Author pre-print version. Paper accepted for PoEM'25.
Final Paper to be published and copyrighted by Springer.

Talk to me!
Toward Speech-based UML Modeling

Simon Schwantler[1], Stefan Klikovits[2][0000−0003−4212−7029], Haydar Metin[3], Philip Langer[3], and Dominik Bork[1][0000−0001−8259−2297]

[1] Business Informatics Group, TU Wien, Vienna, Austria
simon.schwantler@gmx.at, dominik.bork@tuwien.ac.at
[2] Department of Business Informatics, JKU Linz, Linz, Austria
stefan.klikovits@jku.at
[3] EclipseSource, Vienna, Austria
{hmetin,planger}@eclipsesource.com

Abstract. Modeling is a core task in enterprise systems engineering. The use of graphical modeling editors, however, remains cumbersome in general and poses a significant challenge for users with disabilities. Natural language processing (NLP) and intent recognition are at the forefront of making many technologies more accessible and intuitive by allowing users to engage using natural language. This paper presents a natural language interface (NLI) for speech-based UML model interaction that leverages state-of-the-art NLP technologies to enable speech-based modeling. We provide a workflow for the creation of NLIs for modeling editors, a proof-of-concept integration of this approach into the BIGUML open-source modeling editor, and an empirical evaluation that shows promising results in intent recognition, the effectiveness of model creation, and usability. Thereby, this paper makes significant contributions towards more natural, inclusive, and accessible modeling.

Keywords: Natural language interface · Speech-based interaction · GLSP · Modeling tool · UML · Accessibility.

1 Introduction and Background

In the practice of enterprise modeling, the proficient utilization of modeling editors is crucial. The use of graphical modeling editors such as BIGUML [28], Papyrus [24], StarUML[1], or Visual Paradigm[2] is the day-to-day business of software engineers and domain experts in the context of enterprise and software modeling. However, efficient editor-supported modeling typically requires numerous long and complex interaction sequences involving many switches between keyboard and mouse, and the skilled orchestration of both input devices. This form of interaction, which is common to all current graphical modeling editors, often presents accessibility barriers for users with physical disabilities [36].

[1] https://staruml.io/
[2] https://www.visual-paradigm.com/

While recent years have seen notable efforts to improve accessibility in software engineering and web development [3], the field of conceptual modeling still lags behind [37]. Innovative means of model interaction and improved accessibility in modeling have been largely neglected, making the discipline less inclusive and failing to accommodate all modelers' varied needs and abilities. This oversight not only excludes people with disabilities from participating in conceptual modeling and model-driven engineering, but also neglects the potential benefits that accessible interfaces provide, as seen by the widespread adoption of voice assistants (e.g. Siri, Alexa) in everyday homes. Enhancing accessibility in modeling could thus promote inclusion [8] and enable users of all abilities to engage. It further offers potential for more natural means of editor interaction over traditional mouse and keyboard inputs [26].

Natural language processing (NLP) and intent recognition are at the forefront of making software applications more accessible by allowing users to specify their intent in natural language. Nonetheless, the complexity of human language poses challenges, including e.g., ambiguity, colloquial language (a.k.a. *slang*), regional accents and dialects, and speech impairments. Overcoming these challenges would render conceptual modeling more intuitive, more usable because of an increase in modeling efficiency, and more accessible to users without technical training or users facing some form of disability.

This paper introduces speech-based modeling using a natural language interface (NLI) for BIGUML [29], a web-based, extensible, and open-source UML modeling editor. We chose UML class diagrams as the target language because of the availability of a publicly used modeling editor.

Recently, Large Language Models (LLMs) have been studied for conceptual model creation from natural language descriptions [1,17,18,23,35,39]. These approaches rely on one-shot model generation, with the output in textual format (e.g., PlantUML). This process, however, has three main drawbacks: First, model creation is an iterative process of editing steps (adding, deleting, and refactoring), rather than one-shot generation. Second, LLMs mainly support standard languages such as UML or SysML, with limited support for domain-specific languages (DSLs). Third, for larger models, LLMs frequently hallucinate or diverge from intended meaning (e.g. due to ambiguous input).

Our approach overcomes these drawbacks by relying on purpose-trained BERT models. This renders our approach more generally applicable to other modeling languages. Technically, this research proposes four core contributions:

1. A generic workflow for creating NLIs for modeling editors;
2. a dataset of UML class diagram modeling commands in natural language that can be used as a reference for other NLI models and other modeling languages;
3. a reference NLI implementation in BIGUML that can be adapted for, and integrated into, other languages and tools; and
4. an empirical evaluation of our approach.

[3] `https://www.w3.org/TR/WCAG21/`

Our software (incl. installation and usage instructions) is freely available online[4]. Likewise, with this paper, we release the dataset generator we developed to create the NLI training data for natural language modeling commands[5].

In the remainder of this paper, Section 2 describes the results of a survey aimed at gathering NL modeling interaction requirements. In Section 3 we then discuss the development of a natural language (NL) server for UML modeling. The integration of this server into BIGUML alongside the results of a systematic empirical evaluation is covered in Section 4. Section 5 presents a discussion of the key findings, before we conclude this paper with an overview of related works in Section 6 and future research directions in Section 7.

2 Interaction Requirements Analysis

Figure 1 outlines the process we followed to develop the NLI extension for BIGUML. In particular, we point towards the two user studies (cf. (1) and (6)), which we performed next to the technical development.

