Voice-Driven Modeling: Software Modeling Using Automated Speech Recognition

Dana Black, Eric J. Rapos, and Matthew Stephan
Department of Computer Science & Software Engineering
Miami University
Oxford, OH, USA
{blackd,rapose,stephamd}@miamioh.edu

***Abstract*—Voice-driven programming allows engineers to alleviate physical discomfort, pain, and injury. It also has the potential to be faster than typing and assist those with disabilities. While there are a number of solutions to voice-driven programming, Model-Driven Engineering (MDE) has yet to exploit this non-conventional but high-potential approach to software development. Standard convention in MDE practice involves creating software models using a traditional mouse and keyboard combination, or whiteboard sketch hardware. In this position paper, we introduce our vision and ideas for a Voice-Driven Modeling (VDM) approach. Our vision involves a framework that includes 3 phases: Speech Processing, Natural Language Processing, and Context Specific Modeling. We describe these 3 phases in this paper, which others can apply in their attempts to realize VDM. We additionally include our research plans for developing a VDM solution targeted at Simulink models and our early proof of concept capable of implementing several example commands. We establish the pertinence of this work through a survey that finds negligible work on VDM and highlights the potential impact this can have on the field of MDE as a whole. Specifically, it is our position that it can have a positive impact on modelers in general, modelers with disabilities, and domain experts not familiar with modeling. It is our hope that this work helps fuel research in this area, allowing for a new way to develop software models.**

***Index Terms*—speech to model, model by voice, voice-driven software engineering, voice-driven modeling, model-driven engineering**

I. Introduction

The way people interact with technology is constantly evolving. For example, people now issue voice commands and requests to robotic assistants daily, be it on their mobile devices or dedicated devices placed around their homes and workplaces. A related growing non-conventional approach to developing software is using speech recognition to develop system program code. For example, VoiceCode [1], Aenea[1], and Vocola[2] allow developers to program using verbal dictation and commands rather than traditional typing. In addition to having the potential of being faster than traditional typing [2]–[4], voice-driven programming may also help alleviate pain caused by physical strain and prevent repetitive stress injuries and other conditions such as carpal tunnel syndrome [4]–[6]. Further, one of the major difficulties in adoption of modeling techniques relates to the understanding of abstraction. By introducing a voice-driven approach we believe a wider variety of users will be able to leverage the benefits of MDE.

[1]https://github.com/dictation-toolbox/aenea
[2]http://vocola.net/

The standard practice in Model-Driven Engineering (MDE) is to develop system models using a conventional keyboard and mouse, or whiteboard sketch recording and interpretation systems [7]. Despite the abstract and collaborative nature of software modeling, and the early success and potential benefits of voice-driven programming, analogous voice-driven modeling techniques have not been explored or employed in any significant way by researchers and practitioners. The nature of MDE alleviates some of the issues caused by keyboard entry through its use of abstraction and graphical modeling. However, MDE-like navigation tasks, such as those modeled by Fitts' Law [8], can still succumb to variance and error that can be avoided by modelers using a voice-driven technique. Such techniques are pertinent as they can potentially benefit modelers in general, modelers with physical disabilities, and domain expert non-modelers.

In this position paper, we present our vision, current work-in-progress, and plans to realize a framework for Voice-Driven Modeling (VDM). We identify background research in programming by voice from which we have and will continue draw inspiration, and our efforts in ascertaining the state of the art and related work in VDM. We describe our vision for a VDM approach, our research plans and current status, and the potential impact of our VDM approach and VDM in general on the field of MDE.

II. Background - Voice-Driven Software Engineering

Voice-driven software engineering is not a new concept. The first speech recognition system was created in the 1950s [9]. Since that time there have been numerous attempts to create voice-driven systems to assist with software engineering by creating program code.

The benefits in speed and accuracy have been well documented by researchers and studies [10], [11]. The average typing speed of a user working with a keyboard on a daily basis is 60-70 words per minute [12]. This is much slower than the average of 107 words per minute that is typically witnessed by users of speech recognition [3]. The error rate among keyboard users is also higher than that of voice input [13]. Verbal input also allows the use of macros and special characters without disrupting the flow of input. In contrast, keyboard and mouse input often require movement of the hands off of the keyboard home row for those same functions, leading to a disruption of speed and accuracy. While MDE helps address the concerns of speed and accuracy by reducing much of the code entry process and replacing it with graphical modeling, there are additional improvements to be gained from a voice-driven approach. Fitts' Law demonstrates the variance and error in pointing devices and tasks [8], such as those necessary in current MDE practice. Using a voice-driven approach is likely to mitigate this variability and error in the modeling process. This is something we aim to demonstrate further through future validation experimentation of our framework and tool implementation.

