Guidelines for Accessible Textual UML Modeling Notations

Vanessa Petrausch[1], Stephan Seifermann[2] and Karin Müller[1]

[1] Karlsruhe Institute of Technology, Germany
{vanessa.petrausch, karin.e.mueller}@kit.edu
[2] FZI Research Center for Information Technology, Germany
seifermann@fzi.de

Abstract. Textual representations of UML are basic requisites to make UML modeling accessible for visually impaired people. The accessibility, however, varies depending on the concrete realization. Constructing and rating accessible notations is challenging because the notation has to consider requirements of various assistive techniques including screen readers with audio and/or braille output. Neither accessibility metrics for existing textual notations nor comprehensive guidelines for constructing such notations exist. To bridge this gap, we design an interview for rating the accessibility of notations for UML class diagrams and conduct it with six participants for four textual notations. We use the results and related work to derive general design guidelines for accessible textual UML notations. The guidelines allow constructing accessible notations without deep understanding of assistive technologies and can serve as a benchmark for existing notations.

Keywords: UML, Textual Notation, Survey, Accessibility, Formal Modeling, Language Design, Guidelines

1 Introduction

Modeling is frequently used in the software engineering process to describe various aspects of a system in an abstract way. The Unified Modeling Language (UML) is the most commonly used modeling language. It provides a complete graphical notation but only patchy textual notations for parts of diagram types. The lack of a standardized accessible notation impedes equal participation of visually impaired software developers and engineers even if the demand for IT experts is high.

Textual notations are considered accessible in general but their concrete realization influences accessibility. For instance, a verbose notation hinders the usage of braille displays since reading the notation is onerous and reduces the working efficiency. Furthermore, most existing notations do not focus on accessibility and thus do not consider the needs of assistive technologies.

Accessibility ratings for existing textual notations do not exist. Creating such a rating is, however, challenging because of missing rating criteria or notation design guidelines for deriving such criteria.

The contribution of this paper is a comprehensive set of design guidelines for constructing accessible textual UML notations. Notation designers do not need to have extensive accessibility experience to use them. The guidelines also serve as a benchmark for existing notations. We derived the guidelines from related work and an accessibility survey for existing textual UML notations. We drafted an interview sheet for UML class diagram notations that can be reused for further accessibility assessments. Two blind computer scientists, two researchers in the field of assistive technology and two researchers in the field of computer science participated in interviews using those interview sheets.

As a result, we derived five guideline categories covering usability, accessibility and technical aspects. The guidelines contain 18 concrete recommendations for textual UML notations which are not only applicable for class diagrams but for other UML diagram types as well.

The remainder of the paper is structured as follows: section 2 shows related work. We describe our research methodology for the accessibility survey in section 3 and present the results in section 4. Section 5 describes the design guidelines derived from related work and the survey results. We conclude and outline future work in section 6.

2 Related Work

Textual notations are beneficial from an accessibility and a general usability point of view. Loitsch and Weber [4] state that textual notations exploit existing assistive techniques such as screen readers although they focus on tactile displays to make UML diagrams accessible. Various approaches for transforming UML models into haptic representations such as 3D printing [1] exist but lack editing support. Grönniger et al. [2] present advantages of textual notations for general usability including fast editing, layouting and conciseness.

Many surveys on textual notations for UML modeling exist. Luque et al. [5] surveyed 27 and Müller [7] three approaches that exploit textual notations for making UML diagrams accessible. Seifermann and Groenda [11] surveyed 31 notations with focus on UML coverage, editing experience and applicability in engineering teams. No survey focused on rating the accessibility of notations.

Additionally, research on guidelines for textual notations exists. Patil et al. [9] define design guidelines for making textual information accessible. When designing UML notations, the suggestions of naturally expressing concepts and supporting the user's environment are applicable. The other guidelines given by Patil focus on tool support rather than the notation. The W3C formulated general accessibility principles for websites [12] that partially apply to textual UML notations. Karsai et al. [3] derive general design guidelines for improving the usability of languages and cover language purpose, realization, content as well as the concrete and abstract syntax. We cannot apply the guidelines for language content and abstract syntax because the UML [8] already defines them. Mazanec and Macek [6] also define general usability guidelines for textual notations.

3 Methodology for Deriving Accessible Notation Concepts

The overall objective is the definition of design guidelines that allow creating accessible textual UML notations. We mine the available general guidelines for textual notations from the related work section and complement them with rules we derive from expert interviews on accessible realization concepts of textual notations. The following paragraphs describe the selection of representative textual notations and their elements, the interview sheet and the interview conduction.