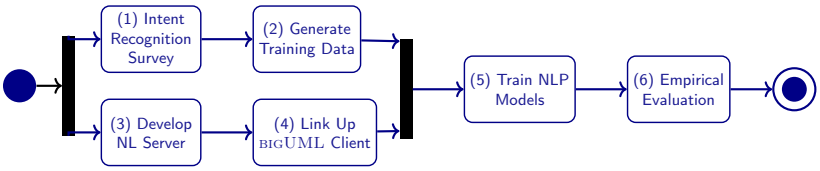


Fig. 1: Research outline

2.1 Intent Recognition Study

The training and the evaluation of the developed NLI are primarily based on a representative dataset of modeling commands and matching *intents* that need to be supported. Thus, we compiled a user survey based on primary UML interaction methods [38], where we asked experienced UML modelers about their choice of voice commands for triggering certain modeling actions on a specific UML model element.

The survey was carried out to learn more about the natural language instructions that modelers intuitively use when engaging with a voice-controlled UML class diagram editor and to identify various ways modelers articulate similar tasks typically performed in UML diagramming tools. The participants were asked to provide full, natural language sentences in response to the modeling scenarios listed in Table 1. We asked the participants to provide as many natural language command examples as they could intuitively think of to steer a modeling tool to execute a certain modeling scenario.

[4] `https://github.com/sschwantler/bigUML/blob/main/NLI_README.md`
[5] `https://github.com/stklik-org/NLI4UML-DataGenerator`

Table 1: User-provided responses on the survey of natural language commands for UML modeling.

Scenario	# Responses	Sample Responses
Create a Class	29	– *Create a class called car.* – *Insert a class Vehicle into the current diagram.* – *I want to add a new Class to the diagram, where the new class' name is "Car".* – *Generate a class for "train".* – *Declare the class bike.*
Add an Attribute to a Class	25	– *Add an attribute called name of type string to the Employee class.* – *Add the string property name to the class Bike.* – *Attach the attribute motor to the class car.*
Add a Method to a Class	22	– *add a method named CalculateSalary with a return type of decimal to the Employee class* – *Create a new operation for the class Bike, which is named getBikeName. The operation expects two input properties which are both of type string.* – *Define the method accelerate for the class car.*
Rename a Class	23	– *change the class name from Employee to Worker.* – *Set the name of the class Bike to Motorcycle.* – *I want to rename the already existing class "Car" to "Vehicle".* – *The class car should actually be called bike.*
Modify an Attribute Name of a Class	26	– *Rename the property "Color" of the class "Car" to "PrimaryColor".* – *Tyre would be a better name for the car class's wheel attribute* – *Rename the car's doors to door.* – *change the attribute name from name to employeeName in the Worker class.*
Modify an Attribute Visibility of a Class	23	– *make the employeeName attribute in the Worker class public.* – *Change/Update the visibility of the attribute name from the class Bike to public.* – *make the "rail with" visible to everyone* – *Make all attributes of the class car public.*
Modify a Method Name of a Class	23	– *rename the method CalculateSalary to ComputeSalary in the Worker class.* – *Edit the method name of the method getName inside the Bike class to change it to getSerialNr.* – *"go somewhere" should now be called "move tram"*
Modify a Method Visibility of a Class	23	– *set the ComputeSalary method in the Worker class to public.* – *I want to change the accessibility of the method "getPrize" of the class "Car" to "private".* – *Make the car class's start method private.* – *Adjust the visibility of the method 'getSupplierInfo' to public in the 'Supplier' class.*
Modify Method Parameters of a Class	24	– *In the method "DriveVihicle" change the parameter "speed" to "double" "kmh".* – *Add one int to the params of the car class's start method.* – *Update the name of the second parameter of the Bike class' method getWithTwoParams to secondParam.*
Add a Relation	25	– *create an association between the Worker and Manager classes.* – *create a generalization from the base class Person to the subclass Teacher.* – *Let the car inherit from vehicle.* – *Create a new relation between the classes car and driver with the following attributes: 1. The relation is called drives. 2. One driver drives one or more cars. 3. The relation type is dependency. A driver requires a car to drive.* – *Add a one to one association from car to person called drives.*
Modify the Positioning of a Class	32	– *The class "Engine" should be below the class "Car".* – *Put the car class to the top right of the diagram.* – *Position the "Vehicle" class north of "Car" class.*
Other Commands	26	– *Delete/clear entire diagram..* – *Undo the last change.* – *Highlight all classes that are subclasses of the 'Vehicle' class.*

Respondents were encouraged to list alternative phrasings and synonyms to better capture the range of commands. For example, the phrases "Create an attribute *XYZ*" or "add a property *XYZ*" represent the same modeling intent. This is an important piece of information for later steps. To be able to cover a wide range of different wordings and avoid bias in the capability of understanding queries. An open-ended question allowed participants to suggest any other commands they believed would be useful for interacting with a UML editor.

The survey was distributed online between March and April 2024 to academic colleagues and industrial partners, who we could attest had prior experience in UML modeling. We gathered a total of 301 user-defined natural language

commands from eleven participants. Table 1 displays an exemplary selection of NL commands for various model interactions, and lists how many synonymous responses we received for each modeling intent.

2.2 Data and Analysis

We reviewed the data to identify common patterns in phrasing and discover how users express modeling commands in natural language. Entries with incomplete or unclear responses were excluded, ensuring that only meaningful data were used in the analysis. Subsequently, different wordings could be derived for each taxonomy entry, which yielded the training data for the NL model.

We then used the Python Natural Language Toolkit (NLTK)[6] scripts to remove stopwords and identify the most frequent terms in all modeling scenarios.