The use of the mouse and keyboard over long periods of time is associated with repetitive stress injury (RSI) [5]. While some studies have questioned the link between RSI and the use of the mouse and keyboard, those studies have made some fundamental assumptions that are not necessarily correct [14]. The use of the mouse in particular, which MDE modelers use heavily, has been associated with bursitis in the elbow and shoulder, capsulitis, and tendinitis, as well as carpal tunnel syndrome [6], [15].

Some established uses of voice-driven software engineering through creation of program code include NatLink [16], SPEED [17], voicecode.io [1], Vocola, and Aenea. NatLink is a voice-driven python scripting environment created by Joel Gould to extend Dragon Naturally Speaking to work well in a software engineering context [16]. It is a foundational project that is extended by VoiceCode, Vocola, and others. Aenea is a project that extends NatLink to function in a client-server environment [18]. It is commonly used by developers to take verbally generated commands and transfer them across a connection as if they were typed on a keyboard.

Andrew Begel of the Harmonia Research project at the University of California at Berkeley developed SPEED [17]. This system uses a verbal interface to create Spoken Java code, which is semantically similar to the standard Java language and can be compiled by most standard Java compilers.

Motivated by his own repetitive stress injuries, Ben Meyer developed VoiceCode to map specific verbal phrases to text strings, macros, and actions, allowing chaining and nesting of commands. It has pre-defined macros allowing explicit integration with nine different code editors and theoretical integration with any other editor, assuming that the user is willing to create and test the appropriate macros.

Vocola is one of the most popular tools for voice-driven software engineering in use today, and has a similar feature set to SPEED and VoiceCode.

We have drawn inspiration from all the design and benefits of these techniques in the creation of our VDM proposed framework and will continue to do so in our first realization and implementation of it.

III. Related Work - State of the Art

In our survey, we found only two notable studies directly related to voice-driven modeling. The first is a dissertation [19] written in Portuguese with no English translation available. We used machine translation to render an English version for study. According to that translation, they developed a supported approach of a tool to improve accessibility, and consequently, to integrate a requirements engineer with physical limitations in the activity of requirements elicitation. Based on this and other text, the article appears to deal specifically with requirements gathering for MDE by individuals with physical disabilities. This differs from our VDM vision and framework in that what we propose is for software modeling in all phases of the software development life cycle, not just requirement gathering. Further, our approach has multiple aims to improve productivity among a variety of modelers with or without disabilities as well.

The second is a project called ModelByVoice [20]. ModelByVoice is a tool intended specifically to aid the visually impaired in modeling activities. It attempts to replace modeling software tools such as Simulink. ModelByVoice utilizes Google Cloud Speech-to-Text to accept input from users and then uses FreeTTS to output text as audible words. It is intended primarily to remove all graphical and textual requirements from the modeling interface. By contrast, our VDM vision and framework attempts to improve and innovate software modeling for a much broader audience. While we are working with Simulink as a baseline, our intent is to create an approach that is ultimately portable to a variety of tools, rather than replacing the modeling software as ModelByVoice attempts to do. Our work is intended to allow modelers with a wider variety of disabilities than just visual impairments to more easily perform software modeling, including but not limited to modelers who experience difficulty with muscular control or manual dexterity. Further, our intent is that all software modelers will be able to use this tool to increase the speed and ease in which they can perform modeling tasks, as witnessed in the analogous voice-driven programming approaches [10], [11]. Lastly, VDM is intended to make software modeling more accessible to domain experts. We elaborate on our intended audiences and the potential impact VDM can have on them later in this paper.

Further applications of natural language processing (NLP) to modeling exist, however they do not leverage the voice-driven aspects proposed by our VDM framework. Arora et. al. [21] present their approach to using NLP to extract domain models from natural language system requirements. In their work they are able to apply NLP to industrial requirements documents and convert the textual inputs into domain models of the system. Pérez-Soler et. al. [22] apply NLP to incrementally build models using update commands, with a focus on Social Networks for collaborative modeling. Both of these approaches demonstrate the potential for NLP use in modeling domains, and our VDM approach aims to improve upon these successes to provide a generalized voice-driven approach to model construction.