The interview shall cover commonly used realization concepts of textual notations. For our initial interview sheet, we restrict the interview to elements usually found in class diagrams to not overcharge our participants. We base the selection of representative notations on our previous survey on textual UML notations [11]. We categorize the notations in two dimensions: *Modeling Focus* and *Modeling Objectives*. Notations in the same category express aspects in a similar way. Figure 1 gives an overview on the categories and the selected notations.

	Conceptual HUTN	Modeling Objective: Graphics Generation	Programming	Analysis	Input for further Processing
Modeling Focus	Sketching	MetaUML Pgf-umlcd UMLet **yUML** **PlantUML**	UMLGraph		TCD
	Modeling	AWMo Modsl Nomnoml	Alf txtUML UML/P **Umple**	Clafer tUML USE	**Earl Grey** TextUML

Fig. 1. Categorization of the analyzed textual notations

The *Modeling Focus* covers the modeled element's type. *Sketching* focuses on graphic-oriented elements. Notations fall into this category if users have to describe graphical shapes rather than concepts. *Modeling* produces layout agnostic elements. The *Modeling Objective* covers the motivation for modeling. The classification into the categories *Graphics Generation*, *Programming* and *Analysis* relies on the intended usage stated by the notation vendors. We also define the objective of a notation to be *Programming* if the language is based on a programming language. If tools generate formal UML models without defining later usage, the objective is *Providing Input for Further Processing*.

We selected at most one textual notation from each combination of *Modeling Focus* and *Modeling Objectives* because notations in such a group express UML concepts in a similar way. We do not consider *Analysis* notations because these are usually tailored for a specific need and therefore often not applicable for general purpose modeling. We considered yUML from sketching because it has a concise, pure graphical notation and thus is an interesting candidate for comparison. We omit other sketching notations because TCD is basically text-based visual art (also called ASCII art) and therefore are not accessible by definition. The syntax of UMLGraph is too close to corresponding notations in the *Modeling* category to gain more insights. For the *Programming* objective, we select Umple over the other candidates because it is well established, covers many language constructs, has a comprehensible documentation, is not fully based on a programming language and is concise. For the *Input for further Processing* objective, we select Earl Grey over TextUML because the former focuses on usability, which often implies accessibility.

In order to compare the notations, we draft an interview sheet that distinguishes between available elements, realization aspects and realization concepts. Available elements are the UML elements that can be modeled. A realization aspect groups realization concepts for a certain purpose such as the location of an element or the order of its parts. Realization concepts are concrete ways of representing realization aspects. An excerpt of the interview sheet is given in Table 1 expressing the element *Relation* and four concepts of realizing the aspect *Display*: It can be located within a class, within a separate section, repeatedly within all involved classes or represented as an attribute. We added examples for each notation to allow easier rating of the realization concept. The complete interview sheet is available on our project's website[3].

Table 1. Excerpt of the interview sheet used for rating the accessibility of textual UML class diagram notations. The example shows how a relation could be displayed. The users rated three times for PlantUML and three times for the realization as a separate section.

Realization	**Example**	**Notation Example**
Within a class	class ClassA { − > ClassB }	Umple: class A{ ∗− > ∗ B; }
Separate section	relation { from Element A; to Element B; }	PlantUML: class A − > class B
Within all involved classes	class ClassA { − > ClassB } class ClassB { − > ClassA }	
Like Attribute	class ClassA { ClassB reference; }	

We conduct expert interviews with two blind computer scientists, two researchers in the field of assistive technologies and two computer science researchers. The three authors are part of the two latter groups. This procedure

[3] `http://www.cooperate-project.de/icchp2016`

ensures that our notation fits the needs of people with and without visual impairment and supports formal modeling as well. Participants are asked to rate the concept or concrete notation example they prefer to realize the given aspect.

4 Results of Our Accessibility Survey

The experts rated 29 aspects for 21 class diagram elements. Because of size limitations, we cannot publish all results here. Nevertheless, we elaborate on our general findings. The complete raw results are available on the project's website[3].