Lessons Learned. Among the survey respondents, we observed a large variety of commands ranging from precise, short instructions to very lengthy commands that are composed of several sub-commands (we discuss the challenge of such composite commands in Section 4). In the following, we summarize other findings; the complete generated dataset is available online[7].

Consistency in Basic Commands Participants showed a clear pattern in command structure across modeling scenarios. Most responses followed a "verb + object" format, e.g. "create class car" or "add an attribute called name".

Use of Synonyms Participants used diverse synonyms to describe the same intention. For instance, "attribute" was also referred to as "field" or "property", and methods were called "functions".

Modification Complexity and Relationships Commands related to modifying attributes, methods, or creating relationships are generally more complex. Participants included specific parameters and visibility settings, suggesting the need for the system to accommodate complex instructions and compound commands with clarity and precision.

Positioning and Layout Control Participants used directional terms such as "below", "next to" or specific coordinates to position elements in the diagram, expecting the ability to adjust layouts relative to other elements.

Focus as a Key Concept One of the most important survey findings was the need to focus certain diagram elements. As one respondent noted, "being able to set a context for future commands [...] would make using such a system a lot easier." This is particularly relevant in complex diagrams with nested classes, attributes, and methods that would require long prompts that become cumbersome for users to phrase and for the system to understand. Thus, instead of saying "rename the attribute 'weight' in the class Car to 'mass' ", the user could say: "focus on attribute weight" followed by "rename to mass".

[6] https://www.nltk.org/

[7] https://github.com/stklik-org/NLI4UML-DataGenerator/blob/main/intents-to-commands-survey.json

2.3 Training Data Generation

We use the cleaned survey data for analysis. We developed a training data generator to enable the generation of large randomized data sets based on the data queries. The reasoning behind the randomization is to enforce training generalization and avoid overfitting. Thus, for each individual intent in our survey (Table 1), we created a set of "pattern templates", which we could automatically populate with specific entities (concrete class and attribute names, geometric coordinate positions, etc.). Thus, given that we obtained a set of alternative formulations for each intent, combined with a large number of possible entities in our catalog, it is possible to flexibly re-generate different training data and experiment with the size of the required training data. The employed script for data generation is available in an open source repository[5].

3 NLI Development

Besides gathering training data, we implemented the NLI as a separate component that could be easily integrated into BIGUML. Our goal was to keep the modifications of the existing BIGUML client and Graphical Language Server Platform (GLSP) [8] server to a minimum. Thus, the task was to develop a middleware layer capable of obtaining standardized GLSP commands from natural language inputs, and translating them into actions that BIGUML's current implementation can recognize and execute without altering its core functionality.

Note, however, that even though the remainder of the section focuses on the development of an NLI for UML class diagrams in BIGUML, the developed architecture could easily be reconfigured for other modeling languages and editors, especially but not exclusively those based on GLSP.

3.1 Architecture

Before developing the tool architecture itself, we defined three design constraints for the subsequent implementation:

1. Avoiding Feature Interference. Any NL capability must not interfere with existing editor interaction methods. Users should always be able to choose between keyboard & mouse, keyboard shortcuts (cf. [37]), or the new NL interaction.

2. Portability. We envision seamless integration across all tools, editors, and platforms. Thus, the NLI a) exposes its functionality by a *REST API* and b) is *containerized* using Docker[9]. These choices aim to improve portability, testability, and deployment with minimal overhead.

[8] `https://github.com/eclipse-glsp/glsp`
[9] `https://www.docker.com/`

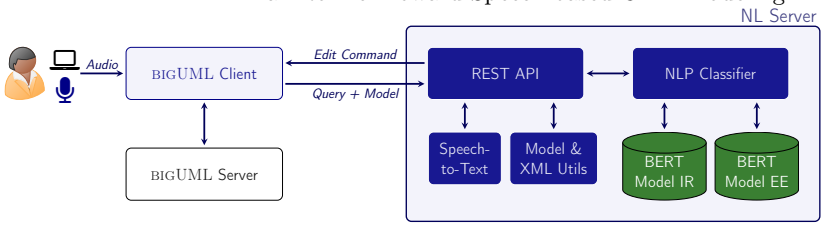


Fig. 2: Application Design and Architecture

3. Performance. For minimal additional strain on the existing client or connection, all components (e.g. speech-to-text transcription) should be implemented at the NL Server and without online dependencies. Thus, the client's only additional responsibility is to capture the audio input.

The conceptual overall application design is shown in Figure 2, with components developed in this project highlighted in the darkblue color. From a high-level view, our application consists of three primary components:

- the **NL Server**, which is the core contribution of our technique. It parses user queries to identify modeling intents and extract relevant UML entities. It exposes a REST API for seamless integration with tools such as BIGUML;
- the minimally adapted **BIGUML Client**, which enables the speech recording and use of NLI; and
- the **BIGUML Server**, which is largely unmodified as the NLI reuses functionality previously developed for modeling use with keyboard and mouse.

3.2 Natural Language Server

Based on the above design decisions, we implemented the NL Server using state-of-the-art NL technologies. Architecturally, the NL Server consists of six components (see Figure 2), of which the NLP Classifier and the two NL models could be seen as the primary components.

NLP classifier The NLP classifier interprets a user's transcribed input and maps it to a concrete modeling action. This task is performed using two separate Bidirectional Encoder Representations from Transformers (BERT) [15] models: 1. one BERT model performs intent recognition (IR), i.e. *what* should be done, while 2. another one is tasked with entity extraction (EE), i.e. *which* UML model entity is affected. Both models were trained on the survey response dataset (see Section 2.3).

