Towards collaboration between sighted and visually impaired developers in the context of Model-Driven Engineering

Filipe Del Nero Grillo, Renata Pontin de Mattos Fortes, and Daniel Lucrédio**

Computer Science Department, Institute of Mathematics and Computer Sciences at University of São Paulo, São Carlos, Brazil. Av. Trabalhador São-carlense 400 - Centro, P.O.Box 668. 13560-970 - São Carlos/SP, Brazil
{grillo,renata}@icmc.usp.br,daniel@dc.ufscar.br
http://www.icmc.usp.br

Abstract. Model-Driven Engineering is rapidly emerging as a powerful way to increase quality and productivity in software development projects. However, its focus on modeling, specially with graphical notations, makes its adoption very difficult to blind and visually impaired users, who have always been able to program with the help of assistive technologies such as screen readers. Without a comprehensive and updated alternative text, this type of software artifact is of little use to a developer with visual impairment. In this paper we present ongoing research and the proposal of a tool to enable the collaboration between sighted and blind/visually impaired software developers. The tool will provide alternative textual representation to models in a web environment, so that collaboration can effectively occur. Details on the technical viability and basic functionality of the tool are presented. We believe these are of great interest to the MDE community, as other researchers and practitioners may build upon our initial ideas to develop their work. We also discuss future investigation possibilities, and the expected contributions of our research.

Keywords: MDE, Graphical, Textual, Accessibility

1 Introduction

Computer programming has historically been a field in which the visually impaired were able to work and teach, because programs are essentially text and, therefore, accessible by the use of assistive technologies such as screen readers. However, the use of visual models became more popular with the growth of the software engineering discipline and visual languages such as the Unified Modeling Language (UML). Visual languages were designed to capture and structure complex problems such as architectural design and requirement specification [1],

** Computing Department at Federal University of São Carlos, Rod. Washington Luís, Km 235 - 13565-905 São Carlos-SP

but as they rely on complex visual-dependent software to be developed and read, the blind and visually impaired have many restrictions or do not have access to them at all [9, 13].

As a workaround, blind developers often depend on others to read and explain the concepts to them. This is not a serious problem if we consider traditional software development, where models are used mainly as support and documentation that help programmers to understand what needs to be done in terms of a software solution. The actual software, i.e. the code, is still accessible by the blind and visually impaired. But with the rise of Model-Driven Engineering (MDE), many types of models and their corresponding visual notations gained a new importance in the software project life cycle. On the model-driven approach, models are the main artifact of the development process [7], and are actually used as input to automatic software transformation and code generation. As a consequence, direct access to them is essential, which poses a serious issue to people with visual disabilities.

In theory, every visual model can be translated into a corresponding textual model. For instance, most UML tools are capable of importing/exporting models to the XMI format (XML Metadata Interchange) [15], which is a textual representation. Of course, XMI has many readability issues, but we believe a similar translation process can be applied to produce a much more readable representation, thus making visual models accessible through screen readers. In a model-driven scenario, blind and visually impaired developers could use this alternative representation to read and modify models directly, thus actively collaborating in the development process.

This paper presents an ongoing work towards a multiple representation of models using graphical and textual notations and proposes the Accessible Web Modeler (AWMo), a web-based tool that aims to leverage collaboration with visually impaired users by using multiple presentations for models. Figure 1 demonstrates how the collaboration will happen: with the use of a screen reader, a blind or visually impaired software developer will be able to interact with the textual model while other developers can interact with a traditional graphical model they are used to, when working on the same model. Changes made to one of the views will be shown on the other view and vice-versa.

The remainder of this paper is organized as follows: Section 2 introduces Model-Driven Engineering concepts. Section 3 shows some of the related work found in the literature. Section 4 presents more details about the ongoing research and proposal of the AWMo tool. In Section 5 we present a discussion on the future possibilities of how valuable research data can be extracted from evaluations involving AWMo. Finally, in Section 6 we conclude the paper with some final remarks.