The participants rated the different notations due to accessibility, easy comprehensibility and technical feasibility in case of the two computer science researchers. Participants chose the best notation for each element and its different aspects of the class diagram from their point of view. In total, PlantUML is chosen for 21 out of 29 aspects, Umple for eight aspects, Earl Grey for three aspects and yUML never. Choosing means that at least one participant rated the notation best. PlantUML is chosen in more than 50% of all aspects and thus seems to be preferred. In 13 ratings, all participants chose the same realization concept of an aspect, even if they chose different notations. For example, all participants decided using a *separate section* (see Table 1) for relations, although three chose PlantUML, one Earl Grey and two chose the concept rather than a concrete notation. Table 2 visualizes the relation location representation in the surveyed notations by a simple directional relation from class "Teacher" with an attribute "subject:String" to class "Pupil". PlantUML, yUML and Earl Grey use a separate association section, whereas Umple represents associations like attributes. The ratings indicate that the participants preferred graphical imitations for relations instead of verbose textual descriptions like used in Earl Grey. Pure graphical representations such as yUML are uncomfortable.

In cases of disjoint opinions, further research and discussions will be done to gain more insights. The results, however, show that participants prefer a mix of graphical imitations and textual descriptions, separated blocks for defining elements like classes and prefer compact notations rather than verbose ones although this reduces intuitive understanding of an element. The complete results are summarized within our derived guidelines in the next section.

Table 2. Excerpt from simple class diagram modeled in four different textual notations

PlantUML

```
class Teacher{
  subject: String
}
Teacher -> Pupil
```

Umple

```
class Teacher{
 String subject;
 1 -> 1 Pupil;
}
```

Earl Grey

```
class Teacher
  subject: String
end
association
  Teacher[1]
  Pupil[1]
end
```

yUML

```
[Teacher|subject: String]
[Teacher]->[Pupil]
```

5 Design Guidelines for Accessible Textual UML Notations

We derive our design guidelines from the results of the expert interviews and the analysis of existing guidelines for accessibility and language design in general. Our guidelines cover accessibility and usability conditions but also technical feasibility to provide an easy integration within development frameworks. The literally fundamentals of the guidelines are taken from Groenninger et al. [2], Mazanec and Mace [6], Patil et al.[9] and Paige et al. [10]. We extend or filter these guidelines based on the results of our expert interview. We reviewed existing guidelines and our final guidelines from three different viewpoints to develop a notation most suitable for diversity teams. We primarily focus on the view of software developers with visual impairments, secondly the view of people working in a diversity team and last from technical feasibility.

The resulting guidelines shown in Figure 2 include five top-level categories with several concrete realizations: (1) Usability of Textual Notation, (2) Accessibility Support for Visually Impaired, (3) Notation Realization, (4) Concrete Syntax and (5) Functional Interaction between Notation and Tools.

Usability of Textual Notations Usability plays an important role to allow all participants working efficiently. Thus, the notation should be simple, easy to learn and intuitively comprehensible. Notations fulfilling these criteria guarantee team members working with two different representations (visual vs. textual) a smooth communication and efficient working. Furthermore, if the concepts are consistently designed, learning is facilitated.

Accessibility Support for Visually Impaired The notation is mainly designed to improve access to UML for visually impaired people and thus we focus on accessibility features. The reading techniques of persons with visual impairment differ significantly from those of sighted persons. They cannot skim texts to pick certain information. Thus, it is important to provide mechanisms which compensate skimming to allow a fast overview and easy search in a textual notation. The identified compensation mechanisms proved to be beneficial in many consulting sessions with visually impaired persons.

Notation Realization This category focuses on general notation conditions. Mimicking some graphical shapes with ASCII code is compact and intuitively understandable by sighted users, e.g. PlantUML uses (*) for the starting point in activity diagrams. Although compactness is preferred by blind persons, they have to learn these arbitrary letter sequences by heart as visual mnemonics are not available. In contrast to the finding in our study, we follow the objections of our blind participants regarding visual mnemonics. We choose textual realizations such as abbreviations based on natural language as they are compact and more easily to learn and to remember. It is also helpful to reuse meaningful text elements used in other notations.

Concrete Syntax The concrete syntax defines the concrete representation of the modeled elements by graphics, texts, or other ways. Browsing code can be very cumbersome. Thus, it helps using a) brackets to show element belongings and ease searching for certain elements b) the same styles for each abstraction level c) programming syntax for programming paradigms. Furthermore, ambiguity of expressions should not be allowed.

Functional Interaction between Notation and Tools Textual notations for software development are designed to be used in digital form especially when visually impaired persons are involved. A tool should support both sighted and visually impaired persons in editing. Code completion can be used by both groups whereas syntax highlighting can be used by sighted users only. Tools should provide similar support for both groups. Moreover, it is necessary that notations are usable across different platforms and support version control mechanisms.