IV. Our Approach to Voice-Driven Modeling

In this section, we describe our VDM vision and framework as we work toward implementing this research. We provide an overview of VDM, details of the framework in its three phases, why we believe our VDM ideas and vision are suitable, potential pitfalls, and our current status and plans.

A. Overview

Our VDM framework consists of three main phases: Speech Processing, NLP, and Context Specific Modeling. We present a high level overview of the framework process in Figure 1. We include a modular construction to allow for different target modeling languages. The first two phases act primarily as preprocessors for software modeling by first transforming the voice to a textual input stream, and then performing preliminary NLP to tokenize the input strings into meaningful words and inputs in the modeling domain. From there, the third phase targets a specific modeling language/tool, and applies meaning to the tokens a VDM implementation has created in phase 2. This allows for the possibility for VDM plugins to incorporate many different modeling tools. As a proof of concept to demonstrate the viability and correctness of VDM, we have chosen to implement the first plugin within the Simulink modeling environment in order to verbally create and manipulate Simulink models. We elaborate on this in the section that follows.

B. Details

In this section, we present lower-level details on the respective phases within the VDM framework. We additionally describe our plans and results in implementing VDM in its early stages.

1) Phase 1: Speech Processing: The first phase of VDM is the initial speech processing of voice commands from the user. This phase primarily involves the reliance on conversion from voice commands to a textual representation. In most cases, we recommend VDM implementers rely on existing work on voice-to-text translation and representations to encode spoken commands into a string of text. This is due to the fact that speech recognition is a well studied area [23], [24], with many well vetted existing solutions. One added benefit of using existing solutions is voice-to-text is a language independent solution, meaning VDM users can speak commands in any number of input languages that are supported by a VDM implementation, requiring minimal additional preprocessing. There are currently two options we are considering in implementing multi-language support: using the speech processing component to translate the input language to an English textual representation in phase 1, or incorporating language support into the NLP phase during phase 2. We present these options in Figure 2 as a supplement to the overview in Figure 1. If we rely on the speech processor in phase 1 to translate the phrases, there is the potential issue of some important semantics being lost in the standard translations. However, this option can leverage the existing strong foundation of language support tools. We discuss the second option further in our description of the second phase, which follows.

In our initial implementation, we will handle this phase through third party commercial software that is already tested and capable of performing the translation from speech to a stream of input text. We based our choice to use existing software on our desire to remove as many possible failure points as possible by relying on commercially developed and tested speech recognition. To that end, we have opted to use Dragon Naturally Speaking by Nuance [3], colloquially known as Dragon Speak, to process the voice inputs and produce streams of input text, which our VDM implementation will then pass to the NLP phase of VDM. It is the top rated speech recognition program; relatively affordable; and used in professional settings include automobile voice command software, medical domains, and more.

2) Phase 2: Natural Language Processing: The next phase within VDM is to provide initial meaning to the spoken inputs. As our goal is to modularize VDM to be useful in different modeling contexts, this NLP phase is not meant to be aware of the specific modeling actions that can be carried out. Rather it is concerned with the general principles of model creation. As such, the NLP phase will remove words that have no direct impact on the modeling context and retain those with semantic implications and intentions. For example, in data flow modeling, the phrase "Add a constant block with the value 10." may become "add constant 10", and "Add a class named Person." may become "add class Person". Further, in order to potentially support numerous input languages spoken by the user, the NLP phase will also include an optional translation step, to translate the text streams obtained in phase one into their corresponding English commands. By including the language translation during this phase of VDM, we are better able to assign the appropriate modeling context to the non-English language rather than relying on an external translation that, while correct, may lose meaning in the MDE domain. However, this requires knowledge of modeling terminology in other languages, as well as the parallel implementation of similar functionality in multiple languages. This becomes imperative when we attempt to extend this approach to non-modelers who do not speak English natively.

In order to implement NLP, we are still in the early phases of exploration, but it is possible we may be able to leverage existing solutions such as Amazon Lex[4], Google Natural Language[5], or Watson Natural Language Understanding[6].

