# The Cooperate Assistive Teamwork Environment for Software Description Languages

Henning GROENDA [a], Stephan SEIFERMANN [a], Karin MÜLLER [b], and Gerhard JAWOREK [b]
{groenda, seifermann}@fzi.de {karin.e.mueller, gerhard.jaworek}@kit.edu
[a] *FZI Research Center for Information Technology, Germany*
[b] *Karlsruhe Institute of Technology, Germany*

**Abstract** Versatile description languages such as the Unified Modeling Language (UML) are commonly used in software engineering across different application domains in theory and practice. They often use graphical notations and leverage visual memory for expressing complex relations. Those notations are hard to access for people with visual impairment and impede their smooth inclusion in an engineering team. Existing approaches provide textual notations but require manual synchronization between the notations. This paper presents requirements for an accessible and language-aware team work environment as well as our plan for the assistive implementation of Cooperate. An industrial software engineering team consisting of people with and without visual impairment will evaluate the implementation.

## 1. Introduction

Description languages such as the Unified Modeling Language (UML) [13] are commonly used in software engineering to describe various aspects of systems, e.g. the structure or the behavior, in all phases of a system's life cycle. They are applied in practice and integral parts of university courses [12] and vocational trainings [20]. Their main advantage is a formal and precise specification easing unambiguous communication between experts. They ease the handling and comprehension of complex systems by providing separate diagrams for separate concerns and often use graphical notations leveraging visual memory for understandability. Those benefits lead to the introduction of description languages in further application domains, e.g. EAST-ADL [5] or AUTOSAR [1] in the automotive or SysML [14] in the systems engineering domain.

The increasing use of graphical notations impedes people with visual impairment from working in an engineering team alongside sighted people. Common assistive technologies like screen readers cannot access the notations and human assistants are required to translate and create textual notations. This delays mutual discussions in a team and is time-consuming, personnel-intensive and error-prone in case of updates.

Existing approaches provide textual notations of the graphical notations making the content accessible. One-way automated translations ease the work of personal assistants

but editing still requires error-prone manual analyses. Additionally, the approaches often only cover a small subset of a language and do not take the synchronization of textual and graphical notations into account. However, this is crucial for a seamless teamwork.

This paper discusses the identified requirements for making description languages accessible and supporting seamless teamwork in a team consisting of people with normal sight, low vision or blindness. Furthermore, it describes our plans to realize the Cooperate assistive teamwork environment fulfilling the requirements and providing state-of-the-art additional support for concurrent editing, auto-completion, and folding. Folding allows selective hiding of parts of the currently edited document.

Besides validating the requirements by discussing them at scientific conferences and with concerned parties, there will be frequent tests with people with visual impairment ensuring a short feedback cycle and good user experience. The industrial applicability and fitness is ensured by evaluating Cooperate in parallel to its development in an existing industrial team of people with and without visual impairment.

The remainder of this paper is structured as follows: Section 2 specifies the identified requirements for assistive team work environments supporting description languages. Section 3 describes our plan to realize the Cooperate assistive team work environment and how it provides solutions fulfilling the requirements. Section 4 discusses related work and existing approaches for making description languages accessible. Section 5 concludes the paper and provides an overview of future work.

## 2. Requirements for Cooperative Editing with Description Languages

This section describes the requirements we have identified for an assistive teamwork environment for description languages. We mined the posts in the Blind UML user group [2] consisting of 16 members for previous UML experiences. Additionally, we examined the personal blog of a visually impaired software engineer. We gathered general information about flaws in practical UML usage from the case study of Petre [15]. Based on our findings, we built purposeful questions and conducted three personal interviews with visually impaired computer science students and a software engineer. In addition, we asked for the UML experience in the Blind UML user group by ourselves, which lead to four responses.

We inferred the following six requirements (R1-R6), which are presented and discussed in the following paragraphs in order to foster the discussion on the completeness or further needs.

*R1: Understanding Notations* All team members must have basic knowledge about the language and its notations. Formal languages such as the UML assign semantics to all language elements. Although some restrictions are enforced by the language, semantic inaccuracies and use of *similar* language elements are often accepted during team discussions. Similarity is often regarded on the graphical notation level and does not take the defined semantics into account. Textual notations show the formal meaning. This requires that all team members know the alternative notations in order to reduce misunderstandings.