Additionally, the NL Server contains three other components, namely

- the `Speech-to-Text` tool transcribes the user's spoken language into text using OpenAI's Whisper API[10].
- the `Model & XML Utils` component, which parses BIGUML's XML encoding and bridges the NL Server and the model; and
- the FastAPI[11]-based REST API to communicate with the BIGUML client.

[10] `https://github.com/openai/whisper`
[11] `https://fastapi.tiangolo.com`

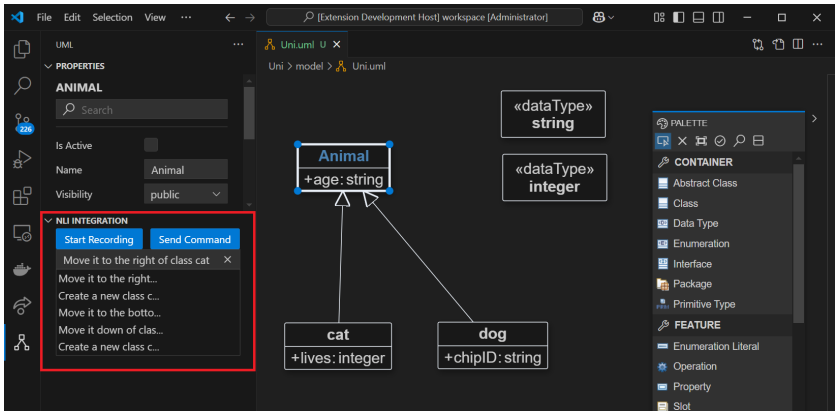


Fig. 3: BIGUML UI Integration of the NL Interface

3.3 Frontend Integration in BIGUML

Our prototype requires only minimal changes to the standard BIGUML interface, as seen in (see Figure 3). Specifically, the new panel contains two buttons to start a voice recording, which is then transcribed and displayed in the `Command` text field. Subsequently, the `Send command` button triggers the command execution. A short screencast showing the tool in action can be seen online[12].

4 NLI Evaluation

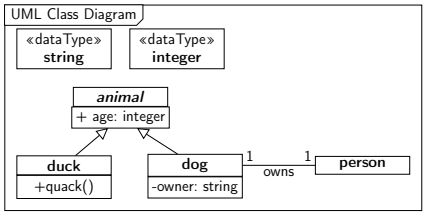
To prove the applicability of NLI-based modeling editor interaction and to empirically evaluate our prototype, we conducted experiments with 23 participants from three different institutes (two universities, one industrial partner). Most participants were technically skilled software engineering students, while some participants did not have a technical background. The user study was conducted in March 2025 and consisted of: 1. a *short questionnaire* about the users' experience with speech assistants (e.g., Siri, Alexa), 2. a *modeling task* to test the NLI, and 3. a *system usability scale* (SUS) [9] questionnaire for the prototype. The survey form we used and the entire questionnaire are available online[13].

4.1 Empirical Evaluation

Our empirical evaluation aims to respond to the following research questions:

RQ1. To what extent are modelers able to create a UML class diagram using NLI?
RQ2. What is the perceived usability of NLI-based UML modeling?
RQ3. How effective is the developed language model in detecting the correct modeling intent expressed in natural language?

[12] `https://github.com/sschwantler/bigUML/blob/main/NLI_README.md`
[13] `https://forms.gle/oe3BEVss5jid3RoF9`



(a) UML diagram in graphical concrete syntax

```
1  Create a new data type called string.
2  Create a new data type called integer.
3  Move it to the right of data type string.
4  Create a new abstract class called animal.
5  Move it to the bottom of data type string.
6  Add a public property called age of type
       integer.
7  Create a new class called Duck.
8  Move it to the bottom of class animal.
9  Add a public method called quack.
10 Create a new class called dog.
11 Move it to the right of class duck.
12 Add a private property called owner of
       type string.
13 Add a new class person.
14 Move it to the right of class dog.
15 Create a generalization from duck to
       animal.
16 Create a generalization from dog to animal
       .
17 Create an association called Owns from
       person to dog.
```

(b) NL commands necessary for creating the UML diagram

Fig. 4: Solution and NL commands for the example used in the evaluation

We prepared a ready-to-use prototype of BIGUML which includes our NLI extension (the UI extensions and the language model), and also with a *tracking component* that logs all interactions between the modeler and the modeling editor (i.e. timestamped audio clips, derived intents, and the actions performed in BIGUML). Using these data, we could systematically evaluate the current prototype and, moreover, learn how humans interact with NLIs. A common modeling use case for all participants ensured comparability amongst the responses. The installation and usage instructions—including a short video that demonstrated how participants could interact with the NLI extension—were distributed as part of the online survey. Furthermore, users were asked to upload the resulting model `.xmi` file, a screenshot of the model, and their usage logs.

Evaluation Use Case The user study was performed to discover how humans *i*) interact with a UML modeling editor in natural language, and *ii*) perceive this interaction. The former was studied using modeling success rates, while *ii*) used a system usability scale evaluation. Most participants were familiar with UML modeling through university classes or practical use in their job, and many of them were already familiar with the BIGUML modeling editor. Each participant was asked to complete the same example task of creating a UML class diagram (cf. Figure 4a) using only the NLI ofBIGUML. The example contains a diverse range of UML modeling elements (abstract class, class, generalization, dataType, property, visibility, role, multiplicity), while remaining purposefully compact. Figure 4b shows a minimal set of speech commands that would create this UML class diagram in BIGUML.

4.2 Results

We report on the evaluation results by responding to the three research questions defined at the outset.

