ModelByVoice - towards a general purpose model editor for blind people

João Lopes
DI FCT
Universidade NOVA de Lisboa
Lisboa, Portugal
jr.lopes@campus.fct.unl.pt

João Cambeiro
DI FCT
Universidade NOVA de Lisboa
Lisboa, Portugal
jmc12976@campus.fct.unl.pt

Vasco Amaral
NOVA LINCS, DI FCT
Universidade NOVA de Lisboa
Lisboa, Portugal
vma@fct.unl.pt

***Abstract*—Context: Current modelling technologies, with the support of modelling frameworks, are in the base of the current adoption of Model-Driven Software development - MDD - and supporting Software Engineering phases.**

Problem: The focus of these tools are solely on graphical support and visual models. In fact, the chosen modelling language's concrete syntax is either graphical or textual or both. This approach is discarding the use of other senses for modelling purposes, and, for instance, the possibility of blind software engineers to take advantage of modelling and deal with the abstractions captured by those. It is necessary to improve the productivity of people with limitations or disabilities while modelling.They should not be excluded from the modelling activity. This situation of accessibility barriers starts already at education of Modelling.

Method: In this paper we present a prototype of a tool that aims to take advantage of current voice recognition and speech synthesis to edit models in diverse modelling languages. The elegance of this work is the fact that, not only it is meant to make MDD accessible to a broader spectrum of practitioners, but also it is developed with an MDD approach.

Results: A prototype was built, named ModelByVoice. This tool is not bound to a particular modelling language, as long as it is meta-modelled. ModelByVoice is the base for a new tool that will enable MDD highlighting the relevant human factor of accessibility via voice and audio to models. Ultimately, it aims at bringing accessibility for blind people to deal with MDD and Domain Specific (Modelling) Languages - DS(M)Ls - the same way it is already done with diagrammatic languages with the current Modelling workbenches.

***Index Terms*—Model-Driven Software Development, Modelling Workbenches, Accessibility, Speech Generation and Synthesis, Audio Models**

I. Introduction

A research study carried out by IBM in 2004 [1] emphasises the idea that modelling software is, and will continue to be, in the light of many engineers' perspective, the foundation for dealing with the complexity of systems. The key is Abstraction. Thus, the modelling activity proves to be fundamental, since it allows not only a better understanding and clarification of what one intends to develop but also to define a plan with the requisites and necessary functionalities for the system to be implemented. Ultimately, the abstractions captured by the models are represented thanks to visual (mostly diagrammatic) languages. Both general purpose and Domain Specific (Modelling) Languages -DS(M)Ls- have a visual nature and assume that in general, a software engineer will not initially have physical limitations that prevent him from handling with computers. However, if we consider visually impaired people who need to use these modelling languages, it is very complicated to use those languages due to the lack of adapted software to these limitations. Typically, language developers delegate the responsibility of supporting disabilities to third-party general purpose software, which focuses on textual reading, which may not be sufficient because the integration with the modelling platforms can be cumbersome, not bringing the required productivity in the modelling effort.

According to Stack Overflows survey in 2018 [2] of over 100,000 software developers that participated in this study, 1.4 % are blind or have difficulty seeing. The small percentage of these professionals makes it economically non-viable to build dedicated software, which justifies the lack of products not only in the daily life but also to support for those software engineers in their professional activity. Software modelling, as one of the phases in the software development life-cycle, is perhaps one of the most affected by this lack of support.

The diagrammatic languages modelling workbenches such as Eclipse GMF / EMF [3], [4], DSL Tools [5], AToMPM [6], GME [7], MetaEdit+ [8], or their textual counterparts such as Xtext [9] for Eclipse or Meta-Programming System(MPS) [10], do not provide the support for blind people to model diagrams through resources such as their voice or touch. The concrete syntax of these languages focuses only on visual aspects, that is, they require a visual analysis, ignoring other senses like the sound and touch, which makes modelling activity extremely difficult for the blind people or visually impaired. We argue that Audio/Voice is yet another aspect of Human interaction in modelling that should be properly supported in modelling. Therefore, we propose to create a tool to use voice synthesis and recognition in a more structured way than just the one-dimensional approach of mapping text syntax solutions (one-to-one) instead of using the natural 2D structure of the models (mostly graphs) abstract syntax. This approach should support easier manipulation of models for the end-user. It must support edition operations, equivalent to the ones already found in the textual/graphical editors, like CRUD's create, update, delete, and others like select, navigate or query. The MDD (Model-Driven Development)

approach was the technique adopted for the creation of the platform since this concept promotes the systematic reuse of components. Beydeda et.al [11] describe the MDD technique by saying that it is based on models that are considered the main elements of the development process. These models aid in thinking about the problem domain and design a resolution in the solution domain, providing abstractions of a physical system that allows engineers to focus on critical aspects of the same system.

In this paper, we will present our tool prototype, named ModelByVoice, that allows the visually impaired people to model diagrams or systems with any modelling language (non-hardwired to a specific language), through their voice thanks to a speech recognition system implemented on the tool. The tool was created under the assumption that the supported domain-specific languages (DSL) are meta-modelled, as the current focus of the approach is on the language's abstract syntax instead of concrete syntax.