The end result of the NLP phase of VDM is to produce a tokenized input string that contains only the relevant modeling words that can be understood by a specific modeling context. It is probable that this phase will include some normalization of the ordering of the inputs to produce a standard set of input tokens. For example, there should be no difference between

[3]https://www.nuance.com/dragon.html

[4]https://aws.amazon.com/lex/

[5]https://cloud.google.com/natural-language/

[6]https://www.ibm.com/watson/services/natural-language-understanding/

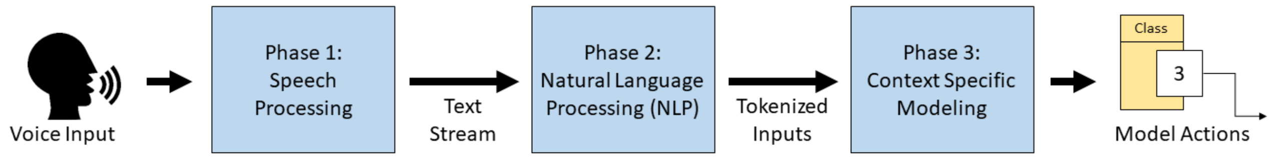
Voice Input
Phase 1: Speech Processing
Text Stream
Phase 2: Natural Language Processing (NLP)
Tokenized Inputs
Phase 3: Context Specific Modeling
Class
3
Model Actions

Fig. 1. Voice Driven-Modeling (VDM) Overview

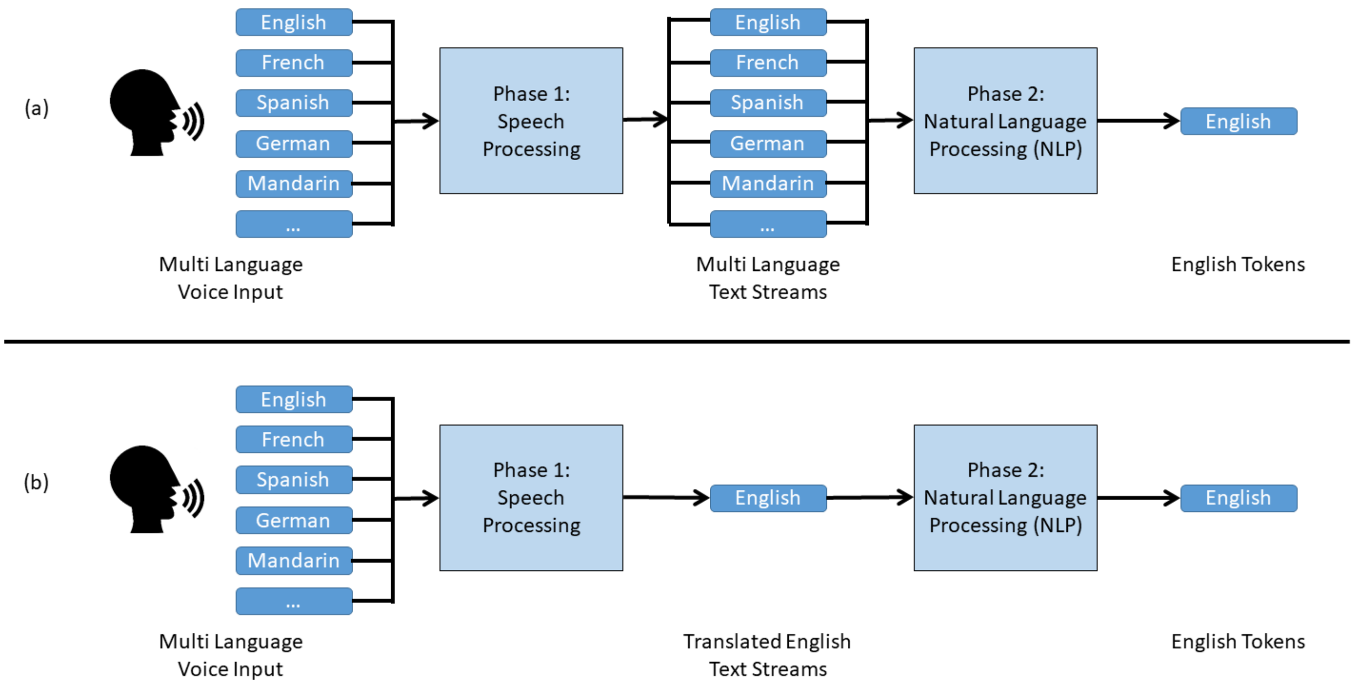
(a)
English
French
Spanish
German
Mandarin
...
Multi Language Voice Input
Phase 1: Speech Processing
English
French
Spanish
German
Mandarin
...
Multi Language Text Streams
Phase 2: Natural Language Processing (NLP)
English
English Tokens
(b)
English
French
Spanish
German
Mandarin
...
Multi Language Voice Input
Phase 1: Speech Processing
English
Translated English Text Streams
Phase 2: Natural Language Processing (NLP)
English
English Tokens

Fig. 2. Two Options for Multi Language Support: (a) language aware NLP phase (b) reliance on speech processing translations

saying “add a constant of 10” and “add a 10 constant value”. Ultimately, the actions taken on a model are generalizable, regardless of the target. The specific objects may change, but modelers are typically interacting with model elements in a specific manner, for example, add, move, connect, change value, et cetera. This generalization will assist in the tokenization and normalization of inputs.