RQ1 – Human Modeler Support. The first and primary concern of our evaluation relates to the usage of the NLI. According to the usage data and the screenshots, 13 of 23 participants (57 %) completed the modeling use case by exclusively interacting in natural language. Nonetheless, we observed that several users struggled with the "move" intent, meaning that the model elements were created on top of another. This was also observable by the Move intent's relatively low recognition accuracy of around 46% (cf. Table 3). The 10 users who could not create the model entirely still managed to create partial models, even though some elements (e.g., individual attributes or methods) were missing. As a result, some participants solved the problem by repositioning the elements using the mouse, while others ignored the move command entirely, such that all elements were placed on top of one another.

Overall, we conclude that some users were able to create models using an NLI-only approach. This is a reaffirming result for our prototype. Nonetheless, we noticed that various problems occurred and that the incorrect functionality of one central intent (in our case, 'Move') causes problems.

Answer to RQ1 A majority of our participants succeeded in modeling using the NLI alone, and many of the remaining participants nearly completed the task with the NLI. We conclude that our tool has the potential to serve as an effective alternative input source to the mouse & keyboard of modeling editors. Nonetheless, we remark that there are still more than 40% of users who could not completely solve the task.

RQ2 – Perceived Usability. Despite the early development stage of our prototypical implementation, we applied the System Usability Scale (SUS) [9] to derive an initial usability score, and thus a quantitative measure of the perceived ease of use, for our system. In the following, we briefly summarize the answers to the SUS questions on a five-point Likert scale ranging from *strongly agree* (++), to *agree* (+), to *neither agree nor disagree* (/), to *disagree* (–), to *strongly disagree* (––). Table 2 shows the frequencies of each specific answer. Aggregating these scores by each participant leaves a mean combined SUS of 66.85 with a standard deviation of 16.14 and a median of 70. Minimum and maximum combined SUS are 30 and 87.5, respectively.

We can notice that the tool has potential, given that most users found the system *easy to use*, and that *the various functions were well integrated.* The vast majority of users also think that *most people would learn to use this system very quickly*, and almost 80% of respondents think that they would not *need the support of a technical person to use the system.* It is also promising that a majority of the respondents reject the statement that the NLI was *very cumbersome to use* and that almost half of them *felt very confident using the system.*

The evaluation, however, also confirmed our expectations that, despite the good intent and entity recognition, the tool's usability can be further improved. Table 2 shows the responses to the SUS. Thus, users responded that they would rather not *like to use the system frequently* in its current state and that it is

Table 2: NLI Prototype – System Usability Scale (SUS)

Question	++	+	/	−	−−
I think that I would like to use the system frequently	0	6	4	10	3
I found the system unnecessarily complex	0	1	3	7	12
I thought the system was easy to use	6	9	5	2	1
I think that I would need the support of a technical person to be able to use this system	0	3	2	8	10
I found the various functions in this system were well integrated	2	16	4	1	0
I thought there was too much inconsistency in this system	8	4	4	7	0
I would imagine that most people would learn to use this system very quickly	6	14	1	1	1
I found the system very cumbersome to use	1	4	5	3	10
I felt very confident using the system	1	8	6	6	2
I needed to learn a lot of things before I could get going with this system	0	3	1	14	5

Table 3: Evaluation Modeling Use Case Log Data

Intent	**Count of Intent**	**Avg. Intent Correct**	**Avg. Entity Correct**
AddAttribute	99	78.8%	65.7%
AddMethod	72	55.6%	69.4%
AddRelation	124	95.2%	72.6%
ChangeDatatype	9	22.2%	22.2%
ChangeName	53	100.0%	88.7%
ChangeVisibility	3	100.0%	100.0%
CreateContainer	231	92.2%	85.3%
Delete	53	94.4%	62.3%
Focus	118	85.6%	76.3%
Move	160	100.0%	45.6%
Undo	10	90.0%	90.0%
Total	**932**	**88.7%**	**70.7%**

acting *inconsistently*. This critical feedback is unsurprising as usability was not a core focus in this initial work on NLI-based UML modeling. Moreover, the inconsistent behavior seems to correlate with participants being unable to move elements using NL commands.

Answer to RQ2 Given the early stage of development of the BIGUML NLI, it was expected that the usability would point out some limitations. Nonetheless, we do see that users found that the system was easy to use, the functionalities well integrated, and that it would be easy to learn for most people.

RQ3 – Intent Detection. Since every participant provided logging data for their modeling session along with all captured voice recordings, we could analyze them for patterns in natural language understanding performance. The logs are structured with timestamps and contain the transcribed query text as interpreted by our speech recognition system. Each query was processed by the NL server to extract two core elements: the intended modeling action (*intent*) and the relevant modeling language elements (*entities*).