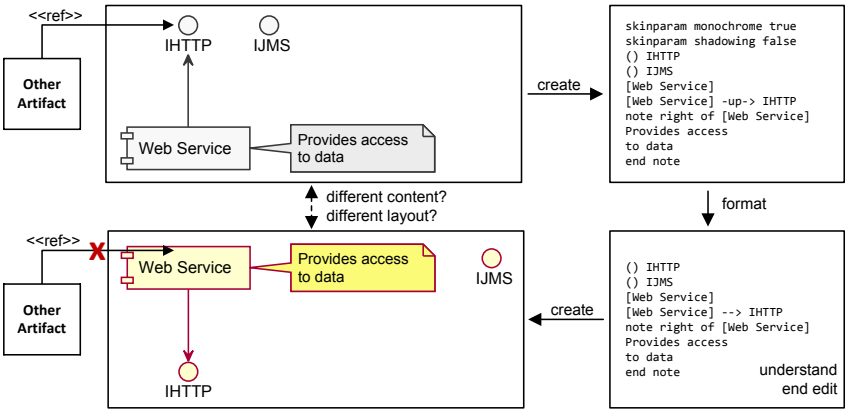


**Figure 1.** Demonstration of one-way synchronization issues using PlantUML [16]: a) layout information is lost or corrupted and b) references from other artifacts are broken.

*R2: Accessibility of Notations* Graphical notations are often used to provide an overview of complex tasks and leverage visual memory for communicating many aspects at once. Any alternative notation must be concise and allow a quick overview as well. The elements required for an overview depend on the work focus of a team member. Therefore, textual notations must be tailored to show and mask information depending on common tasks.

*R3: Editing Support* Editing models must be tailored to assistive needs. Assistive techniques support using the graphical or textual syntax correctly and human-friendly referencing of elements, e.g. by using readable names instead of long technical reference numbers. Team members additionally require customizable structuring of information depending on the notation by reordering textual or layouting grahical elements.

*R4: Consistency of Notations* The main challenge is allowing team members to edit information in their most convenient way by providing various notations and keep them consistent. A person with blindness would choose a textual notation whereas most of the sighted members will prefer a graphical notation. All members of a team must be able to modify the same underlying information while keeping it consistent throughout all notations. Modifying notations must not break references or layout information as shown in Figure 1. The time for updating a notation for one team member based on changes of another must be small enough to allow real-time collaboration. No human intervention must be necessary during non-conflicting consistency recovery ensuring a smooth and seamless environment.

*R5: Real-World Applicability* Team members require an approach usable in industrial practice. This requires support for the diagrams used in industry as well as appropriate learning material for team members with and without visual impairment.

*R6: Sustainability* There is no general concept yet on how to create accessible textual notations for modeling languages. However, this is required by tool providers and language developers in order to cater for assistive needs. A concept allows maintenance if the UML is updated and it can be transferred to different languages such as BPMN or ER as well.

| Requirements \ Solutions | S1 | S2 | S3 | S4 | S5 | S6 | S7 |
|---|---|---|---|---|---|---|---|
| R1 | ✓ | | | | | | |
| R2 | | | ✓ | ✓ | | | |
| R3 | | | | ✓ | | | |
| R4 | | | ✓ | | ✓ | | |
| R5 | ✓ | ✓ | | | | ✓ | |
| R6 | | | | | | | ✓ |

**Table 1.** Coverage of identified requirements by proposed solutions.

## 3. Implementation of the Cooperate Assistive Teamwork Environment

This section describes our plans on how to realize the Cooperate assistive teamwork environment. It provides an overview, sketches the basic concepts and provides details on the solutions addressing the requirements shown in the section above. An overview of the requirements covered by the solutions is provided in Table 1.

Figure 2 shows the basic idea of the Cooperate environment on an example using UML's structural composition notation. All team members work on tailored notations of the same information. The team member on the left is blind and works with a textual notation using screen reader, audio output, and braille technologies. The member in the middle uses a classical graphical notation. On the right hand side, a member with low vision works with a textual notation with magnification.