In order to realize NLP in phase 2 of the VDM framework, we first intend to create a dictionary of modeling commands, which will include synonymous words to allow flexibility in the spoken commands. The initial dictionary will focus on modeling concepts specific to our chosen modeling technologies as a proof of concept with the ability to expand to a wider spectrum of modeling phrases and commands. Initially our VDM framework implementation and tool will work on English input only, but eventually with the support of modeling practitioners fluent in other languages, we will look to incorporate a translation sub-phase to expand the usefulness of the VDM framework beyond English-speaking users.

Once we complete the dictionary, the next part of our phase 2 framework realization will be the implementation of a look-up function that will replace the spoken words with the intended modeling concepts from the dictionary. The intent is to leverage existing NLP solutions to remove “filler” words and phrases from the spoken commands and leave only words with semantic meaning in the modeling context. While numerous techniques exist for NLP, our context likely will not merit any heavy processing, as the target language of a relatively small set of modeling words is quite manageable in scale.

After iteratively applying NLP solutions to the input text stream and using the dictionary as a look-up, the result will be an ordered stream of tokens consumable by the third phase to produce modeling actions in a specific context.

3) Phase 3: Context Specific modeling: The final phase of VDM consists of taking the tokenized inputs and providing specific meaning to each of the tokens. This phase requires detailed knowledge of the target modeling language and environment in order to react to the input tokens. Given the normalized structure of the inputs created in the second phase, the context specific modeling phase must be able to rely accurately on properly formatted input, and, if an unexpected token is received, report this as an invalid command.

For each target modeling language, an independent implementation is required. For each modeling language, the implementation will take a different form, such as scripts in the modeling language or applying source transformations to the model file using a source transformation language, for example, TXL [25]. Regardless of the target modeling language, each spoken command must be reflected in real time to the modeler by the VDM implementation so they are able to observe their changes directly in the model on the screen.

With respect to the specifics of phase 3, regardless of implementation, VDM tooling must be able to convert the tokenized inputs into observable actions. For example, if the user speaks the command "Add a Class called Person", a VDM tool must add a new class to the class diagram with that name. However, if the user speaks a command without meaning to the target environment, the tool should not introduce some incorrect action and should inform the user that the command was not implemented.

Because this phase relies on knowledge of the modeling language and its capabilities, each individual implementation will require in-depth knowledge of the target environment. As such we initially begin with one environment as a proof of concept and expand functionality based on other popular modeling tools and technologies. We present our current status in doing so in Section IV-E.

C. Suitability

This section will address why we believe this particular approach is suitable for realizing a VDM framework.

Based on the idea that each voice command issued by a modeler is akin to a statement in a source code program, the process of VDM can be considered "programming a model". With that in mind, we felt it appropriate to model the approach after that of a traditional compiler [26].

The first phase of VDM is the translation from the input stream into an internally understandable format, similar to the lexical analysis and parser phases of compilation. The second phase has a VDM implementation provide semantic meaning to the inputs, which is similar to the semantic analysis phase of compilation. Finally, the context specific modeling phase is akin to the target code generation phase of a compiler, in which the intermediate representations produce output in a target language. These similarities provide justification for the 3 phases we presented, as well as the suitability of our chosen pipeline.

D. Potential Pitfalls

While we are confident in our plans to realize our VDM implementation and tool, it is important to acknowledge potential pitfalls and hurdles we may encounter. An obvious hurdle we will have to overcome is having established modelers become comfortable using voice-driven techniques. Allowing them to perform VDM in conjunction with traditional modeling simultaneously may help with this. Additionally, in our Simulink implementation, we can have a "toggle" option to turn off the VDM option and also a facility to submit feedback about the accuracy of the voice interpretation and subsequent model operations. Another potential threat involves layout and aesthetic considerations. Unlike voice-to-text software programming, an important consideration is how the models actually appear, visually. To address this at first, we propose the use of best-practices making use of a grid-style layout. Essentially our implementation will place any new blocks surrounding the existing blocks, expanding from the top left outwards, since typical Simulink models follow this flow. This, along with the built-in ability to reposition blocks using voice commands, as we demonstrated earlier should mitigate this potential pitfall.