2 Model-Driven Engineering

Model-Driven Engineering (MDE) is the combination of generative programming, domain-specific languages and software transformations, concepts that

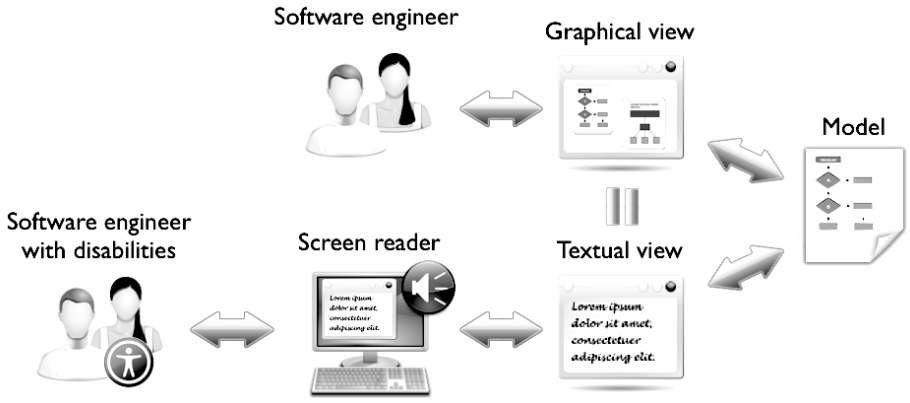

Fig. 1. Schematic of how the AWMo tool will work. Developers should be able to edit the model on both graphical and textual views and achieve the same results.

have been explored since 1980 [14, 11]. Its purpose is to reduce the semantic gap between the problem and the solution/implementation, through high level models that shield developers from the complexities of the underlying platform [7]. In MDE, models are used to express domain concepts more effectively, while transformations automatically generate the artifacts that reflect the solutions contained in the models [19].

As a result, the task of the developer becomes simpler and less repetitive. With the use of automations and the higher level of abstraction, the developer can work on a much more conceptual level, leaving implementation details to the responsibility of the code generators.

A particularly effective way of specifying models in the context of MDE is Domain-Specific Modeling (DSM). In contrast with generic modeling approaches like those using UML, DSM uses a Domain-Specific Language (DSL), a smaller, more focused language, designed specifically to provide maximum expressiveness in a particular area or domain [3]. DSLs are normally used in MDE due to some problems with general-purpose languages: UML, for instance, requires that special markings are inserted in order to facilitate transformations, which ends up polluting the models [10]. Even extension mechanisms, which are used in practice to solve part of the problems, are insufficient when MDE is being adopted [10], given its low expressiveness, flexibility and inadequate notation [22].

In MDE, models with different levels of abstraction may be used, from models that have no relation with any computation platform to models that are highly specialized to a specific platform such as Web or mobile development.

Transformations are procedures used to transform a model into another artifact during the software development life cycle. With transformations it is possible to transform a model into a different model, documentation artifacts or even executable code.

3 Related work

In [8], Guerra et al. explore multiple graphical views for the same model, maintaining the consistency between them by the use of a global model and triple grammars. In a later evolution of their work, the authors explore the use of multiple notation on the metamodeling tool AToM^3 with the goal of allowing users to use either visual and textual notations on what they called *Multi-View* DSLs (Domain-Specific Languages). The main goal of their project was to allow the user to model on the notation that is better suited for different perspectives or viewpoints [16].

The TeDUB project (Technical Diagram Understanding for the Blind) tried to address some of the issues of access of blind users to UML by creating a UML reader for the blind, where the users were able to open common UML diagram files like XMI (XML Metadata Interchange), navigate and access its contents with the use of a joystick, sound cues and text to speech [9].

The work of Metatla et al. [12] and Bryan-Kinns et al. [1] explore the use of cross-modality to make diagrams more accessible to workers with visual disabilities. In this context, cross-modality means using more than one sensorial channel to convey or acquire information. A workshop was organized to help the researchers better understand the issues they were dealing with and the needs of their target users. As a result of the workshop they identified that the two limitations that all current aproaches share are the inability to create and edit the diagrams without the assistance of a sighted person and the inefficiency on use of collaborative interaction.

Our approach is closer to the work of Guerra et al. [8, 16], however we intend that the two views represent the whole model, instead of submodels or projections of the model, creating different perspectives. This is required mainly because the different views in our approach are not meant to be used together, in fact, they should be able to completely replace each other in terms of model understanding. We believe a consistent mapping between visual and textual notations can bring better results, given the familiarity that the blind and visually impaired users have with text and screen readers.

