This paper will be presented at MODELSWARD 2016 (http://www.modelsward.org/)
The published paper is available via https://doi.org/10.5220/0005644900280039

Survey on Textual Notations for the Unified Modeling Language

Stephan Seifermann[1] and Henning Groenda[1]
[1]*Software Engineering, FZI Research Center for Information Technology, Haid-und-Neu-Str. 10-14, Karlsruhe, Germany*
{*seifermann, groenda*}*@fzi.de*

Keywords: UML, Textual Notation, Survey, Editing Experience

Abstract: The Unified Modeling Language (UML) has become the lingua franca of software description languages. Textual notations of UML are also accessible for visually impaired people and allow a more developer-oriented and compact presentation. There are many textual notations that largely differ in their syntax, coverage of the UML, user editing experience, and applicability in teams due to the lack of a standardized textual notation. The available surveys do not cover the academic state of the art, the editing experience and applicability in teams. This implies heavy effort for evaluating and selecting notations. This survey identifies textual notations for UML that can be used instead of or in combination with graphical notations, e.g. by collaborating teams or in different contexts. We identified and rated the current state of 16 known notations plus 15 notations that were not covered in previous surveys. 20 categories cover the applicability in engineering teams. No single editable textual notation has full UML coverage. The mean coverage is 2.7 diagram types and editing support varies between none and 7 out of 9 categories. The survey facilitates the otherwise unclear notation selection and can reduce selection effort.

1 Introduction

The Unified Modeling Language (UML) has become the lingua franca of software description languages. The UML specification comes with a graphical notation, which is commonly applied. There is no standardized textual notation. According to Spinellis (Spinellis, 2003), textual notations provide an alternative representation. It can be more compact or more intuitive for certain user groups than the graphical representation. An example is the textual representation of an UML activity diagram-like specification of a service's behavior (Erb, 2011). The textual version requires significantly less space for the representation and is more intuitive to developers.

There are several tailored textual notations tailored for specific purposes such as creating documentation integrated in the code, having a more compact representation, or increasing accessibility. They largely differ in their syntax, coverage of the UML, user editing experience, and applicability in an engineering teams.

The latest surveys covering textual UML notations were performed by Luque, et al. (Luque et al., 2014a; Luque et al., 2014b). The former (Luque et al., 2014a) focuses on the accessibility of UML for blind students in classrooms and the latter (Luque et al., 2014b) on tools for use-case and class diagrams used in industry at 20 companies in the state of Sao Paolo (Brazil). Both surveys target notations used in practice without regarding the state of the art described in scientific publications. (Mazanec and Macek, 2012) focuses on textual notations in general but is a few years old and covers few notations. It does not represent the current development state and available variety of the notations and modeling environments. The three existing surveys show that there is a wide range available with individual advantages and drawbacks. They do not analyze the user's editing experience, which is crucial for using a notation in a collaborating engineering team, and requires heavy analyses effort. The overall effort reduces the chance for a comprehensive analysis and can endanger the selections's quality.

The contribution of this survey is the identification and classification of textual UML notations including the user experience. Engineering teams can use the classification for identifying appropriate notations for their usage scenarios. The classification scheme is tailored to support this selection. This survey examines accessibility of notations with respect to their syntax, editors, and modeling environment. Usability in realistic scenarios is determined by covered diagram types, data format exchanges, and synchronization approaches with other notations. It additionally evaluates whether non-necessary parts of the notation can be omitted. This sketch support eases lowoverhead discussion and brainstorming. For instance, the UML specification allows to omit the types of the class attributes.

The survey identified and rated 31 notations of which 15 were not covered in previous surveys. 20 categories allow fine-grained pre-selection and cover the applicability in engineering teams.

The remainder of this survey is structured as follows: Section 2 describes the review method of the survey by defining the objectives and the review protocol consisting of three phases. Section 3 describes the classification scheme based on the defined objectives. Section 4 presents the analysis results in terms of classified textual notations. Section 5 discusses the validity of the results and the findings. Finally, Section 6 concludes the paper.