The UML [13] consists of overall 14 notations. Cooperate focuses on supporting the notations Activity, Class, State Machine, and Use Case. Activity and State Machine notations are often used to the define behavior during development. Class notations are common in data descriptions and allow describing entities and their relations. Use Case, Activity, Class, and State Machine notations can be used in requirements elicitation. The implementation of the Cooperate environment will be made publicly available as Open Source at Github [4] allowing its free use and further extension.

*S1: Teaching Material on Description Languages* Solution 1 addresses R1 and R5. Learning material ensures basic knowledge and addresses communication in diversity teams. First, an introduction on UML teaches the basic concepts, advantages and why it is commonly used. The second part of the teaching material covers the various types of UML charts and their graphical notation. Tactile versions of the UML charts give people with visual impairment an overview of the visual concepts. The third part introduces the textual notation for people with and without visual challenges ensuring easy communication within teams.

*S2: Tutorial for the Cooperate Environment* Solution 2 addresses R5 and is based on S1. It introduces the Cooperate environment for people with and without visual impairments. Part 1 shows the accessible usage of the Eclipse IDE, which is used by Cooperate. Part 2 shows the available assistive technologies and how they can be used. Part 3 provides guidelines for the communication and collaboration within diversity teams addressing specific needs and potential pitfalls.

*S3: Textual Notation* Solution 3 addresses R2 and R4. Existing textual notations are analyzed for their advantages and drawbacks. Common errors such as including layouting

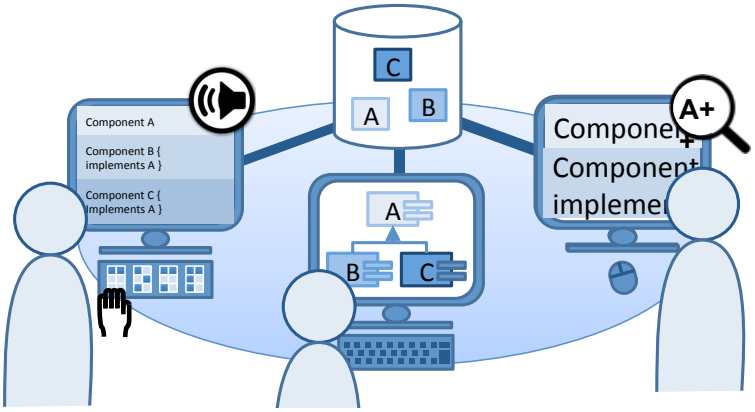


**Figure 2.** Concept of Cooperate with tailored support for each team member working on the same underlying information.

information are omitted. The resulting concise notation is defined and ensures a bijective mapping between textual and graphical elements.

*S4: Assistive Editor Support for Textual Notations* Solution 4 addresses R2 and R3. The editor for textual notations provides all features known from state-of-the-art graphical and code editors: An outline allows easier navigation. Shortcuts simplify navigation between elements, e.g. jump to the next element or to the definition of a referenced element. Folding of hierarchical elements reduces the number of visible elements and eases navigation. Auto-completion techniques provide easier creation and correct modification of elements. Accessible (audio) signals report errors and inconsistencies directly to the editing team member. This fast feedback loop increases locality and productivity. Textual identifiers targeted at humans represent references to complex elements. The editor ensures consistency of identifiers even across documents. The Open Source editor [4] leverages and will be realized based on the Eclipse Modeling IDE [19].

*S5: Real-Time Synchronization of Notations* Solution 5 addresses R4. A real-time synchronization of textual and graphical notations provides consistency between them. A UML model represented in a specific notation can be edited as usual using graphical or textual editors (see S3). The synchronization between the notations is a background task and therefore fully transparent to the user. It adjusts the shown notations rather than replacing them. Therefore, layout and structural information are not lost after an update. No human intervention is necessary during non-conflicting consistency recovery. Processing will be fast as the changes processed are small.

*S6: Integration of Enterprise Architect* Solution 6 addresses R5. Enterprise Architect[1] (EA) is a UML modeling tool developed by SparxSystems commonly used in industrial contexts. A proof-of-concept integration with this commercial product reduces the time to learn new tools and lowers the entry barrier. Single team members can use the Cooperate environment while others stick with their existing solution. This eases the evaluation effort for companies and reduces the barrier for visually impaired people to join teams.