The rest of this paper is organised as follows. In section II we discuss the current efforts that to the best of our knowledge contribute to tackling the problem towards supporting modelling in software engineering for blind people. In section III we present an overview of our prototype. In section IV we discuss a preliminary assessment of our prototype tool. Finally, in section V we conclude and discuss future challenges to address.

II. Sate-of-the-art

The current state-of-the-art does not provide tools that make it possible for the visually impaired to model diagrams satisfactorily. Some tools were developed to circumvent this problem, but in the most of cases, the success rate was reduced. As an example, we have the Technical Diagram Understanding for the Blind, as known as TeDUB [12]. This project was developed to provide a UML modelling tool accessible to visually impaired individuals. Mainly a visualisation tool, the users created and explored the diagrams through a joystick or a keyboard [13], which means that the creators of this tool adopted a haptic communication approach for the domain users. The models should be persisted in XMI format so that it could work with models exported from tools like Rational Rose or Poseidon UML. As the tool was a model navigator and not a proper model editor, the project did not have the expected success due to the poor adherence of the domain users, and it frequently failed when had to handle a significant amount of data. The project stopped in 2005 with a claim from the authors that this sort of solutions [14] would have more success if they were merely transformation tools to export the models into HTML code, maintaining the links and use of plain text, as it would be more inter-operable with current screen-readers that already read web pages.

Another approach to this problem, named PRISCA [15], tried to circumvent this obstacle through 3D printing. The users through their touch, once confronted with the diagrams in 3D, tried to interpret them, but the solution proved to be expensive, slow and ineffective. Thus, given the reasons stated above, the ModelByVoice developing process was very challenging, motivating and innovating task, since the search for new concepts and paradigms is almost always associated with the construction of a more adjusted reality and following the demands of today's world.

According to the documentary research carried out, the tool that most closely resembles the features and resources of the ModelByVoice is the VoiceToModel tool [16]. This tool was designed to enable the visually impaired requirements engineers to derive KAOS models, conceptual models, and feature models. It uses mechanisms of speech recognition and synthesis. The user, through his voice, enunciates the execution commands associated with the platform to create the type of model that he wants. Subsequently, an audio response is given to the user, which serves as feedback associated with the voice commands. Its great limitation compared to ModelByVoice, is that it only allows the creation and editing of diagrams that are composed for one language of the set of languages described above, which means that, is limited to three languages, while with ModelByVoice, theoretically, we can use any desired language as long as it is metamodel is defined.

III. Solution overview

A. Architecture

At the architecture level, the ModelByVoice platform is implemented in three layers. The first layer is related to the voice recognition process. The platform is continuously capturing the sound generated by the surrounding environment. When a speech utterance is detected, the sound is converted to text, and the result is delivered to the second layer. The second layer then tries to match the received user command to an operation defined by the modelling language. If a valid operation is detected, this layer is responsible for the execution of the operation. The feedback mechanism is implemented in the third layer, and the platform uses speech synthesis to communicate the input commands results back to the users. In situations where an invalid command is detected, the system alerts the user, and it prompts the user to a new attempt.

B. Technology

To create the necessary basis to start the developing process of the domain specific language, we used the Epsilon [17], which is an inter-operable and consistent set of languages for model-driven engineering. This toolset is grounded on the Eclipse Modelling Framework, or EMF [3] which is an open source code generation framework in Eclipse based on a structured data model called Ecore. The representation of the models in EMF is achieved by making use of the Ecore meta-modeling language (a "de facto" implementation of MOF) [3]. Concerning the usefulness of this tool for the development of ModelByVoice, the EMF proved to be fundamental as we precisely define the metamodel of the modelling language with Ecore. ModelByVoice makes use of the EMF with the purpose of reading any language as long as it has its metamodel defined with Ecore. This is a key functionality and highlight that we

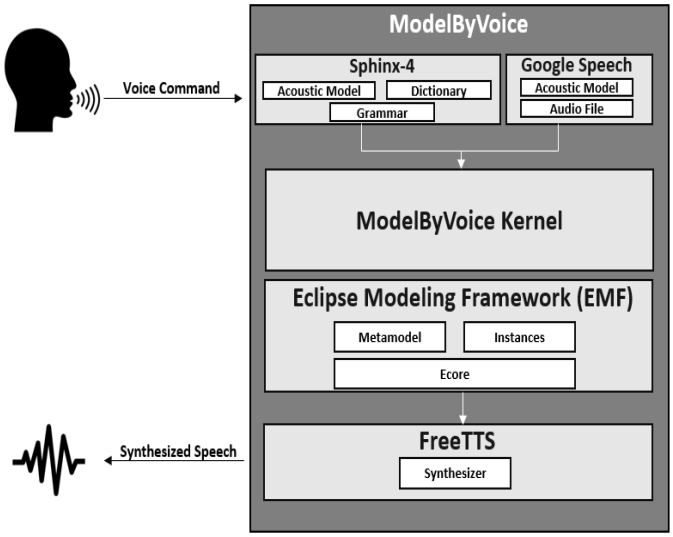

Fig. 1. ModelByVoice Architecture

make use in our platform, being an advantage when compared to other approaches.