2 Review Method

The review process follows the guidelines of Kitchenham and Charters (Kitchenham and Charters, 2007). They developed guidelines for structured literature reviews (SLR) in software engineering based on the guidelines in the field of medical research. Their guidelines cover the planning, conduction, and writing of reviews. Planning involves defining research objectives and creating a review protocol describing the activities in each step during the review conduction.

The following sections describe our implementation of the SLR and mapping to the proposed method. The results of our search activities are documented and available for reproducibility at `http://cooperate-project.de/modelsward2016`.

2.1 Objectives

Our objectives are to determine each notation's (O1) coverage of the UML, (O2) user editing experience and (O3) applicability in an engineering team. The reasoning requires an analysis of the textual notations and of the modeling environments. Section 3 presents the detailed classification scheme based on the objectives and instructions on the according information extraction procedure from literature.

2.2 Review Protocol

Figure 1 shows an overview of our review protocol. We distinguish three phases during the conduction: classic SLR, Quality Assurance and Complement.

The classic *SLR* follows the guidelines of review conduction by Kitchenham, et al. (Kitchenham and Charters, 2007). We extend the SLR with two additional phases in order to increase the quality of the results and to take notations into account that are mainly used in (industrial) practice: The *Quality Assurance* phase focuses on incoming and outgoing literature references as suggested by the Snowballing search approach (Wohlin, 2014). In contrast to the original proposal, we use Snowballing only to cross-check our SLR search strategy. The *Complement* phase focuses on textual notations that are available in practice but are not scientifically published.

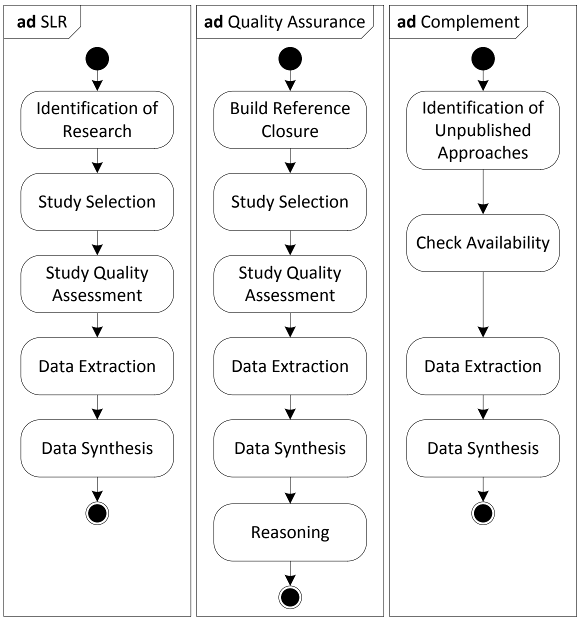

Figure 1: The three phases of the review conduction process used in this survey.

2.3 Phase 1: SLR

The review conduction according to (Kitchenham and Charters, 2007) consists of the five activities marked as *SLR* in Figure 1.

The *Identification of Research* describes the search strategy for collecting literature. We chose a keyword-based search approach using the search engines ACM Digital Library, IEEExplorer, CiteSeer, ScienceDirect, SpringerLink and Google Scholar. These search engines cover relevant journals and are suggested by Kitchenham and Charters for the software engineering domain. We did not include EI Compendex and Inspec as we could not query these search engines without subscriptions. Their focus is on high-qualitative entries and metadata and they do not belong to a not-covered established publishing authority. We are confident that the selected search engines are sufficient and that no relevant paper is

Group	Keywords
Textual T	CTS, textual modeling, textual modelling, text-based modeling, text-based modelling, textual notation, text-based notation, textual UML, text-based UML, textual syntax
UML U	UML, unified modeling language, unified modelling language

Table 1: Keyword groups used in search queries.

not selected by our keywords because of insufficient metadata provided by the selected engines.