The literature also has some reports about Web modeling environments and applications. One of them is SLiM (Synchronous Lightweight Modeling). SLiM is an environment that allows users to collaborate on modeling activities in a synchronized way. It uses techniques such as COMET [2] for server communication and Scalable Vector Graphics [6] for the diagram visual representation over the Web. The main goal of SLiM is to allow the collaboration of users that are geographically apart [20].

Another example is GEMSjax [5], a Web implementation of GEMS (Generic Eclipse Modeling System [23]). GEMS is a project that was created by the Eclipse Foundation to bring together the experience about visual metamodeling tools of the GME community at Vanderbilt university and Eclipse communities such as EMF and GMF. The GEMSjax uses the Google Web Toolkit framework to create a Web interface for modeling and metamodeling activities. There are some other examples that are not Web applications, but Desktop, such as COMA

(COllaborative Modeling Architecture tool) that focus on collaboration [18] and the Eclipse Foundation GEMS that inspired and was used by the GEMSjax mentioned earlier.

Our approach is also web-based, and aimed at leveraging collaboration, and thus we intend to employ well-known techniques for this kind of tool. However, our focus is on the inclusion of blind and visually impaired users, and therefore the support for synchronization and real-time collaboration will be limited.

4 Proposed tool and methods

In this section we discuss the building blocks of our approach. We discuss the technical viability of AWMo and then present more details on how the tool should work.

4.1 Viability

The web environment of AWMo will be based on JSF (Java Server Faces), which will be used to construct the menus and basic interaction functionality. Accessibility guidelines [21] will be adopted during the construction of this base environment, helping to make the interface accessible.

For the textual representation, we intend to use Xtext [4]. Xtext provides a set of tools that allows the definition of a grammar in EBNF (Extended Backus–Naur Form) notation and generation of a textual editor resources such as a language parser, validators and code generators. Xtext can also generate an Ecore metamodel and EMF (Eclipse Modeling Framework) classes from the language grammar. This makes the programmatic management of the parsed (textual) models and metamodels possible.

One of the advantages of Xtext is that the generated tools are independent from the Eclipse environment and can be used, for instance, in a JSF Web application. This indicates that the integration of the Xtext tools with JSF for the development of the AWMo tool is possible and will happen naturally.

For the visual modeling functionality, such as diagram creation and disposition of the visual elements, we intend to use some pre-existing web-based library for graph rendering and construction. Some examples include Raphaël JS[1], jsUML2[2] and Joint JS[3]. From these, we intend to adopt the jsUML2, which is a JavaScript library that already implements many aspects of UML modeling in a web application.

With Xtext, EMF and a visual library, much of the desired functionality is already available: textual language definition, parsing, model manipulation and the creation of visual diagrams. What is left is the mapping between the parsed (textual) model and the appearance and disposition of the visual elements, which will have to be built by hand. However, the generated EMF classes will greatly facilitate this task.

[1] http://raphaeljs.com/
[2] http://code.google.com/p/jsuml2/
[3] http://www.jointjs.com/

4.2 Our approach

Initially, we intend to build a simplified UML class model, mainly because of jsUML2. However, other than that, we see no reason why the idea could not be later extended to other types of models.

Once the textual grammar is defined using Xtext, and the respective textual language infrastructure is generated, they will be integrated into the AWMo JSF Web application. The language parser will be used to parse the defined textual class model and create an EMF-based representation of the model. This representation will be used, along with external positioning data, to transform the textual model into a notation that is accepted by the jsUML2 JavaScript library on the application view.

The following code shows an example of a typical textual class model in AWMo:

Example of the AWMo class diagram textual model

```
class Person {
    attribute firstName, type string, visibility public
    attribute lastName, type string, visibility public

    method getName, return null
}

class Student {
    inherit Person

    attribute grade, type float, visibility private

    method setGrade, return void, parameters  {
        parameter grade, type float
    }

    method getGrade, return float, no parameters
}
```

One of the issues faced by having both graphical and textual views for the same model is that some of the information about the model will be only present on the graphical view. As an example we can cite the spatial information regarding a class such as its X and Y coordinates on the diagram. We intend to store any spatial information apart from the model itself, so it can be used when displaying the graphical editor but will not be represented or even parsed during the use of the textual editor. If spatial information is not available, for example when a developer creates a new element in the textual view, default coordinates will be assigned to it on the graphical representation. This position may not be adequate, however it could be later changed by another user, without any effect on the textual view.