We used the Epsilon Transformation Language (ETL) [18] and Epsilon Generation Language (EGL) [19]. These languages belong to Epsilon. ETL was used to associate (compose), via model transformatiom, which elements of the entry metamodel (metamodel of the language with which one intends to model) are nodes, links and / or compartments (as explained in subsection III-C), elements that will compose the final metamodel. For example, if we want to model using the state machine language (formed by states and transitions), we have to associate by equivalence, the notions of state to a node, and transition to a link. Each time we want to have a modelling environment for a particular language, it will be necessary to generate the platform code with EGL from its metamodel(result of composing the previously referred elements). In this case, this tool was used mainly to embed the types of modelling language elements that are to be modelled, in which the data structure chosen to store this information were two lists, one for the nodes and another for the links, thus generating the program code adapted to that modelling language, and which restricts the user to the language domain.

As said before, the ModelByVoice platform is composed of speech recognition and synthesis system. The technological tools used for the speech recognition system were the Sphinx-4 [20] together with the Google Cloud Speech-to-Text. The first one, Sphinx-4 , is used for the generic application commands such as create, remove and save commands, for example, while the Google Cloud Speech-to-Text is used to recognise the variables of the elements that will compose the diagram, such as the name and the type of objects that user can create or edit. The interaction between the user and ModelByVoice is established in the English language, because it is a universal language and may cover a more significant number of potential users around the world. Is it a standard procedure among programmers that the names of variables and classes result from the composition of more than one common word. For example, getListOfClients could be used to name a function. Google Cloud Speech-to-Text only supports words that are present in the English dictionary, and as such we restrict the set of variables names to words that belong to the English dictionary.

The FreeTTS tool [21] is used to implement the speech synthesis component. FreeTTS is an open source tool that converts text to audio, by voice synthesis. This system was implemented with the purpose of giving feedback about the execution of the operations and the actual state to the user during the modelling of the systems. The MBROLA voice system [22] was also integrated into this tool. This project contains several synthesised voices, in different languages and genres. It was decided to use the English male voice, and in this system, voices are less robotic than the standard voice of FreeTTS, getting closer to the recording of the human voice.

C. Functionality

An intended key aspect of this platform is that it gives the possibility to the user to model with any modelling language as long as its metamodel exists. The platform, through the reading of the meta-model of that language, allows making the association of the elements of this language, for the standard notions under described.

At the operation level, it is possible for the user to model diagrams based on the notions of node, link, compartment and attribute. A node represents a particular entity or element, a link allows to make the connection between nodes and / or compartments, a compartment represents a hierarchical set of nodes, that is, it is a node composed of other nodes, and, finally, an attribute can be associated with one or more nodes.

In the figure 2 we have represented the standard language metamodel, which covers most elements of the languages, by association, and that for most input languages, it will be the output metamodel after the conversion of the respective metamodel original input. As explained before, we have to convert the metamodel of the language that we want to model, by associating the composing elements to the notions represented on the following metamodel, which is in the most of cases, the final metamodel.

The ModelByVoice functionality is based on operations that are performed from associated voice commands. It is required that the edited languages support voice commands for CRUD (create, read, update, delete) operations on the diagram and its elements, giving the user the ability to create, listen (”read”), edit and remove the variables that compose the elements of each model. In the following figure 3, we have the supported commands of the platform.

The primary challenge found in the implementation process was the way in which the user would navigate the diagrams and perform the desired operations. The notion of ”navigation element” was used to overcome this problem. This element, which may be a node or a compartment (which is also a node), acts as a reference and guidance point for the user to explore