We defined a set of keywords T for identifying textual notations and another one U for identifying the usage of UML. Both sets are shown in Table 1 and are variations of our original terms *textual notation*, and *UML*. They are based on commonly used terminology in the modeling domain. A search query is given by $(t_0 \vee t_1 \vee \ldots \vee t_9) \wedge (u_0 \vee u_1 \vee u_2)$ with $t_i \in T \wedge u_i \in U$. The query enforces the exact matching of keywords. It considers abstracts and titles because this restricts the search to literature that focuses on textual notations for UML. Google Scholar has API restrictions that limit queries on abstracts to papers that have been released at most one year ago. This restriction does not apply to our title-based search. We restrict ScienceDirect queries to computer science papers. We implemented a search on the SpringLink results enabling keyword identification in the abstract. After collecting the results of all search engines, we merge them and filter duplicates.

Study Selection covers a rough screening based on titles and abstracts to allow spending more time on relevant literature. We focus on textual notations for graphical parts of the UML notation, which are given in the UML specification (OMG, 2015a, p. 683). We exclude all textual notations only extending UML or its elements rather than expressing UML itself. We exclude all notations that are not related to UML. We exclude notations not intended for usage by humans such as data transfer containers, e.g. XMI serialization (OMG, 2015b). We include a) primary papers describing a single textual notation, and b) secondary survey-like papers including their references as primary sources.

The *Study Quality Assessment* considers title, abstract, and the content of the full paper. We decide on in-/exclusion of the remaining papers in this step.

Data Extraction is the process of determining the information required to judge about the fulfillment of the objectives. Section 3 shows the analyzed features of the notations, their hierarchy, and individual decision basis in detail. We reason on the modeling environment based on information found directly in literature, implemented prototypes, prototype websites, and source code. We identify prototypes, their website, and the source code by: a) following links in the papers, b) mining the website of the institute or company of the authors, c) and searching for the name of the notation (full name and abbreviation if used) via the Google search engine and on Githuband visit the first one hundred search results. Data extraction takes place for the declared primary editor. If there is more than one prototype, we use the declared primary editor and an IDE-integrated editor. We assume the latter to profit from advanced accessibility features of the IDE. If there are editors for several IDEs, we decide in favor of the Eclipse-based one because Eclipse is open source, highly extensible, and offers many accessibility features[1].

Data Synthesis summarizes the information. We show and summarize the analysis results according to the classification given in Section 3.

2.4 Phase 2: Quality Assurance

The Quality Assurance phase is based on the Snowballing approach (Wohlin, 2014) of Wohlin. Wohlin suggests starting with an initial set of relevant literature and including relevant forward and backward references. Afterwards, the resulting set of literature is processed as described in common SLR processes. We do not use Snowballing as primary source for relevant literature because its quality heavily depends on the initial literature set as described by Wohlin. Instead, we accept the overhead of a prior SLR phase with broad search terms and use Snowballing to verify the quality of our SLR phase as described below.

Build Reference Closure determines the completeness of results from the SLR phase. We collect all directly referenced and referencing literature for the analyzed papers. We derive the referenced literature from the references section of the paper. We use Google Scholar to determine the literature that cites the paper under investigation.

The *Study Selection* and *Study Quality Assessment* from phase SLR are applied to identify additional notations.

We perform *Data Extraction* on selected papers as in the SLR phase and add the notation to our database.

Data Synthesis summarizes the information as carried out in the SLR phase.

Reasoning addresses why the newly identified notations have not been found in the SLR phase. The results are presented in Section 5. This phase is dif-

[1] `https://wiki.eclipse.org/Accessibility`

ferent from the Snowballing approach of Wohlin and allows us to verify the quality of our SLR phase.

2.5 Phase 3: Complement

Identification of Unpublished Approaches focuses on textual notations that are available in practice but are not scientifically published. We use the Google search engine to identify the top 5 pages for 'UML textual notation', 'UML textual notations', 'UML textual notations list'. We mine the resulting websites to identify new approaches. We follow the links from the identified websites looking for notations or comparisons of notations.