In total, we obtained 932 intent recordings as shown in Table 3, which we manually matched to the actual user input (speech and transcripts) to the tool-predicted intents and extracted entities, and classified the correct/incorrect information by intent. We see that, overall, many of the NL commands

Table 4: Examples of Wrong Entity Extraction

Query	Entities
Move animal below data type string	{"element_type": "data type", "element_name": "string"}
Create an owns association between owner and person	{"relation_type": "owns", "element_name": "association", "class_name_from": "owner", "class_name_to": "person"}
Add attribute age integer to animal	{"element_type": "attribute", "element_name": "age integer"}

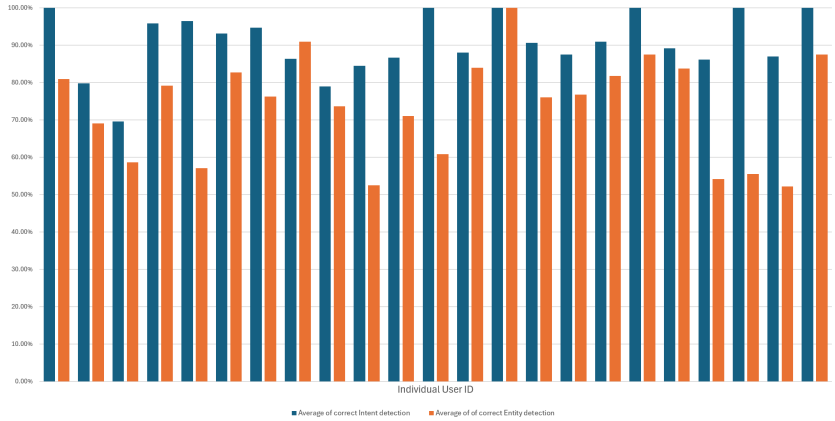


Fig. 5: Average accuracy of intent and entity detection per participant

operated successfully, such that the intent recognition worked very well (except `ChangeDatatype` and `AddMethod`). Still, we also see that the entity recognition is less capable, leading to an overall decreased accuracy in command interpretation.

Table 4 shows some examples of entities wrongly identified. The first query misses an entity containing the direction to move, the second one extracts the wrong entities (relation type and name are wrong), and the third one displays an incorrect attribute name.

In a subsequent analysis, we were interested in whether the accuracy changed between different users. As the use of natural language is subjective (dialects, different ways non-native speakers speak English, mumbling, audio quality, etc..), we computed accuracy scores per individual participant. Fig. 5 presents the user-specific accuracy metrics for all 23 participants. One can see that the results are quite homogeneous for intent detection, with values ranging from 69.57% to 100.00%. For the entities detection, the values are more diverging with a range from 52.17% to 100.00%.

The heterogeneity of these results made us curious to check the submitted original audio snippets of some participants, especially those for which the accuracy values were low. We present these findings, amongst others, in the subsequent discussion section.

Answer to RQ3 Across all 932 user-defined NL queries, our approach could correctly identify 88.7% of modeling intents and 70.7% of entities. We observe that for specific intents and entities, the recognition requires better training data. We thus conclude that our NL server is capable of handling NL input correctly for many user queries in our evaluation case. We can also state that the accuracy changes between participants.

5 Discussion

The development of the NLI prototype provided insights into how users interact with model editors using speech. Here we discuss some of the lessons learned.

Synonymous Sentences. Primary success factors were the two user surveys (see Section 2) through which we discovered a broad set of specific commands that modelers would use to describe a modeling intent. Although we do not claim to have an exhaustive dataset, our prototype evaluation showed that the initial survey covered most commands. Still, our tool also faced an input that had not been seen before (but was still synonymous) that it could not handle. Gathering more usage logs enables us to iteratively improve our tool, especially with respect to intent recognition and entity detection. It also remains an open question whether and how much data should be used to pre-train our two BERT models.

BERT and model sizes. The recently popularized Generative Pre-trained Transformers (GPTs) are a robust and powerful competitor technology. In combination with retrieval-augmented generation (RAG), these technologies can also be customized and typically used with chat prompts.

The BERT models used in our implementation build upon a similarly powerful text processing technique and certainly showed their capability for intent recognition and entity extraction. The technology is also customizable and proven capable of specialized tasks such as programming (CodeBERT[14]) or medical domains (e.g. BiomedVLP CXR BERT[15]) recognition.

As we deliberately used small language models that could be deployed directly on the user's computer, an open question is whether larger (pre-trained) models that run on dedicated servers or as SaaS could be capable of comparable or even better results.

Accents & Dialects. Accents and dialects had a strong influence on voice transcription. Table 5 shows example results of wrong transcriptions with their interpretation of what was intended, along with an evaluation comment when listening to the audio recording. Note that every example is from a different participant. Recurring issues were poor pronunciation and poor microphone quality.

[14] `https://github.com/microsoft/CodeBERT`
[15] `https://huggingface.co/microsoft/BiomedVLP-CXR-BERT-specialized`

Table 5: Examples of Wrong Transcriptions

User	Transcript	Interpretation of Intent	Comment
A	Select class stack	Select class duck	Accent
B	Move it to the left of the glass anymore	Move it to the left of class animal	Accent
C	Add a public method for WAC 2	Add a public method called quack to duck	Accent, Quiet and bad Microphone
D	Rename it to any matter	Rename it to animal	Accent, Reverb. and Echo Effects
E	Place it to the left of class torque	Place it to the left of class dog	Accent

Moreover, the OpenAI Whisper transcription model did consider that the participants were non-native English speakers, which might explain why these errors were observed across different users.

5.1 Limitations

While our NLI shows promising results, several limitations remain.

User Selection Bias. Our user surveys relied on personal invitations primarily comprised of academic participants, with some additions from our industry partner. Their feedback might not be fully representative of the general user population.

Exclusion of Accessibility Considerations. Although we aim to increase accessibility, due to our prototype's early development stage, we could not include users with disabilities in our study. We plan to reach out to these users with the upcoming beta version of our software, as their insights are of utmost interest.

Uncontrolled Evaluation Variables. We did not control for factors such as microphone quality or noisy environments, which impact the quality of the recordings, and, subsequently, the intent recognition and entity detection quality.

Evaluation Realism. It remains unclear how well the NLI performs with larger, more complex UML diagrams requiring advanced navigation and layout support, as our current evaluation was based on a compact UML model (cf. Fig. 4a).