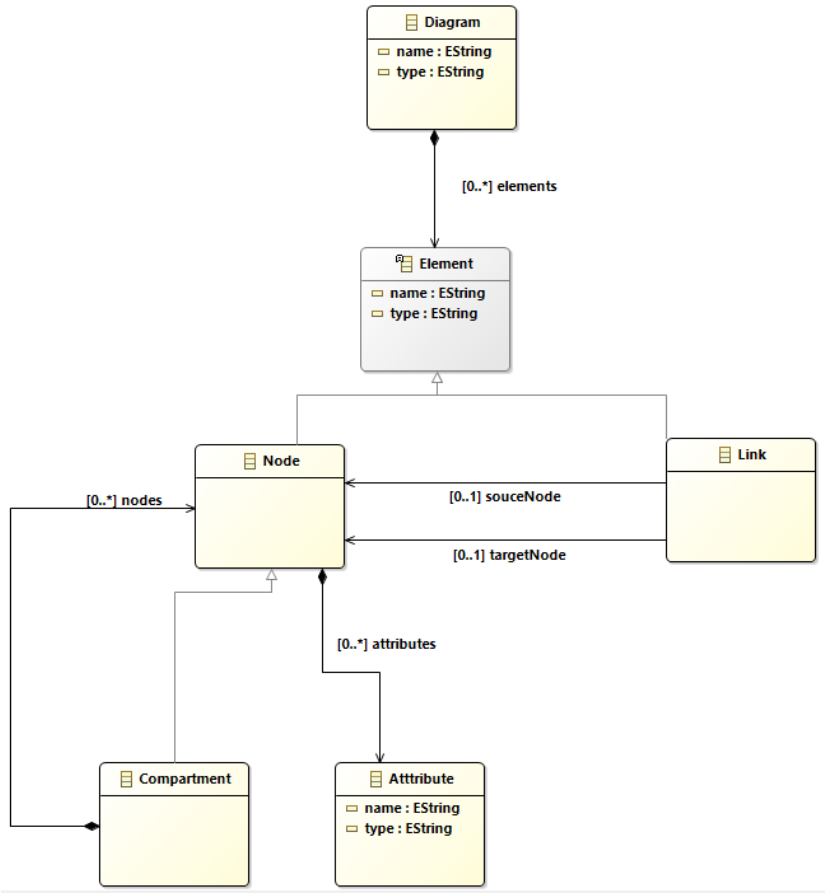

Fig. 2. Standard Meta Model

```
#JSGF V1.0;

/**
 * JSGF Grammar
 */

grammar ModelByVoice;

public <diagram_ops> = create (diagram | node | link | compartment);
            list diagram;
            open diagram;
            save diagram;
            undo;
            sleep;
            help;
            exit program;

public <navelm_ops> = remove (navigation element | attribute | link);
           create attribute;
           update attribute;
           update navigation element;
           change navigation element;
           detail navigation element;
           say neighbours;

public <number> = (one| two | three | four | five | six | seven | eight | nine | ten | <NULL>);

public <bool> = (yes | no | <NULL>);
```

Fig. 3. Sphinx Commands Grammar

the diagrams. The election of the navigation element is based on three possibilities: i) when a new node/compartment is created, it is assigned the role of navigation element; ii) the user by enunciating the command ”change navigation element” can designate which will be the new current navigation element through the enunciation of its name by a voice command; iii) when the navigation element is deleted, and the user loads a previously created diagram for edition, the oldest element in that diagram is called the navigation element. Therefore, this was the idealised and chosen form, to avoid that the user gets lost during the creation, exploration, and edition of the diagrams.

The diagrams are presented to the user trough the list diagram command. This command allows to list all the diagram content trough voice synthesis, or part of it, where the user have to state the range number of elements to list by the platform.

Another essential implemented function was the help mode command. This mode allows users to be guided during navigation, calculating the possible commands that the user can enunciate from the state in which the diagram is.

Each time the user announces a voice command, the operation associated with this command will be executed by the platform, and then the voice synthesis will issue a response, so that, the user will know about the success of the operation execution. The provision of feedback during the execution of the platform allows the user not to become lost during the execution of the operations.

Once the user executes all the desired operations, there is the necessity to call the save diagram command. This command will save the diagram in XMI (XML Metadata Interchange)

format, in the directory of the machine where the platform is located.

In the figure 4 we have represented through an activity diagram, the possible executions of the ModelByVoice and its interaction with the user. All the possible operations are represented in this activity diagram, as well as all the options that the user can take during the process of the platform.

In turn, in the figure 5 we have the internal representation of the help mode available to the user. This helpful resource is intended to calculate the possible operations to perform from a given element, and to inform the user about which commands can be executed. Figures 6 and 7 are session logs to exemplify the models creation and edition by the operator while using our tool prototype.

IV. Preliminary assessment

With the collaboration of the Portuguese Association for Visual Impairment, including blindness and amblyopia, ACAPO - Associao dos Cegos e Amblopes de Portugal, two blind users were involved in a preliminary assessment session where they had the chance to use the prototype and answer to a questionnaire.

The profile of the subjects to perform the usability tests of the tool was divided into two categories, blind or visually impaired users, and users without any visual limitation. Relatively to the first profile, one subject was MSc. in Software Engineering with previous experience with modelling using Graphviz [23] and TeDUB with keyboard and joystick. The second subject did not have a formal education in Computer Science and Engineering. The second group of users, without visual limitations, were three MSc students of Software Engineering.

The selection process of the participants began by defining a set of prerequisites, which the users would have to satisfy. The requirements outlined were as follows: the user must present a basic knowledge of the English language; there must be total unfamiliarity about the platform; and finally, the need for basic theoretical-practical notions in the computing area.

