreover, semantic round-tripping is another useful concept that could ameliorate the verification process of the conceptual modelling diagrams generated by these frameworks and enable diagram-based requirements specification by the users. Additionally, verification of the frameworks needs to be automated and formalised (e.g., inclusion of semantic round-tripping), as most of these frameworks lack automatic verification support. Considering these perspectives, the article critically reflects on several notable conceptual modelling frameworks, pointing out the potential opportunities for further developments in this research domain.

REFERENCES

[1] Md Abdullah Al Alamin, Sanjay Malakar, Gias Uddin, Sadia Afroz, Tameem Bin Haider, and Anindya Iqbal. 2021. An empirical study of developer discussions on low-code software development challenges. In *Proceedings of the 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR'21)*. IEEE, 46–57.

[2] Vincenzo Ambriola and Vincenzo Gervasi. 2006. On the systematic analysis of natural language requirements with circe. *Automated Software Engineering* 13, 1 (2006), 107–167.

[3] Richard Barker. 1990. *CASE Method: Entity Relationship Modelling*. Addison-Wesley Longman Publishing Co., Inc.

[4] Wahiba Ben Abdessalem Karaa, Zeineb Ben Azzouz, Aarti Singh, Nilanjan Dey, Amira S. Ashour, and Henda Ben Ghazala. 2016. Automatic builder of class diagram (ABCD): An application of UML generation from functional requirements. *Software: Practice and Experience* 46, 11 (2016), 1443–1458.

[5] Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. 2005. Reasoning on UML class diagrams. *Artificial Intelligence* 168, 1 (2005), 70–118.

[6] Patrick Blackburn, Johan FAK van Benthem, and Frank Wolter. 2006. *Handbook of Modal Logic*. Elsevier.

[7] Diego Calvanese. 2013. *Description Logics for Conceptual Modeling Forms of Reasoning on UML Class Diagrams*. EPCL Basic Training Camp 2012-2013.

[8] Daniel Cruz, Eduardo Figueiredo, and Jabier Martinez. 2019. A literature review and comparison of three feature location techniques using ArgoUML-SPL. In *Proceedings of the 13th International Workshop on Variability Modelling of Software-Intensive Systems*. 1–10.

[9] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. 2002. GATE: An architecture for development of robust HLT applications. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 168–175.

[10] Deva Kumar Deeptimahanti and Muhammad Ali Babar. 2009. An automated tool for generating UML models from natural language requirements. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 680–682.

[11] Meryem Elallaoui, Khalid Nafil, and Raja Touahni. 2018. Automatic transformation of user stories into UML use case diagrams using NLP techniques. *Procedia Computer Science* 130 (2018), 42–49. (Also in *Proceedings of the 9th International Conference on Ambient Systems, Networks and Technologies (ANT'18)/The 8th International Conference on Sustainable Energy Information Technology (SEIT'18)/Affiliated Workshops*).

[12] Alessio Ferrari. 2018. Natural language requirements processing: From research to practice. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. ACM, 536–537.

[13] Pablo R. Fillottrani, Enrico Franconi, and Sergio Tessaris. 2012. The ICOM 3.0 intelligent conceptual modelling tool and methodology. *Semantic Web* 3, 3 (2012), 293–306.

[14] Enrico Franconi, Alessandro Mosca, and Dmitry Solomakhin. 2012. ORM2: Formalisation and encoding in OWL2. In *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems."* Springer, 368–378.

[15] Joseph Frantiska. 2018. Entity-relationship diagrams. *Visualization Tools for Learning Environment Development*. Springer, 21–30.

[16] Robert Gaizauskas and Kevin Humphreys. 1997. Using a semantic network for information extraction. *Natural Language Engineering* 3, 2 (1997), 147–169.

[17] Roberto Garigliano, Agnieszka Urbanowicz, and David J. Nettleton. 1998. University of Durham: Description of the LOLITA system as Used in MUC-7. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*. https://www.aclweb.org/anthology/M98-1005.

[18] Vincenzo Gervasi and Vincenzo Ambriola. 2001. The Cico domain-based parser. In *Convegno Finale del Progetto MURST cofin "Agenti Intelligenti: Interazione ed Acquisizione di Conoscenza."*

[19] Terry Halpin. 2009. Object-role modeling. *Encyclopedia of Database Systems*. Springer, 1941–1946.