6 Related Work

Our research towards the integration of a capable NLI for UML editors touches upon several related fields. In this section, we position our contributions relative to the most closely related domains.

Natural Language-based Modeling. The use of NL input in modeling workflows has been pursued for several decades. Early approaches used a controlled, structured, or restricted natural language to simplify intent processing [2,16,19,40] for requirements, translation to use cases, or data extraction. Previous approaches use NL input for automated derivation of UML diagrams [4, 12, 13, 20, 27, 30], although these approaches are based on "classical" means of language processing (parse trees, word tagging, phrasal grammars).

Recently, LLMs were shown to be effective for translating natural language into models [10,17], including retrieval-augmented generation (RAG) for domain-specific environments [3]. These approaches, however, lack support for incremental editing and often depend on specific tools. In contrast, our work natively integrates NLI into an existing UML editor. An alternative approach uses chatbots for incremental domain-specific language (DSL) modeling [32, 33].

Speech & Voice Input. Speech interfaces for programming have been studied since at least 2006 [14]. Since then, powerful AI and LLM approaches [31] were applied, recently peaking in so-called *vibe coding*[16], whose goal is the exclusive use of LLMs and NL input to complete small-scale programming projects.

Speech-based modeling has also been explored from several directions. [6] integrates voice for Simulink diagram editing, while [21] uses voice to draw on interactive whiteboards, showing user acceptance of voice-enabled software. ModelByVoice [11] combines voice, non-vocal sounds, and gestures for modeling. This project is close to our work, although the tool primarily supports the matching of short voice commands, rather than the use of an NLI with intent recognition.

Accessibility. Other related work [25] explores NLI for accessibility, e.g., developing a model editor for blind people. The TeDUB project [34] spearheaded the accessibility of technical drawings and graphical models, as outlined in [7]. [22] explores blind users' interaction with UML class and state diagrams, while [5] enables their interaction with graph models.

To summarize, we note that existing NL and speech-based modeling approaches often rely on structured inputs, simple commands, or resource-intensive LLMs, which lack incremental editing and offer limited DSL support. Our work instead integrates a lightweight BERT-based NLI into a UML editor, enabling efficient intent recognition and domain-specific adaptability without the computational overhead of LLMs.

7 Conclusion & Future Work

This paper introduces a natural language interface (NLI) to improve accessibility and usability in enterprise modeling editors. Our approach integrates speech input and state-of-the-art natural language processing (NLP) capabilities into the BIGUML editor, using BERT models for modeling intent recognition and UML entity extraction. The models were trained on a dataset derived from a user survey to capture varied natural language modeling commands. An empirical evaluation with modelers showed that our prototype is capable of fulfilling many modeling tasks, despite the prototype's early-development phase usability problems.

In the future, we will extend our approach using the user feedback to support a more diverse and complex set of modeling actions, including compound multi-intention commands. Finally, we will also reach out to the accessibility community to gain insights and feedback from persons with disabilities.

[16] https://arstechnica.com/ai/2025/03/is-vibe-coding-with-ai-gnarly-or-reckless-maybe-some-of-both/

References

1. Ali, S.J., Reinhartz-Berger, I., Bork, D.: How are llms used for conceptual modeling? an exploratory study on interaction behavior and user perception. In: Conceptual Modeling - 43rd International Conference, ER 2024. pp. 257–275
2. Ambriola, V., Gervasi, V.: Processing natural language requirements. In: Int. Conference Automated Software Engineering. pp. 36–45 (1997)
3. Ardimento, P., Bernardi, M.L., Cimitile, M., Scalera, M.: A rag-based feedback tool to augment uml class diagram learning. In: MODELS Companion (2024)
4. Arora, C., Sabetzadeh, M., Briand, L., Zimmer, F.: Extracting domain models from natural-language requirements: approach and industrial evaluation. In: 19th Int. Conf. on Model Driven Engineering Languages and Systems. pp. 250–260 (2016)
5. Balik, S.P., Mealin, S.P., Stallmann, M.F., Rodman, R.D., Glatz, M.L., Sigler, V.J.: Including blind people in computing through access to graphs. In: 16th Int. ACM SIGACCESS conference on Computers & Accessibility. pp. 91–98 (2014)
6. Black, D., Rapos, E.J., Stephan, M.: Voice-driven modeling: Software modeling using automated speech recognition. In: MODELS Companion. pp. 252–258 (2019)
7. Blenkhorn, P., Evans, D.G.: Using speech and touch to enable blind people to access schematic diagrams. J. Netw. Comput. Appl. **21**(1), 17–29 (1998)
8. Bork, D., Klikovits, S., Michael, J., Netz, L., Rumpe, B.: Inclusive model-driven engineering for accessible software. In: MODELS 2025 NIER (2025)
9. Brooke, J.: SUS-A quick and dirty usability scale. Usability evaluation in industry **189**(194) (1996)
10. Cámara, J., Troya, J., Burgueño, L., Vallecillo, A.: On the assessment of generative ai in modeling tasks: an experience report with chatgpt and uml. Software and Systems Modeling **22**(3), 781–793 (2023)
11. de Carvalho, J.F., Amaral, V.: Towards a modelling workbench with flexible interaction models for model editors operating through voice and gestures. In: 45th Annual Computers, Software, and Applications Conference. pp. 1026–1031 (2021)
12. Deeptimahanti, D.K., Babar, M.A.: An automated tool for generating uml models from natural language requirements. In: ASE 2009. pp. 680–682 (2009)
13. Deeptimahanti, D.K., Sanyal, R.: Semi-automatic generation of uml models from natural language requirements. In: India Software Engineering Conf. (2011)
14. Désilets, A., Fox, D.C., Norton, S.: Voicecode: An innovative speech interface for programming-by-voice. In: CHI'06 extended abstracts. pp. 239–242 (2006)
15. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. In: NAACL-HLT 2019 (2019)
16. Fantechi, A., Gnesi, S., Ristori, G., Carenini, M., Vanocchi, M., Moreschini, P.: Assisting requirement formalization by means of natural language translation. Formal Methods in System Design **4**, 243–263 (1994)
17. Ferrari, A., Abualhaijal, S., Arora, C.: Model generation with llms: From requirements to uml sequence diagrams. In: Req. Eng. Conf. Workshops (2024)
18. Fill, H., Fettke, P., Köpke, J.: Conceptual modeling and large language models: Impressions from first experiments with chatgpt. Enterp. Model. Inf. Syst. Archit. Int. J. Concept. Model. **18**, 3 (2023)
19. Giannakopoulou, D., Pressburger, T., Mavridou, A., Schumann, J.: Generation of formal requirements from structured natural language. In: REFSQ 2020 (2020)
20. Hamza, Z.A., Hammad, M.: Generating uml use case models from software requirements using natural language processing. In: ICMSAO 2019. pp. 1–6 (2019)