[1] `http://www.sparxsystems.eu/enterprisearchitect/`

*S7: Guidelines for Assistive Tool Developers* Solution 7 addresses R6. We will publish guidelines and best practices based on our experiences and identified pros and cons of existing notations and approaches. Frequent accessibility barriers are pointed out. This eases the design of accessible textual notations and the implementation of accessibility features in editors. The Cooperate environment showcases accessibility solutions easing the adoption in further tools.

## 4. Related Work

Common description languages such as the UML provide specifications for graphical notations but do not focus on alternatives. Those are mainly textual and haptic notations. Existing approaches for those areas are shown in the following subsections.

### *4.1. Textual Notations*

There have been several approaches to create textual notations. Only some of them incorporate aspects about accessibility for people with visual impairment. Their notations are largely incompatible. Mazanec et. al. [10, Section 3] created a survey based on requirements such as coverage of the UML, readability and simplicity. In the following paragraphs, we review all of the identified approaches plus UML Graph with respect to their usability for people with visual impairment and on collaborative editing.

PlantUML [16] is a commonly used notation for creating diagrams based on textual specifications. It is integrated in various collaboration platforms such as the MediaWiki and supports the most common UML notations. As [10] points out, the notation is cluttered with layout information due to its heritage. It tries to express the visual representation with characters similar to ASCII graphics. Reading an ASCII graphic line by line with a Braille display is very time consuming and needs a lot of concentration. Another problem is that there is only a one-way creation of graphical representations but no synchronization. Despite this, it has been suggested as interim solution for accessible UML diagrams by the BlindUML initiative [2].

TextUML [3] is a textual notation for class, state and component notations similar to a programming language. It is intuitive for programmers but enforces users to provide information that is usually omitted in early life cycle stage such as the types of attributes. Umple [8] (UML class and state machine notations) and UML Graph [18] (UML class and sequence notations) are similar to TextUML but allow omitting information. Layout information can be omitted, but automatic layouting prohibits customization of the representation. Despite of [8] the approaches do not provide a synchronization concept.

yUML [6] was designed for small use case and class notations as described in the FAQ section of its homepage. It does not provide editor support. Like PlantUML, it memes graphical counterparts with characters. According to [10, Section 3] it is not readable anymore for complex diagrams, although usage in industrial context requires this. Thereby, people with visual impairment are excluded from real projects.

The Earl Grey [10] notation focusses on class, sequence, and state notations. It is designed for usability but has not been evaluated by people with visual impairment yet. We consider it a good starting point for a tailored notation. Earl Grey does not address synchronization.

The Audio Programming Language [17] focuses on writing simplified Java code, which is also a textual notation. The corresponding editor provides auto-completion, context-sensitive menus, and extensive audible feedback. The approach provides tested features for accessible teamwork environments. Relevant features will be included in Cooperate.

### *4.2. Haptic Notations*

Besides from textual notations, there are various other approaches to make information accessible for people with visual impairment.

The Technical Drawings Understandable for the Blind (TeDuB) [7] approach makes a hierarchical representation of the UML class, use case, sequence, and state notations accessible via joystick navigation. It focusses on comprehension and does not support editing. Unfortunately, users have to use a default hierarchical representation.

Metatla, et.al. [11] use haptic devices and hierarchical representations to make notations accessible. Their approach supports the UML class notation and can be extended to support more. It allows collaborative editing with automatic change propagation and logging to review recent changes. We only tested the graphical notation because the hierarchical notation was not available in the provided tool. It remains unclear how well the synchronization between different notations works.

Tactile graphics make visual information accessible. They can be printed for instance on swell paper, embossed on paper, or presented on special tactile displays. Loitsch and Weber evaluate the readability of UML sequence notations in [9]. They conclude that only a limited amount of data can be incorporated in a tactile graphics because of resolution and size constraints. Editing of tactile graphics and the UML model behind it is not possible. Tactile graphics ease teaching the visual notation but are not suitable for collaboration.