Figure 2 shows the basic AWMo textual and visual modeling scheme. The AWMo model is made of the combination between the textual model, which contains all the semantic and information about the model that conforms to the Ecore metamodel generated by Xtext, and the spatial information which contains all the data required to display the textual model in a graphical way. Every time the model is saved from one of the views, the entire AWMo model will have to be checked in order to maintain the relation between the textual model and the spatial information. For example, if a class is removed from the textual view, its spatial information must also be removed. In the case a new class is added, its the spatial information must be created with default values, as mentioned earlier.

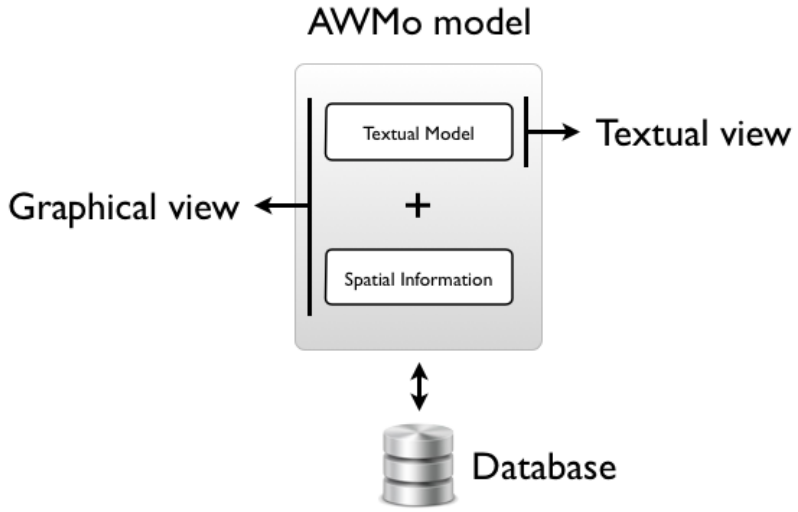

Fig. 2. Representation of the data structure store on database for the AWMo models including both textual and graphical information that are stored apart.

Figure 3 shows an example of use of the proposed tool. At first, on (A), the graphical view is shown and the user added a Person class along with its methods and attributes in the same way he would have done on his preferred UML tool. When the textual view is accessed, the user should see a textual representation of the Person class that is semantically equivalent to the graphical model (B). Then on the same textual view, the user adds a second class, Student (C). Once again, when accessing the graphical view, both Person and Student classes will be visible, reflecting the changes performed by the user on the textual view (D). At last, the developer adds as specialization between the Student and the Person classes (E); This last change is also reflected on the graphical view (F). This

simple use case illustrates the bidirectional nature of the approach and gives an idea on how the collaboration between users of the different views will happen.

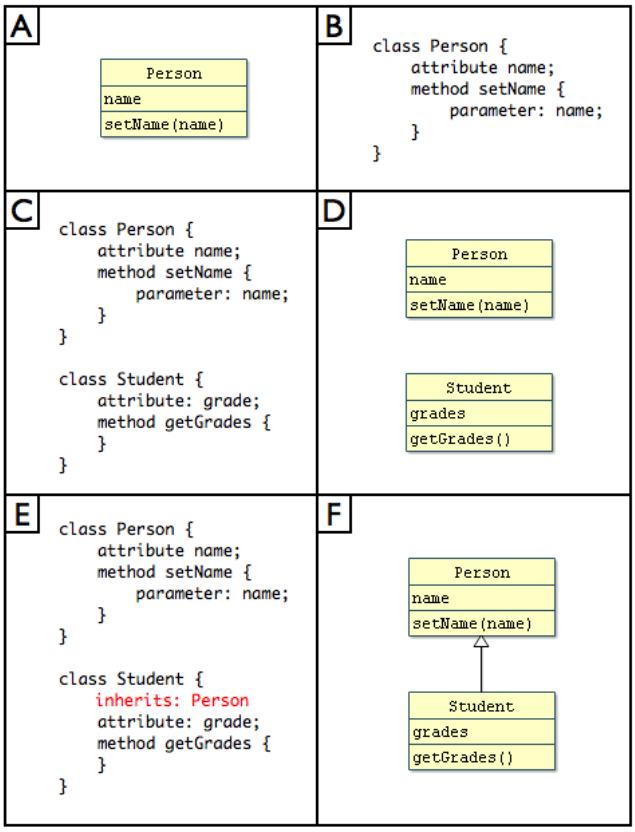

Fig. 3. Illustration of a usage scenario for the AWMo where both graphical and textual views are used to build a simple class model