Additionally, we search for unrecognized scientific surveys or notation comparisons. We perform a full-text search via Google Scholar with the names of the three most popular non-scientific notations. We assume that recent surveys including non-scientific notations cover them and thereby will be included in the search results. We determine a notation's popularity by querying Google with the name of the notation and comparing the announced results with the amount of other notations. We only included notations that do claim to relate to the UML.

In *Check Availability*, we exclude potential notations with dead links for all results.

We perform *Data Extraction* for new notations, analyze the information, and add the notation to our database.

Data Synthesis summarizes the information as carried out in the SLR phase.

3 Classification

This section presents the classification and information extraction goals derived from the three objectives presented in Section 2.1. The objectives cover aspects of what can be edited based on the textual notation definition (O1, O2) as well as how it can be edited based on modeling environments (O2, O3). We use feature modeling to represent the evaluation classes, their hierarchy, and possible values. The resulting overview is depicted in Figure 2. The features themselves and how their values are evaluated for the notations are presented in the following.

Each *Textual Notation* is defined by a Language (O1, O2) and an optional Implementation (O2, O3) in a modeling environment.

The *Implementation* is optional and covers all aspects with respect to a modeling environment for a notation. It can have Recent Activity (O3), a License (O3), and can support Change Propagation (O2, O3) between different notations, data Format Exchange (O2), and Editor (O2) features.

We divide the classification of the implementation into two parts for a better overview: integration aspects, and the editor itself. The former covers the features relevant for integrating an implementation into a tool chain. The latter covers the editing experience of the editors.

The following subsections will cover the language, integration, and the editor in that order.

3.1 Language

The *Language* definition is mandatory and describes the language's syntax and semantics. It consists of the UML Support (O1) for diagram types and can have Sketch Support (O2), integrated Layout Information (O2), and be Similar to UML Graphics (O2).

UML Support is mandatory and describes the supported UML diagram types. At least one type has to be supported. A type is supported if the documentation states it to be supported or the modeling environment allows the creation of a corresponding type. The considered diagram types are based on the UML specification (OMG, 2015a, p. 682). The abbreviations are based on the official abbreviations from (OMG, 2015a, p. 682), or self-made if there is no official one: Activity Diagram (ACT), Class Diagram (CLS), Communication Diagram (COM), Component Diagram (CMP), Composite Structure Diagram (COS), Deployment Diagram (DEP), Interaction Overview Diagram (INT), Object Diagram (OBJ), Package Diagram (PKG), Profile Diagram (PRO), Sequence Diagram (SEQ), State Machine Diagram (STM), Timing Diagram (TIM), and Use Case Diagram (UC).

Sketch Support is optional and can ease the notation's usage during discussions. Discussions benefit from quick interaction and formal full-fledged modeling can extend the interaction time. There is support if only mandatory elements of UML's abstract syntax are required.

Layout Information is optional states if the textual model can contain graphical layout information. This information allows to improve graphical presentation of the textual statements. The information is irrelevant to describe the model itself. The interpretation is difficult as graphical coordinates or positions are only visible in graphic notations. The information can be either *Mixed* with model elements or kept *Separated.* It is marked as *Mixed* if at least one element has mandatory layout information.

Similar to UML Graphics is optional and denotes if graphical syntax elements such as arrows are replicated in the textual notation by ASCII art. For in-