Before the experiment, it was established that the first task would be to conduct interviews with the users. The interviews objectives were the verification if the subjects effectively fulfilled the prerequisites defined, and collect personal information such as age, experience in the area and if they had previous experience with modelling tools that have integrated speech recognition and/or synthesising mechanisms. Regarding usability tests, two tasks were outlined, both similar, but for two different languages. The first usability test task involved modelling with the state machine language (states and transitions) because it is a simple language and involves few concepts. The problem proposed was to create a state machine that represented all the possible states and transitions of a traffic light. First, the subjects had to produce a diagram named Traffic Light. They had to create three state-type nodes, and assign the names red, yellow, and green. Subsequently, they generated three transitions, one from the red state to the yellow state, another from the yellow to the green state, and finally from the green to the red state. After that, the subjects had to list the diagram to verify that all elements were created correctly. If all parts were created with success, they would be asked to create an attribute with the name seconds, with integer type, to represent the time that each of the semaphores would be active. In the figure 8 we have a diagrammatic representation of the process for the first task performed by the subjects.

It was decided to perform this first task, with the purpose of the users to get to know the platform, its operation, and the commands that they can execute. The first blind subject took about 7 minutes to complete the task, the second blind subject took around 10 minutes, and the rest of the non-blind subjects, took on average, 5 minutes to complete the proposed task. A possible explanation for the deviation in the time observed in the first group is that the first blind subject took less than the second because he had academic qualifications and was more familiar with the modelling activity. Concerning the explanation for the difference between groups, the non-blind users took less time than the blind, can be explained by the fact that they have more practice and freshness in diagram modelling. Once this task was performed, it was given the opportunity to model with a modelling language of their choice, which is the second and the last task that was executed. The first blind user opted to model with a language that he learned in his computing course, the UML class diagram language. The subject explored all the possible commands of the platform, and it took him about 10 minutes to try out all the possible executable commands. In general, all users were comfortable with the tasks involved, except the second blind subject, who was not confident in the second proposed task and preferred not to execute it because he did not know which modelling language to choose for not being familiar with Software Engineering.

After the experiment, all users answered a questionnaire. The questionnaires assessed whether if they liked the experiment if ModelByVoice was challenging to use, if they would recommend the platform to other users, among other aspects. One of the open box questions was about the strong and weak points of the platform. All users highlighted the concept in the tool of the navigation element, also, the simplicity of the commands (not hard to remember), the existence of a help mode and the quality of the feedback given by the platform. Concerning the weak points, the majority of users reported some glitches on the speech recognition technology, and in their opinion, this resource is quite annoying after some time expended on modelling. Another weak point was the fact that there is no interruption of the feedback during the execution, this is, the user must wait for the end of the response of the speech synthesiser, even if he has already heard the information he would like to know.

Their suggestions for implementing change, was the use of the keyboard, instead of the voice recognition as input resource, and the ability to start and pause the voice recognition at any time with the keyboard as well.