E. Current Status and Plans

In order to demonstrate the validity, viability, and potential of our VDM vision and framework, we are currently developing a VDM tool implementation intended for Simulink modeling. We chose Simulink due to a number of factors. These include its popularity, applicability, and support. Simulink is very popular in both university/academic and industrial settings, making it an excellent choice for a first target language as it will be easier to find a user base for validation experiments. Beyond this, targeting a more "popular" language allows us to observe usage in a wider variety of applications. This leads perfectly into the second factor of our choice: applicability. Since Simulink is used in various industries to create a wide variety of models and software systems, our choice of Simulink will help demonstrate the context independence of VDM. Further, since there are a wider number of domains that leverage Simulink, we will be better able to demonstrate that VDM will alleviate the need for modeling specific knowledge and relies primarily on the domain knowledge of the user. Finally, the fact that Simulink is a commercially supported tool makes it an ideal candidate for the first implementation of the VDM framework. Rather than facing struggles of open-source tools, we are able to focus on intent rather than implementation issues. It is for these reasons that we have selected Simulink for our initial proof of concept implementation of the VDM framework.

For the first phase of our VDM proof of concept implementation, we have begun experimenting and familiarizing ourselves with Dragon Speak and its speech recognition capabilities. Specifically, we acquired Dragon Speak software and purchased a high quality headset, the BlueParrott C400-XT Noise Canceling Bluetooth Headset by Jabra, as it was very well reviewed, has demonstrated success with Dragon Speech Recognition, and was not prohibitively expensive.

We have not yet begun working on our VDM implementation of Phase 2, as the NLP relies on the text strings as an input and we are still experimenting with Dragon Speak

TABLE I
EXAMPLE TOKENIZED INPUTS FOR A SIMULINK MODEL

1	"add"	"inport"	
2	"add"	"constant"	"10"
3	"move"	"constant"	"down"
4	"add"	"sum"	
5	"move"	"sum"	"right"
6	"connect"	"inport"	"sum"
7	"connect"	"constant"	"sum"
8	"add"	"outport"	
9	"move"	"outport"	"right"
10	"move"	"outport"	"right"
11	"connect"	"sum"	"outport"
12	"save"		

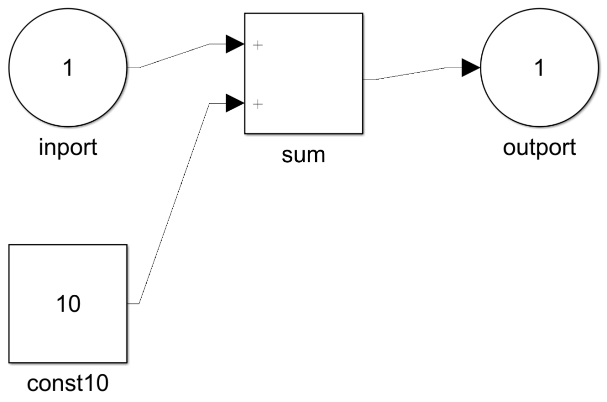

Fig. 3. Example Simulink Model produced with commands from Table I

and its abilities. As an early proof of concept for Phase 3, we have created Simulink scripts to process simulated input, as if tokenized inputs were being received from a completed phase 2. Our Simulink scripts process each line of tokens one input at a time and then make decisions based on the inputs. The structured input provides the action first, for example, add, move, connect, or save. This is followed by the artifact(s) upon which the actions are transpired.

Table I provides 12 example input commands that our Simulink scripts are currently able to interpret and act upon for a new blank model. Each row represents one spoken command, totalling 12 commands.

By iterating over each of the commands found in Table I, our early phase 3 implementation automatically produces and saves the Simulink model we present in Figure 3.

A final area that we have yet to begin implementing is foreign language support. We have identified two possible options, each with the advantages and disadvantages we listed earlier. We will need to chose one option to realize this functionality.

Currently, our working prototype implements Phase 3 of the VDM framework on a limited set of input streams, passed as textual input to a Matlab script. This initial implementation provides promising results for a first-pass prototype.

V. POTENTIAL IMPACT

While voice-driven programming techniques exist to leverage the power of NLP to produce functional source code, there does not currently exist a reliable technique to apply this to model-driven engineering. Our VDM framework proposes leveraging the demonstrated success of voice-driven approaches in the MDE domain.