21. Jolak, R., Vesin, B., Chaudron, M.R.V.: Using voice commands for UML modelling support on interactive whiteboards: Insights and experiences. In: XX Iberoamerican Conference on Software Engineering. pp. 85–98 (2017)
22. King, A., Blenkhorn, P., Crombie, D., Dijkstra, S., Evans, G., Wood, J.: Presenting uml software engineering diagrams to blind people. In: Computers Helping People with Special Needs: 9th International Conference, ICCHP 2004. pp. 522–529 (2004)
23. Kolev, P., Pruss, H.H., Wilken, J.R., Sandkuhl, K.: Grass-root enterprise modelling: How large language models can help. In: PoEM 2024. pp. 123–139 (2024)
24. Lanusse et al.: Papyrus uml: an open source toolset for mda. In: Europ. Conf. on Model-Driven Architecture Foundations and Applications. pp. 1–4 (2009)
25. Lopes, J., Cambeiro, J., Amaral, V.: Modelbyvoice - towards a general purpose model editor for blind people. In: Proceedings of MODELS 2018 Workshops. CEUR Workshop Proceedings, vol. 2245, pp. 762–769 (2018)
26. Lukyanenko, R., Bork, D., Storey, V.C., Parsons, J., Pastor, O.: Inclusive conceptual modeling: Diversity, equity, involvement, and belonging in conceptual modeling (short paper). In: ER 2023 Forum. CEUR Workshop Proceedings (2023)
27. Mala, G.A., Uma, G.: Automatic construction of object oriented design models [uml diagrams] from natural language requirements specification. In: 9th Pacific Rim Int. Conference on Artificial Intelligence. pp. 1155–1159 (2006)
28. Metin, H., Bork, D.: Introducing BIGUML: A flexible open-source glsp-based web modeling tool for UML. In: MODELS 2023 Companion. pp. 40–44 (2023)
29. Metin, H., Bork, D.: A reference architecture for the development of glsp-based web modeling tools. Software and Systems Modeling (2025)
30. More, P., Phalnikar, R.: Generating uml diagrams from natural language specifications. International Journal of Applied Information Systems **1**(8), 19–23 (2012)
31. Nowogrodzki, A.: Speaking in code: how to program by voice. Nature **559**(7712), 141–143 (2018)
32. Pérez-Soler, S., González-Jiménez, M., Guerra, E., de Lara, J.: Towards conversational syntax for domain-specific languages using chatbots. J. Object Technol. **18**(2), 5:1–21 (2019)
33. Pérez-Soler, S., Guerra, E., de Lara, J.: Flexible modelling using conversational agents. In: MODELS Companion 2019. pp. 478–482 (2019)
34. Petrie et al.: Tedub: A system for presenting and exploring technical drawings for blind people. In: Computers Helping People with Special Needs: ICCHP 2002. pp. 537–539 (2002)
35. Reinhartz-Berger, I., Ali, S.J., Bork, D.: Leveraging llms for domain modeling: The impact of granularity and strategy on quality. In: CAiSE 2025. pp. 3–19 (2025)
36. Sarioglu, A., Metin, H., Bork, D.: Accessibility in conceptual modeling - A systematic literature review, a keyboard-only UML modeling tool, and a research roadmap. Data Knowl. Eng. **158**, 102423 (2025)
37. Sarioğlu, A., Metin, H., Bork, D.: How inclusive is conceptual modeling? a systematic review of literature and tools for disability-aware conceptual modeling. In: Conceptual Modeling: 42nd Int. Conference, ER 2023. p. 65–83 (2023)
38. Seidl, M., Scholz, M., Huemer, C., Kappel, G.: UML @ Classroom: An Introduction to Object-Oriented Modeling (2015)
39. Silva, J., Ma, Q., Cabot, J., Kelsen, P., Proper, H.A.: Application of the tree-of-thoughts framework to llm-enabled domain modeling. In: Conceptual Modeling - 43rd International Conference, ER 2024. pp. 94–111 (2024)
40. Zhang, H., Yue, T., Ali, S., Wu, J., Liu, C.: A restricted natural language based use case modeling methodology for real-time systems. In: 9th Int. Workshop on Modelling in Software Engineering (MiSE). pp. 5–11 (2017)