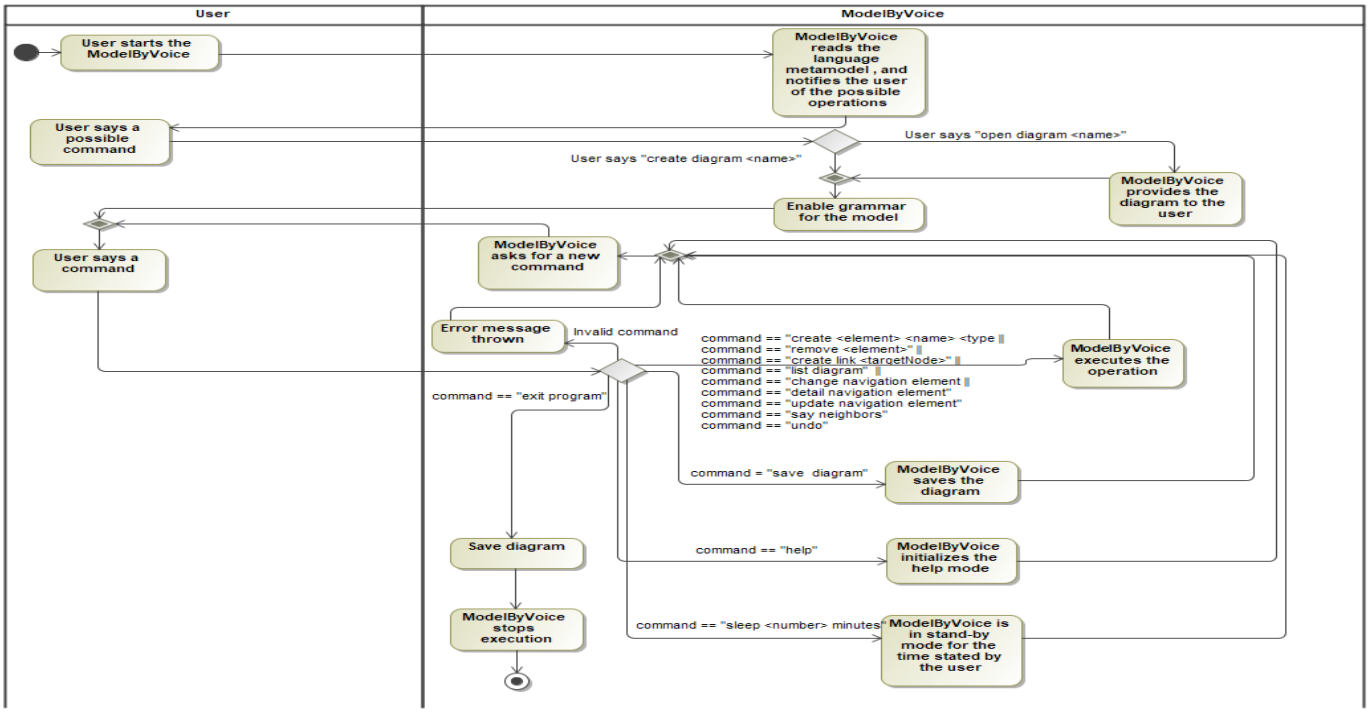

Fig. 4. Activity Diagram of the ModelByVoice

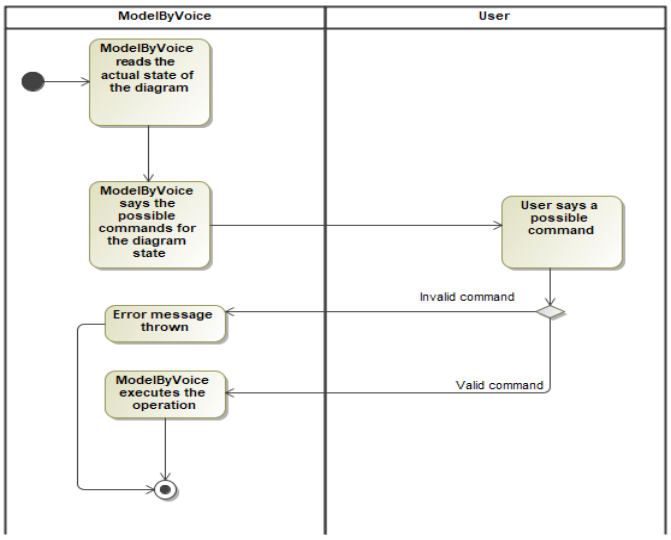

Fig. 5. Internal execution of the help mode

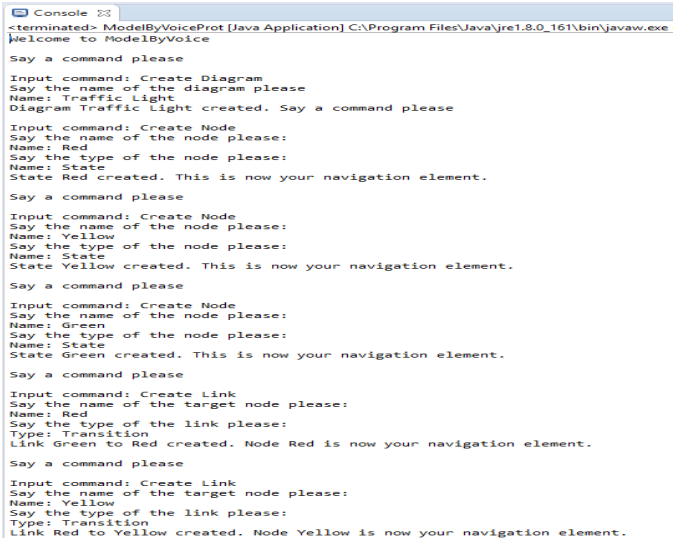

Fig. 6. Example of a session using the console in debug mode - model creation

A. *Limitations*

The evaluation included only two blind people, which is not statistically relevant. The reduced numbers of participating subjects are a threat to the validity of the preliminary assessment. However, this reflects the difficulty of contacting software engineers that are severely sight impaired. Still thanks to ACAPO, we managed to contact the two users involved in the experiment. The number of modelling tasks performed in the exercises can also be a threat due to its simplicity. However, as a preliminary assessment meant to guide us in future iterations of the tool, we opted to do so as the users have never used a voice recognition modelling application, and because one of them has not even experienced before the modelling activity. As expected, despite the reduced subject numbers, the performed tests showed promising and confident results and turned out to be valuable feedback for future improvements that are underway.

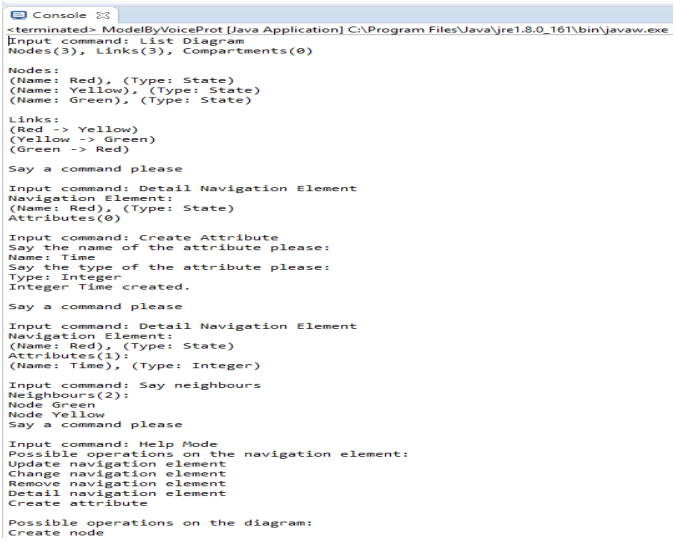

Fig. 7. Example of a session using the console in debug mode - edition with create, remove and navigation