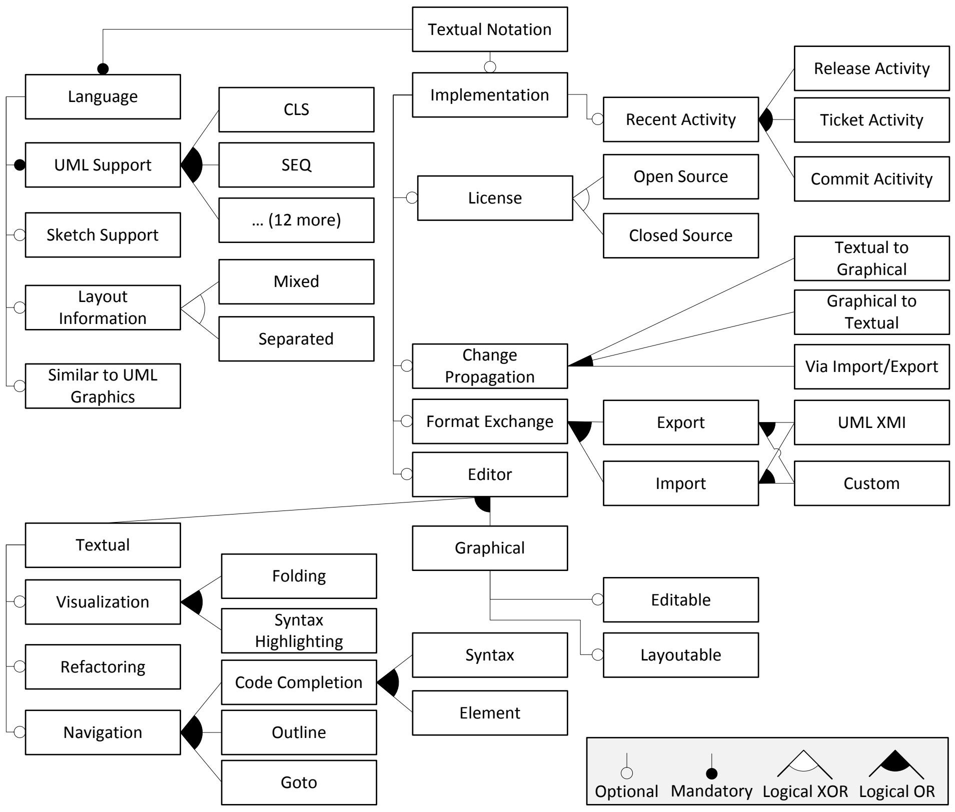

Figure 2: Feature model for analyzed characteristics and their hierarchy.

stance, the characters <>--> are similar to the UML graphical representation for an aggregation. This can work well for people knowing the graphical representation but have adverse effects on people using accessibility tools like Braille displays. A notation is marked as similar if there is at least one ASCII art mapping.

3.2 Integration

The integration covers all features that are relevant for integrating an implementation into a tool chain. Such a decision is based on the costs, extensibility, support, maintainability and compatibility to existing tools. The following features cover these aspects in more detail.

The *Recent Activity* is optional and indicates the support status. In contrast to a maintained project, a discontinued project will not receive bugfixes and might be incompatible to recent software such as new versions of an IDE. We determine three activity dates that allow judging project activity. One of them has to be identifiable: *Release Activity* relates to the date of the last release. A release can be a proper release, snapshot, or nightly build. *Ticket Activity* is determined by the date of the most recently closed ticket. *Commit Activity* is given if we can determine the most recent commit.

The *License* is optional and can be crucial for using and maintaining the modeling environment. Open Source licenses allow own bug fixing and the development of extensions and adaptations. The individual requirements for a license depend heavily on the usage context of the modeling environment. An expert review is required to check for a notation of interest if it applies to the own use case. We therefore differentiate solely between Open Source and Closed Source licenses. We rely on the list of the Open Source Initiative (Open Source Initiative, 2015). If the license is listed on their website, we treat the project as open source. All other licenses are considered *Closed Source*.

Change Propagation can be supported and addresses transferring changes from one notation into another. The modification in the modeling environment for a textual notation can therefore result in an according change in a graphical notation of the same content. This targets a consistent view of the content and allows different team members to work with different notations during discussion. This can mean updates in real-time for close collaboration or based on exporting and importing models in different environments. We consider the three cases: Textual to Graphical, Graphical to Textual, and Via Import/Export propagation. *Textual to Graphical* and *Graphical to Textual* apply if the modeling environment includes a textual and a graphical editor. We consider it supported if changes in one editor are reflected in the other one. *Via Import/Export* applies if there is an import or export functionality and notations can be updated sequentially. It is marked if it provides import and export function for UML models in the standardized XMI data format.