1. **Usability of Language**
   (a) Keep the language simple and consistent
   (b) Create an intuitive language
   (c) Easy to learn
   (d) Ensure consistency of concepts
2. **Accessibility Support for Visually Impaired**
   (a) Constrain certain general structure
   (b) Use title for main structures
   (c) Enable transcribing mechanisms (Braille, Voice, Text)
3. **Language Realization**
   (a) Minimize graphical realizations
   (b) Use textual realizations based on natural language
   (c) Use descriptive notations, based on common known keywords
   (d) Adopt existing notations if appropriate
4. **Concrete Syntax**
   (a) Use special characters to provide organizational structures
   (b) Allow the same style for every abstraction level
   (c) Use programming syntax for programming paradigms
   (d) Use unambiguous expressions
5. **Functional Interaction between Notation and Tools**
   (a) Provide platform independence
   (b) Support writing by certain mechanisms for sighted and visual impaired persons
   (c) Provide version control mechanisms

Fig. 2. Overview of all guidelines

6 Conclusion and Future Work

Textual notations make UML modeling accessible. However, comprehensive guidelines for designing and rating such notations do not exist. We bridge this gap by drafting an interview sheet for assessing the accessibility of textual notations for UML class diagrams. We conduct an expert interview with six participants and rate four representative textual notations covering 29 realization aspects. Our participants often chose the same realization concepts independently of a potential visual impairment or working domain.

Textual notation designers can use our interview sheet to survey the accessibility of their notation. Additionally, they can construct new sheets for other diagram types by applying our method described in section 3.

Based on the results of our survey and existing guidelines for designing textual notations, we derive our design guidelines for the notation developed in the Cooperate project. The guidelines are organized in five main categories and cover aspects for accessibility, usability and implementation feasibility. The guidelines give 18 concrete instructions in total for designing a textual UML notation.

Our next step is to evaluate the effect of editing support such as code completion on accessibility and whether editing support can compensate insufficiencies of textual notations. We will also evaluate how notations constructed according to our guidelines fit the requirements of screen readers and Braille displays and how they enhance working processes for UML diagrams.

Acknowledgements

This work has been funded by the German Federal Ministry of Labour and Social Affairs under grant 01KM141108.

References

1. Doherty, B., Cheng, B.H.C.: UML Modeling for Visually-Impaired Persons. In: HuFaMo'15. pp. 4–10 (2015)
2. Grönninger, H., Krahn, H., Rumpe, B., Schindler, M., Völkel, S.: Textbased Modeling. In: ATEM'07 (2007)
3. Karsai, G., Krahn, H., Pinkernell, C., Rumpe, B., Schindler, M., Völkel, S.: Design Guidelines for Domain Specific Languages. In: DSM'09 (2014)
4. Loitsch, C., Weber, G.: Viable Haptic UML for Blind People. In: ICCHP'12. pp. 509–516 (2012)
5. Luque, L., de Oliveira Brandão, L., Tori, R., Brandão, A.A.F.: On the Inclusion of Blind People in UML e-Learning Activities. RBIE'15 23(02), 18 (2015)
6. Mazanec, M., Macek, O.: On General-purpose Textual Modeling Languages. In: DATESO'12. pp. 1–12 (2012)
7. Müller, K.: How to Make Unified Modeling Language Diagrams Accessible for Blind Students. In: ICCHP'12. pp. 186–190 (2012)
8. OMG: Unified Modeling Language (UML) – Version 2.5. `http://www.omg.org/spec/UML/2.5/PDF` (March 2015)
9. Patil, B., Maetzel, K., Neuhold, E.J.: Universal usability issues of textual information structures, commands, and languages of native visually challenged users: An inclusive design framework. In: ICCHP'02. pp. 403–405. Springer (2002)
10. R.F Paige and J.S Ostroff and P.J Brooke: Principles for modeling language design. Information and Software Technology 42(10), 665 – 675 (2000)
11. Seifermann, S., Groenda, H.: Survey on Textual Notations for the Unified Modeling Language. In: MODELSWARD'16. pp. 20–31. SciTePress (2016)
12. W3C: Accessibility Principles - How People with Disabilities Use the Web. `https://www.w3.org/WAI/intro/people-use-web/principles` (August 2012), accessed 01/28/2016