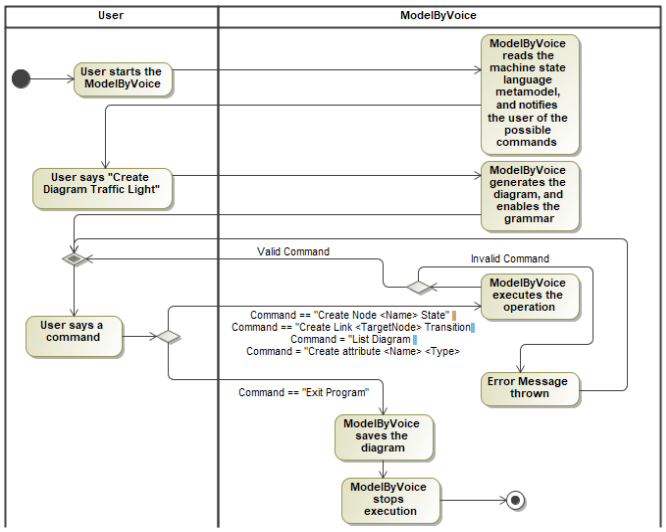

Fig. 8. State Machine Modelling Task - Traffic Light

V. Conclusions and Future Challenges

A. Contributions

This work presents a prototype tool that allows people with visual impairment to create and edit diagrams for any type of modelling language as long as this language is meta-modelled.

The goal is to provide accessibility to the modelling activity that can be reached both for blind people (and with problems with vision) and users without physical limitations. With this tool, it is possible to edit and query(navigate) diagrams, via voice. We expect that this will be an enabler for the full integration of blind people into group projects involving modelling.

The preliminary usability tests or evaluations with blind subjects and non-blind, although with anecdotal figures, allowed us to get a first feedback on the previously mentioned characteristics, since the users approved the platform, and left good suggestions of change and confidence for a future continuous development work.

B. Limitations and challenges

As mentioned in sectionIV, the main limitations pointed out by the subjects in our preliminary experiment, were related to technical failure (error rate) of the existing underneath speech recognition technology, while running ModelByVoice. The expert users (with extensive practice with software technology and used to use the keyboard) have suggested using the keyboard as an input component, as it may prove to be faster and more efficient when executing commands, instead of the speech recognition. It should be noted that this limitation in speech recognition is due to the fact that it is limited to an open source voice tool (Sphinx), and a voice tool (Google Cloud Speech-to-Text) that, despite being more efficient than the previous one, is also not 100 % reliable in translating the voice to text. Another interesting observation, besides the need for some improved interaction mechanisms (like pause), is the request for introducing audio signals instead of pure voice synthesis feed-back. We foresee that an interesting approach to deciding for the adequate concrete syntax, in this case, is to use an adaptation of the proposed design principles of PoN (Physics of Notations) [24] to audio. Those principles are currently used to evaluate, compare and enhance the communication properties a given software modelling language when designing its visual notations (concrete syntax).

C. Future Work

Taking into consideration the feedback and suggestions given by the users about the ModelByVoice, some features may prove to be interesting to implement in the future.

Among them we have to consider giving the users the possibility to choose the input resource that they intend to use to practise the modelling activity, that is, to allow them to select the voice recognition mechanism or the keyboard as an input resource, or even combine both input sources at the same time.

Another interesting upgrade in the tool's architecture is the introduction of an intermediate speech recognition platform independent layer, that would support any speech recognition API. This way, the tool would be easily reconfigured to handle with any speech recognition tool.

Additionally, it would be interesting, and straight away, to create an editor that could convert the created diagrams (which are in XMI) and generate the diagram in graphical mode. This would allow the diagrams to be analysed by other entities or people who would eventually analyse or evaluate those diagrams.

Finally, as a more challenging future work, this project raises the issue of what should be the systematic approach to assess audio concrete syntax and interaction model regarding its usability. We argue that it should be a similar framework to PoN. In our perspective, an interesting line of research could be followed in this direction.

ACKNOWLEDGMENT

The authors would like to thank NOVA LINCS Research Laboratory (Grant: FCT/MCTES PEst UID/ CEC/04516/2013) and DSML4MAS Project (Grant: FCT/MCTES TUBITAK/0008/2014).

The authors would also like to thank ACAPO (Associação dos Cegos e Amblíopes de Portugal) for providing us the contact of software professionals, and for their availability to test our tool.

REFERENCES

[1] G. Cernosek and E. Naiburg, "The value of modeling," *IBM White Paper. Retrieved July*, vol. 31, p. 2008, 2004.