[20] Terry Halpin. 2019. Reference scheme modeling. *New Perspectives on Information Systems Modeling and Design*. IGI Global, 227–254.

[21] Jorge Hankamer. 1989. *Morphological Parsing and The Lexicon*. MIT Press. 392 pages.

[22] H. M. Harmain and Robert Gaizauskas. 2003. CM-Builder: A natural language-based CASE tool for object-oriented analysis. *Automated Software Engineering* 10, 2 (2003), 157–181.

[23] Bayzid Ashik Hossain. 2022. *Specifying and Verbalising Conceptual Models using Controlled Natural Language*. PhD dissertation. Macquarie University. https://doi.org/10.25949/20647662.v1. Accessed 06-02-2023.

[24] Bayzid Ashik Hossain, Gayathri Rajan, and Rolf Schwitter. 2019. CNL-ER: A controlled natural language for specifying and verbalising entity relationship models. In *Proceedings of the 17th Annual Workshop of the Australasian Language Technology Association*. Australasian Language Technology Association, Sydney, Australia, 126–135. https://www.aclweb.org/anthology/U19-1017.

[25] Bayzid Ashik Hossain, Abdus Salam, and Rolf Schwitter. 2020. A survey on automatically constructed universal knowledge bases. *Journal of Information Science* 47 (2020), 0165551520921342.

[26] Bayzid Ashik Hossain and Rolf Schwitter. 2018. Specifying conceptual models using restricted natural language. In *Proceedings of the Australasian Language Technology Association Workshop 2018*. 44–52. https://www.aclweb.org/anthology/U18-1005.

[27] Bayzid Ashik Hossain and Rolf Schwitter. 2020. Semantic round-tripping in conceptual modelling using restricted natural language. In *Australasian Database Conference*. Springer, 3–15.

[28] Kevin Humphreys, Robert Gaizauskas, Saliha Azzam, Christian Huyck, Brian Mitchell, Hamish Cunningham, and Yorick Wilks. 1998. University of Sheffield: Description of the LaSIE-II system as used for MUC-7. In *Proceedings of the S7th Message Understanding Conference (MUC-7)*.

[29] Mohd Ibrahim and Rodina Ahmad. 2010. Class diagram extraction from textual requirements using natural language processing (NLP) techniques. In *Proceedings of the 2010 2nd International Conference on Computer Research and Development*. IEEE, 200–204.

[30] M. G. Ilieva and Olga Ormandjieva. 2006. Models derived from automatically analyzed textual user requirements. In *Proceedings of the 4th International Conference on Software Engineering Research, Management and Applications (SERA'06)*. IEEE, 13–21.