Our VDM approach will be helpful to current modelers, enabling them to streamline the modeling process and potentially increase the speed and efficiency at which they can model systems. By removing the need to interface with modeling tools through typical peripheral devices, a VDM approach and tooling support will allow users to interact with other artifacts such as requirements documentation or specifications. In Agile development environments, a voice-driven approach to modeling fits well with a pair programming or rapid prototyping approach where a team of engineers could theoretically all interact with the same model on the same machine using only voice commands.

Another aim and impact we want is to broaden the user group of those employing software modeling approaches. Currently, the use of MDE techniques requires a non-trivial amount of knowledge of MDE and its applications. However, through VDM, much of that required knowledge is abstracted leaving only the requisite domain knowledge and minimal tool understanding as entry points. By opening the use of modeling tools up to a wider audience, we are enabling non-technical users to become software developers of their own model-driven systems. Since the NLP phase will include modeling-intuitive processing, users of the VDM framework do not need to be overly familiar with modeling terminology or practices. The user is able to describe their intent in domain-familiar terminology, and the VDM framework will apply MDE contexts to their commands, when possible.

Finally, another potential impact of our VDM approach is the ability for those with disabilities to perform modeling tasks. Whether this includes limited mobility, the inability to use standard peripherals, or some form of vision impairment, our VDM approach allows users that may otherwise be hindered/unable to create models and leverage the benefits of MDE approaches. While we have no data for such claims yet other than existing analogous work in voice-driven programming [4], our long term plans include doing user studies on those with disabilities to better measure this impact.

We thus summarize the potential impact of the VDM framework by highlighting the three target user groups:

- **software modelers** - technical users who would utilize the technology to streamline the modeling process and improve modeling efficiency
- **domain-expert non-modelers** - domain experts who are not familiar with modeling, but are able to accurately describe algorithms they wish to see modeled and implemented
- **modelers with disabilities** - the technology will also be assistive for those with mobility and other accessibility

issues to create models proficiently

VI. Future Directions

Through the realization of VDM, it is likely that we will discover aspects of modeling that are not well suited for voice-driven approaches. This is due current modeling tools not being designed with voice commands in mind. Through our implementation and validation of the VDM framework, there is the potential that we may be able to inform modeling language design decisions to better serve voice-driven approaches while still maintaining the integrity of modeling fundamentals.

Beyond the application to general purpose modeling language design, a further direction is the ability to inform the design of domain-specific modeling through the creation of languages that are geared towards voice-driven approaches. This line of future work may yield a new field of language design for domain-specific voice modeling.

Further, if VDM proves to be an efficient and effective method of modeling, there is the potential for integration into existing modeling tools, such as Simulink and the Eclipse Modeling Framework, rather than existing as a standalone tool sitting on top of other technologies.

VII. Conclusion

Voice-driven technology is advancing and being used in increasingly innovative ways, including software code creation. MDE has yet to fully exploit voice-driven technologies and its potential benefits as witnessed in voice-driven source code creation including efficiency, accessibility for a variety of disabilities, and the potential to alleviate some physical ailments. To that end, we have presented our position on and vision for voice-driven modeling and our ideas of a VDM framework with the goals of creating a proof-of-concept tool and igniting research in this area. We describe our VDM framework through its 3 phases, and support its suitability through the analogy of a traditional compiler. Our immediate goal is to create an end-to-end solution for Simulink modeling through verbal input in English, for which we outlined our current research in its early stages and our future plans. We contend that VDM and its associated research has the potential to benefit software modelers in general, modelers with disabilities, and non-modelers who are domain experts. Further, the work on VDM has potential to influence the design and evolution of future modeling language design to better support voice-driven approaches. We believe this work to be the first significant step towards making VDM a reality, and look forward to discussing our new ideas and visions on VDM with those at the workshop in order to elicit feedback, suggestions, and concerns.

References

[1] A. Désilets, D. C. Fox, and S. Norton, "Voicecode: An innovative speech interface for programming-by-voice," in *CHI'06 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2006, pp. 239–242.

[2] M. English, "The efficiency of programming through automated speech recognition," Master's thesis, Haverford College. Department of Computer Science, 2015, http://hdl.handle.net/10066/17627.

[3] C.-M. Karat, C. Halverson, D. Horn, and J. Karat, "Patterns of entry and correction in large vocabulary continuous speech recognition systems," in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. ACM, 1999, pp. 568–575.

[4] A. Nowogrodzki, "Speaking in code: how to program by voice," *Nature*, vol. 559, no. 7712, pp. 141–142, 2018.