[2] Stackoverflow, "Stats and analysis," Mar. 2018. [Online]. Available: https://insights.stackoverflow.com/survey/2018/#overview

[3] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: eclipse modeling framework*. Pearson Education, 2008.

[4] D. S. Kolovos, L. M. Rose, S. B. Abid, R. F. Paige, F. A. Polack, and G. Botterweck, "Taming EMF and GMF Using Model Transformation," in *Model Driven Engineering Languages and Systems SE - 15*, ser. Lecture Notes in Computer Science, D. C. Petriu, N. Rouquette, and Ø. Haugen, Eds. Springer Berlin Heidelberg, 2010, vol. 6394, pp. 211–225. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-16145-2_15

[5] S. Cook, G. Jones, S. Kent, and A. Wills, *Domain-specific Development with Visual Studio Dsl Tools*, 1st ed. Addison-Wesley Professional, 2007.

[6] J. de Lara and H. Vangheluwe, "Atom^3: A tool for multi-formalism and meta-modelling," in *Fundamental Approaches to Software Engineering, 5th International Conference, FASE 2002, held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8-12, 2002, Proceedings*, 2002, pp. 174–188. [Online]. Available: https://doi.org/10.1007/3-540-45923-5_12

[7] V. University, "Gme: Generic modeling environment," 2007. [Online]. Available: http://www.isis.vanderbilt.edu/Projects/gme/

[8] S. Kelly, K. Lyytinen, and M. Rossi, "Metaedit+ a fully configurable multi-user and multi-tool case and came environment," in *8th International Conference on Advanced Information Systems Engineering, CAiSE'96*, vol. 1080/1996. Heraklion, Crete, Greece: Springer Berlin / Heidelberg, 1996, pp. 1–21.

[9] M. Eysholdt and H. Behrens, "Xtext: Implement your language faster than the quick and dirty way," in *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, ser. OOPSLA '10. New York, NY, USA: ACM, 2010, pp. 307–309. [Online]. Available: http://doi.acm.org/10.1145/1869542.1869625

[10] S. E. Dmitriev, "Language oriented programming : The next programming paradigm," Onboard, Jetbrains, Tech. Rep., 2004. [Online]. Available: http://www.onboard.jetbrains.com

[11] S. Beydeda, M. Book, V. Gruhn *et al.*, *Model-driven software development*. Springer, 2005, vol. 15.

[12] H. Petrie, C. Schlieder, P. Blenkhorn, G. Evans, A. King, A.-M. ONeill, G. Ioannidis, B. Gallagher, D. Crombie, R. Mager *et al.*, "Tedub: A system for presenting and exploring technical drawings for blind people," *Computers helping people with special needs*, pp. 47–67, 2002.

[13] A. King, P. Blenkhorn, D. Crombie, S. Dijkstra, G. Evans, and J. Wood, "Presenting uml software engineering diagrams to blind people," in *International Conference on Computers for Handicapped Persons*. Springer, 2004, pp. 522–529.

[14] U. of Manchester, "Tedub and accessible uml," 2005. [Online]. Available: http://www.alasdairking.me.uk/tedub/index.htm

[15] B. Doherty and B. Cheng, "Uml modeling for visually-impaired persons," in *CEUR Workshop Proceedings*, vol. 1522. CEUR-WS, 2015, pp. 4–10.

[16] F. Soares, "Uma abordagem para derivar modelos de requisitos a partir de mecanismos de reconhecimento de voz," Master's thesis, Faculdade de Ciências e Tecnologia da Universidade de Lisboa, 2014.

[17] D. S. Kolovos, R. F. Paige, and F. A. Polack, "Eclipse development tools for epsilon," in *Eclipse Summit Europe, Eclipse Modeling Symposium*, vol. 20062, 2006, p. 200.

[18] D. S. Kolovos, R. F. Paige, and F. Polack, "The epsilon transformation language," in *International Conference on Theory and Practice of Model Transformations*. Springer, 2008, pp. 46–60.

[19] L. M. Rose, R. F. Paige, D. S. Kolovos, and F. A. Polack, "The epsilon generation language," in *European Conference on Model Driven Architecture-Foundations and Applications*. Springer, 2008, pp. 1–16.

[20] W. Walker, P. Lamere, P. Kwok, B. Raj, R. Singh, E. Gouvea, P. Wolf, and J. Woelfel, "Sphinx-4: A flexible open source framework for speech recognition," SMLI TR-2004-139, Sun Microsystems, Inc., Tech. Rep., 2004.

[21] W. Walker, P. Lamere, and P. Kwok, "Freetts: a performance case study," SMLI TR-2002-114, Sun Microsystems, Inc., Tech. Rep., 2002.

[22] Mbrola. [Online]. Available: http://tcts.fpms.ac.be/synthesis/mbrola.html

[23] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull, "Graphvizopen source graph drawing tools," in *International Symposium on Graph Drawing*. Springer, 2001, pp. 483–484.

[24] D. Moody, "The physics of notations: toward a scientific basis for constructing visual notations in software engineering," *IEEE T Software Eng*, vol. 35, no. 6, pp. 756–779, 2009.