[31] Mustafa Jarrar, C. Maria, and Keet Paolo Dongilli. 2006. *Multilingual Verbalization of ORM Conceptual Models and Axiomatized Ontologies*. Citeseer.
[32] Faezeh Khorram, Jean-Marie Mottu, and Gerson Sunyé. 2020. Challenges and opportunities in low-code testing. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 1–10.
[33] Philippe Kruchten. 2004. *The Rational Unified Process: An Introduction*. Addison-Wesley Professional.
[34] Tobias Kuhn. 2010. *Controlled English for knowledge representation*. Ph.D. Dissertation. University of Zurich.
[35] Ronald W. Langacker and Ronald Langacker. 2008. *Cognitive Grammar: A Basic Introduction*. Oxford University Press.
[36] Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. 2019. A survey of DevOps concepts and challenges. *ACM Computing Surveys (CSUR)* 52, 6 (2019), 1–35.
[37] Domenico Lembo, Daniele Pantaleone, Valerio Santarelli, , and Domenico Fabio Savo. 2016. Easy OWL drawing with the graphol visual ontology language. In *Proceedings of the 15th International Conference on the Principles of Knowledge Representation and Reasoning*.
[38] Domenico Lembo, Daniele Pantaleone, Valerio Santarelli, and Domenico Fabio Savo. 2016. Eddy: A graphical editor for OWL 2 ontologies. In *Proceedings of IJCAI*. 4252–4253.
[39] François Lévy and Adeline Nazarenko. 2013. Formalization of natural language regulations through SBVR structured English. In *Proceedings of the International Workshop on Rules and Rule Markup Languages for the Semantic Web*. Springer, 19–33.
[40] Carsten Lutz. 2002. Reasoning about entity relationship diagrams with complex attribute dependencies. In *Proceedings of the International Workshop in Description Logics 2002 (DL'02)*, number 53 in CEUR-WS (http://ceur-ws.org). 185–194.
[41] Luisa Mich. 1996. NL-OOPS: From natural language to object oriented requirements using the natural language processing system LOLITA. *Natural Language Engineering* 2, 2 (1996), 161–187.
[42] Luisa Mich and Roberto Garigliano. 2002. NL-OOPS: A requirements analysis tool based on natural language processing. *WIT Transactions on Information and Communication Technologies* 28 (2002).
[43] Antoni Olivé. 2007. *Conceptual Modeling of Information Systems*. Springer-Verlag, Berlin, 1–10 pages
[44] Mussa Omar and George Baryannis. 2020. Semi-automated development of conceptual models from natural language text. *Data & Knowledge Engineering* 127 (2020), 101796.
[45] Mussa Omar and George Baryannis. 2020. Semi-automated development of conceptual models from natural language text. *Data & Knowledge Engineering* (2020), 101796. https://doi.org/10.1016/j.datak.2020.101796
[46] The Apache Software Foundation. 2018. Apache OpenNLP. https://opennlp.apache.org/. Accessed 06-02-2023.
[47] Gerard O'Regan. 2017. Unified modelling language. *Concise Guide to Software Engineering*. Springer, 225–238.
[48] K. M. Jawadur Rahman, Faisal Ahmed, Nazma Akhter, Mohammad Hasan, Ruhul Amin, Kazi Ehsan Aziz, A. K. M. Muzahidul Islam, Md. Saddam Hossain Mukta, and A. K. M. Najmul Islam. 2021. Challenges, applications and design aspects of federated learning: A survey. *IEEE Access* 9 (2021), 124682–124700.
[49] Gayathri Rajan. 2019. Graphical rendering of conceptual models. Bachelor Thesis, School of Engineering, Macquarie University, Sydney, Australia.
[50] Doug Rosenberg and Kendall Scott. 1999. *Use Case Driven Object Modeling with UML*. Springer.
[51] Motoshi Saeki, Hisayuki Horai, and Hajime Enomoto. 1989. Software development process from natural language specification. In *Proceedings of the 11th International Conference on Software Engineering*. IEEE, 64–73.
[52] Raquel Sanchis, Óscar García-Perales, Francisco Fraile, and Raul Poler. 2019. Low-code as enabler of digital transformation in manufacturing industry. *Applied Sciences* 10, 1 (2019), 12.
[53] Douglas C. Schmidt. 2006. Model-driven engineering. *Computer* 39, 2 (2006), 25.
[54] Matt Selway, Georg Grossmann, Wolfgang Mayer, and Markus Stumptner. 2015. Formalising natural language specifications using a cognitive linguistic/configuration based approach. *Information Systems* 54 (2015), 191–208.
[55] Matt Ryan Selway. 2016. *Formal Models from Controlled Natural Language via Cognitive Grammar and Configuration*. Ph.D. Dissertation. University of South Australia.
[56] Harald Störrle. 2017. How are conceptual models used in industrial software development?: A descriptive survey. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. ACM, 160–169.
[57] Han van der Aa, Henrik Leopold, and Hajo A. Reijers. 2017. Comparing textual descriptions to process models—The automatic detection of inconsistencies. *Information Systems* 64 (2017), 447–460.
[58] Robert Waszkowski. 2019. Low-code platform for automating business processes in manufacturing. *IFAC-PapersOnLine* 52, 10 (2019), 376–381.
[59] Marcus Woo. 2020. The rise of no/low code software development—No experience needed? *Engineering (Beijing, China)* 6, 9 (2020), 960.

Received 4 November 2021; revised 12 February 2023; accepted 2 May 2023