5 Discussion

With the AWMo tool, there are many evaluation areas to explore. One of the possibilities is to evaluate how it leverages the collaboration between blind or visually impaired and sighted software engineers when modeling with the proposed tool. We expect to completely eliminate the need for an auxiliary person to read diagrams, thus making this collaboration possible in a MDE scenario.

Another common problem we faced while contacting computer programmers with disabilities is that some of them never came to learn UML class diagram

because of their special needs and the lack of accessible tools. This makes the process of gathering initial requirements for AWMo a challenge. For that reason we plan on developing AWMo on an interactive way, building a first version that will be incrementally evolved in constant contact with real users, in order to identify their needs with the working tools and then iterate on improvements and validations.

Another – more delicate – issue is the so-called "secondary notation". Usually on a model, there are many ways to convey the same information, however the clarity and readability of the information relies on something that is not always on the language syntax. This is called 'secondary notation', and is defined as additional visual cues that are not part of the language itself but greatly affects the way the information on the model is perceived [17]. An example of such cues is when elements of a graphic that are closely related are represented near each other in a diagram. This information is not part of the language itself and an alternative graphic with the two elements far from each other does not make the graphic wrong according to the visual language syntax, but strongly affects the way the graphic will be read. Another example is inheritance: classes on the top of a diagram are usually higher on the class hierarchy. The same happens with textual models. In most languages, indentation is not part of the syntax, and yet they play an important part on readability. The exact part played by this notation and its importance to blind and visually impaired developers will have to be investigated.

Another possibility of investigation is how these visual cues pointed by Petre (1995) [17] can influence the use of the graphical representation and if, in such cases, the textual representation presents any advantages to help the users understand the models. This comparison transcends the accessibility scenario, and may be useful even without considering the blind and visually impaired developers. We believe there are many interesting issues that could be identified by means of usability and HCI evaluation techniques.

6 Concluding remarks

Visual modeling has been a major problem for blind and visually impaired developers. Without the natural capacity of understanding visual information, these types of model are of little use to these people, who depend on other types of mental abilities to develop software. Most are able to program through the use of screen readers, but in the new MDE scenario the increased importance of models makes this task nearly impossible.

In this paper we present an ongoing work and a proposal for a tool that will allow the collaboration between sighted and blind/visually impaired users in a model-based software development project. We discuss its technical viability and our ideas of how such functionality will be implemented in a web tool. We believe this information can generate healthful discussion and provide some insight for other researchers and practitioners interested in web-based modeling and accessibility.

We also discuss the future possibilities for investigation, highlighting different types of data that could be gathered once the tool is ready. We expect to be able to identify interesting issues and draw important conclusions in the MDE and software engineering areas, helping to advance our knowledge and deliver such inclusion benefits for our society.