## 5. Conclusion

The requirements in Section 2 showed the prerequisites for equal collaboration of people with visual impairment within a software developing team. The specific usability features for description languages were pointed out. Synchronization is the most urgent issue enabling seamless collaboration. The requirements for real world applicability and sustainable transfer to different contexts was shown. The solutions described in Section 3 presented how those requirements are addressed as part of the Cooperate assistive teamwork environment. Cooperate will allow seamless collaboration for all team members regardless if they use textual or graphical notations. The discussion of related work showed that widespread textual notations such as PlantUML reduce comprehensibility by graphical layout information and currently lack synchronization mechanism. Most approaches lack navigation and advanced editing capabilities like auto-completion and error notifications. There is limited support for structuring although it is required to tackle the complexity for real-world cases. Artificial hierarchical structures are superimposed for haptic notations reducing complexity but cannot be used as variably like folding for tailoring the representation for specific tasks. Tactile paper fosters understanding of graphical notations but cannot be applied for realistic use cases within collaborative teams yet.

Our first step is to close requirement elicitation for Cooperate in the short-term. Then, we will validate the presented solutions with stakeholders and in parallel within a project of a professional team.

## Acknowledgments

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

## References

[1] AUTOSAR Development Cooperation. Autosar – automotive open system architecture 4.2. http://www.autosar.org, 2014. [accessed 2015-02-20].

[2] BlindUML Initiative. Blinduml user group. https://groups.yahoo.com/neo/groups/blinduml/info/, 2015. [accessed 2015-05-28].

[3] R. Chaves. Textuml toolkit. http://abstratt.github.io/textuml, 2015. [accessed 2015-02-20].

[4] Cooperate-Project. Cooperate-project on github. https://github.com/Cooperate-Project, 2015. [accessed 2015-02-20].

[5] EAST-ADL Association. East-adl domain model specification v2.1.12. http://www.east-adl.info, 2013. [accessed 2015-02-20].

[6] T. Harris. Create uml diagrams online in seconds, no special tools needed. http://yuml.me, 2015. [accessed 2015-02-20].

[7] M. Horstmann, C. Hagen, A. King, S. Dijkstra, D. Crombie, D. G. Evans, G. T. Ioannidis, P. Blenkhorn, O. Herzog, and C. Schlieder. Tedub: Automatic interpretation and presentation of technical diagrams for blind people. In *Proceedings of CVHI'04*, 2004.

[8] T. Lethbridge. Teaching modeling using umple: Principles for the development of an effective tool. In *Proceedings of CSEET'14*, pages 23–28, April 2014.

[9] C. Loitsch and G. Weber. Viable haptic uml for blind people. In *Proceedings of ICCHP'12*, pages 509–516. Springer Berlin Heidelberg, 2012.

[10] M. Mazanec and O. Macek. On general-purpose textual modeling languages. In *Proceedings of DATESO'12*, pages 1–12. CEUR-WS.org, 2012.

[11] O. Metatla, N. Bryan-Kinns, T. Stockman, and F. Martin. Cross-modal collaborative interaction between visually-impaired and sighted users in the workplace. In *Proceedings of ICAD 2012*, pages 164–171. Georgia Institute of Technology, 2012.

[12] K. Müller. How to make unified modeling language diagrams accessible for blind students. In *Proceedings of ICCHP'12*, pages 186–190. Springer-Verlag, 2012.

[13] Object Management Group. Omg unified modeling language 2.4.1. http://www.omg.org/spec/UML/2.4.1/, 2011. [accessed 2015-02-20].

[14] Object Management Group. Omg systems modeling language standard 1.3. http://www.omg.org/spec/SysML/1.3/PDF/, 2012. [accessed 2015-02-20].

[15] M. Petre. Uml in practice. In *Proceedings of ICSE'13*, pages 722–731. IEEE Press, 2013.

[16] PlantUML. Plantuml. http://plantuml.sourceforge.net, 2015. [accessed 2015-02-20].

[17] J. Sánchez and F. Aguayo. APL: audio programming language for blind learners. In *Proceedings of ICCHP'16*, pages 1334–1341, 2006.

[18] D. Spinellis. On the declarative specification of models. *Software, IEEE*, 20(2):96–95, March 2003.

[19] The Eclipse Foundation. Eclipse modeling project. http://eclipse.org/modeling/, 2015. [accessed 2015-02-20].

[20] Zentrale Prüfungserstellungs-Stelle (ZPA) . *Fachinformatiker/-in – Anwendungsentwicklung Prüfungskatalog für die IHK-Abschlussprüfung*. U-Form Verlag, Solingen, 8 edition, 2012.