[5] D. Sharan, P. Parijat, A. P. Sasidharan, R. Ranganathan, M. Mohandoss, and J. Jose, "Workstyle risk factors for work related musculoskeletal symptoms among computer professionals in india," *Journal of occupational rehabilitation*, vol. 21, no. 4, pp. 520–525, 2011.

[6] A. Aarås, M. Dainoff, O. Ro, and M. Thoresen, "Can a more neutral position of the forearm when operating a computer mouse reduce the pain level for visual display unit operators? a prospective epidemiological intervention study: part ii," *International Journal of Human-Computer Interaction*, vol. 13, no. 1, pp. 13–40, 2001.

[7] J. Whittle, J. Hutchinson, and M. Rouncefield, "The state of practice in model-driven engineering," *IEEE software*, vol. 31, no. 3, pp. 79–85, 2014.

[8] I. S. MacKenzie, "Fitts' law as a research and design tool in human-computer interaction," *Human-computer interaction*, vol. 7, no. 1, pp. 91–139, 1992.

[9] M. Pinola, "Speech recognition through the decades: How we ended up with siri," *Web log post. TechHive. IDGTechNetwork*, vol. 2, 2011.

[10] K. Jahandarie, *Spoken and written discourse: A multi-disciplinary perspective*. Greenwood Publishing Group, 1999, no. 1.

[11] J.-F. NUNAMAKAR, A. Dennis, J. Valacich, D. Vogel, and J. George, "Electronic meeting systems to support group work," *Communications of the ACM*, vol. 34, no. 7, pp. 40–61, 1991.

[12] R. Bailey, "Human interaction speeds," *Disponibile in rete allindirizzo http://webusability. com/article_human_interaction_speeds_9_2000. htm (ultimo accesso 18.12. 2006)*, 2000.

[13] J. Leggett and G. Williams, "An empirical investigation of voice as an input modality for computer programming," *International Journal of Man-Machine Studies*, vol. 21, no. 6, pp. 493–520, 1984.

[14] B. Rhode and W. Rhode, "Occupational risk factors for carpal tunnel syndrome," *MOJ Orthop Rheumatol*, vol. 4, no. 2, p. 00131, 2016.

[15] C. Jensen, L. Finsen, K. Søgaard, and H. Christensen, "Musculoskeletal symptoms and duration of computer and mouse use," *International journal of industrial ergonomics*, vol. 30, no. 4-5, pp. 265–275, 2002.

[16] J. Gould, "Implementation and Acceptance of NatLink, a Python-Based Macro System for Dragon NaturallySpeaking," in *The Ninth International Python Conference*, 2001, pp. 5–8.

[17] A. Begel, "Spoken language support for software development," in *2004 IEEE Symposium on Visual Languages-Human Centric Computing*. IEEE, 2004, pp. 271–272.

[18] Dictation-Toolbox, "dictation-toolbox/aenea," Mar 2019. [Online]. Available: https://github.com/dictation-toolbox/aenea

[19] F. J. T. Soares, "Uma abordagem para derivar modelos de requisitos a partir de mecanismos de reconhecimento de voz," Ph.D. dissertation, Universidade Nova de Lisboa, 2014.

[20] J. Lopes, J. Cambeiro, and V. Amaral, "Modelbyvoice-towards a general purpose model editor for blind people," in *Third International Workshop on Human Factors in Modeling (HuFaMo 2018). CEUR-WS*, 2018, pp. 35–42.

[21] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, "Extracting domain models from natural-language requirements: Approach and industrial evaluation," in *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS '16. New York, NY, USA: ACM, 2016, pp. 250–260.

[22] S. Pérez-Soler, E. Guerra, J. de Lara, and F. Jurado, "The rise of the (modelling) bots: Towards assisted modelling via social networks," in *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2017. Piscataway, NJ, USA: IEEE Press, 2017, pp. 723–728.

[23] D. Bijl and H. Hyde-Thomson, "Speech to text conversion," Jan. 9 2001, uS Patent 6,173,259.

[24] P. Khilari and V. Bhope, "A review on speech to text conversion methods," *International Journal of Advanced Research in Computer Engineering & Technology*, vol. 4, no. 7, 2015.

[25] J. R. Cordy, "The txl source transformation language," *Science of Computer Programming*, vol. 61, no. 3, pp. 190–210, 2006.

[26] S. Muchnick *et al.*, *Advanced compiler design implementation*. Morgan Kaufmann, 1997.