References

1. Bryan-Kinns, N., Metatla, O., Stockman, T.: "collaborative cross-modal interfaces". In: Proceedings of Digital Futures '10 RCUK Digital Economy All Hands Meeting (2010)
2. Crane, D., McCarthy, P.: Comet and Reverse Ajax: The Next-Generation Ajax 2.0. Apress, Berkely, CA, USA (2008)
3. Deursen, A.v., Klint, P., Visser, J.: Domain-specific languages: An annotated bibliography. SIGPLAN Notices - ACM Press 35(6), 26–36 (2000)
4. Eysholdt, M., Behrens, H.: Xtext: implement your language faster than the quick and dirty way. In: Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion. pp. 307–309. SPLASH '10, ACM, New York, NY, USA (2010)
5. Farwick, M., Agreiter, B., White, J., Forster, S., Lanzanasto, N., Breu, R.: A web-based collaborative metamodeling environment with secure remote model access. In: Proceedings of the 10th international conference on Web engineering. pp. 278–291. ICWE'10, Springer-Verlag, Berlin, Heidelberg (2010)
6. Ferraiolo, J., Jackson, D.: Scalable vector graphics (SVG) 1.1 specification. W3C recommendation, W3C (Jan 2003), http://www.w3.org/TR/2003/REC-SVG11-20030114/
7. France, R., Rumpe, B.: Model-driven development of complex software: A research roadmap. In: 29th International Conference on Software Engineering 2007 - Future of Software Engineering. pp. 37–54. IEEE Computer Society, Minneapolis, MN, USA (2007)
8. Guerra, E., Diaz, P., de Lara, J.: A formal approach to the generation of visual language environments supporting multiple views. In: Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on. pp. 284 – 286 (sept 2005)
9. King, A., Blenkhorn, P., Crombie, D., Dijkstra, S., Evans, G., Wood, J.: Presenting UML Software Engineering Diagrams to Blind People. In: Miesenberger, K., Klaus, J., Zagler, W., Burger, D. (eds.) Computers Helping People with Special Needs, Lecture Notes in Computer Science, vol. 3118, pp. 626–626. Springer Berlin / Heidelberg (2004)
10. Kühne, T.: Making modeling languages fit for model-driven development. In: Fourth International Workshop on Software Language Engineering, Nashville, USA (2007)
11. Lucrédio, D., Fortes, R.P.d.M., Almeida, E.S.d., Meira, S.R.d.L.: The Draco approach revisited: Model-driven software reuse. In: VI WDBC - Workshop de Desenvolvimento Baseado em Componentes. pp. 72–79. Recife - PE - Brazil (2006)
12. Metatla, O., BryanKinns, N., Stockman, T., Martin, F.: Designing for collaborative cross-modal interaction. In: Proceedings of Digital Engagement '11: The 2nd Meeting of the RCUK Digital Economy Community (2011)
13. Metatla, O., Bryan-Kinns, N., Stockman, T.: Comparing interaction strategies for constructing diagrams in an audio-only interface. In: Proceedings of the 22nd

British HCI Group Annual Conference on People and Computers: Culture, Creativity, Interaction - Volume 2. pp. 65–69. BCS-HCI '08, British Computer Society, Swinton, UK, UK (2008)
14. Neighbors, J.M.: Software Construction Using Components. Ph.d. thesis, University of California at Irvine (1980)
15. OMG: MOF 2 XMI Mapping Specification. Disponível em http://www.omg.org/spec/XMI/2.4.1/. Acesso em 15/01/2012 (08 2011)
16. Pérez Andrés, F., de Lara, J., Guerra, E.: Domain specific languages with graphical and textual views. In: Schürr, A., Nagl, M., Zündorf, A. (eds.) Applications of Graph Transformations with Industrial Relevance, Lecture Notes in Computer Science, vol. 5088, pp. 82–97. Springer Berlin / Heidelberg (2008)
17. Petre, M.: Why looking isn't always seeing: readership skills and graphical programming. Commun. ACM 38, 33–44 (June 1995)
18. Rittgen, P.: COMA: A tool for collaborative modeling. In: CAiSE Forum - CEUR-WS.org. pp. 61 – 64 (2008)
19. Schmidt, D.C.: Guest editor's introduction: Model-driven engineering. IEEE Computer 39(2), 25–31 (2006)
20. Thum, C., Schwind, M., Schader, M.: SLIM - A Lightweight Environment for Synchronous Collaborative Modeling. In: Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems. pp. 137–151. MODELS '09, Springer-Verlag, Berlin, Heidelberg (2009)
21. W3C: Web Content Accessibility Guidelines (WCAG) 2.0 - W3C Recommendation. Disponível em http://www.w3.org/TR/WCAG20/. Acesso em 23/01/2012 (12 2008)
22. Weisemöller, I., Schürr, A.: A comparison of standard compliant ways to define domain specific languages. In: Fourth International Workshop on Software Language Engineering, Nashville, USA. megaplanet.org, Grenoble, France (October 2007)
23. White, J., Schmidt, D.C., Mulligan, S.: The Generic Eclipse Modeling System. In: Model-Driven Development Tool Implementer's Forum at 45th International Conference on Objects, Models, Components and Patterns (2007)