Data *Format Exchange* is optional and allows integrating the modeling results into other tools or existing tool chains. We only consider fully-automated exchange procedures provided by the implementation itself. We do not consider other procedures such as the error-prone manual translation between notations or tools that is usually done by assistants. A modeling environment can support the *Import* or *Export* of a different set of data formats. This feature can have the value *UML XMI* as standardized UML data exchange and can list *Custom* formats supported by the tools. The values are selected based on the documentation or file extensions provided in the editing environment.

3.3 Editor

Editor categorizes properties related to user input, interaction, and presentation. They can be Textual (O2), Graphical (O2) or both. An editor is considered textual if it contains only text and no graphical elements. Text coloring may be used. This ensures that textual editors are accessible by accessibility techniques such as screen readers. Otherwise, it is treated as Graphical.

Textual editors address several features to increase user experience and accessibility. A textual editor can support Visualization (O2), Refactoring (O2) of the model, and user Navigation (O2) within the model. Previous surveys did not focus on the editing experience in detail. Therefore, we selected the features according to our objectives.

A *Visualization* is optional and allows focused presentation of content by means of information hiding. It can support Folding (O2), and Syntax Highlighting (O2).

Folding (un)hides selected partitions of the model, eases comprehension for complex models and focused presentation. It is selected if there is at least one partition in a model that can be hidden or shown based on the editor's UI.

Syntax Highlighting highlights keywords or important structural parts of the model. It eases comprehension and identifying the structure of models. It is selected if colors or text formatting is used in the editor to highlight at least one keyword of the language.

Refactoring is optional and addresses batch changes to the model. For instance, all occurrences of a model element can be replaced with another one in one single step instead of using a manual search and replace approach. This feature exists if there is at least one supported refactoring.

Navigation is optional and addresses the navigation to model elements and providing an overview to users. There can be support for Code Completion (O2), overviews on model element by Outline (O2), and direct model element navigation by Goto (O2). Navigation is selected if there is at least one of its child features selected.

Code Completion is optional and provides completion of a language's syntax or referenced model elements. It can provide hints on keywords of the Syntax or model Elements allowed at the current position. It aids users in specifying correct models and speeds up changes. We consider two types of values: *Syntax*-based and *Element*-based completion. They are selected if there is at least one corresponding code completion feature in the editor.

Outline is optional and provides an overview of the elements in a model. This can include their hierarchical structure. It is selected if there is at least a list of all top-level elements in a model depicted in the editor.

Goto is optional and allows direct navigation or jumps to specific model elements. This eases comprehension and look-up of elements. It is selected if there is a navigation or jump support for a least one type of element in the editor. It is included if it is directly in the textual notation and excluded if its only in the Outline.

Graphical editors are optional and allow displaying and editing graphical version of the models. There are many advanced graphical UML editors available based on the formal UML specification. (Kern, 2014) gives a good overview in his survey of interoperability of UML tools. (Wikipedia, 2015) illustrates the features of various UML tools. There are many comparisons between few selected tools such as IBM Rational Software Architect, MagicDraw, and Papyrus in (Safdar et al., 2015) or between Rational Rose, ArgoUML, MagicDraw, and Enterprise Architect in (Khaled, 2009). This survey focuses on the synchronization aspect with textual languages and their editors (O3). Our categories show if the editor is mainly a pure static presentation of the model or allows interactions. We distinguish for Graphical editors if their content is Editable (O2) and Persistable (O2). This feature is selected if there is a graphical presentation of the model in the modeling environment.

Editable is optional and denotes if the graphical content can be modified, e.g. a user can rename elements. This feature is selected if at least some elements in the graphical editor can be modified.

Layoutable is optional and denotes if modifications to the graphical layout, e.g. the position of model elements, can be done. Users can structure the graphical representation in this way. This feature is selected if elements can be moved.

4 Analysis Results

This chapter presents the analysis results for all notations. Table 2 provides an overview and shows the determined characteristics for all notations. The following paragraphs provide a short description of the notations. They point out features or provide comments, which are not already covered by the overview.

Alf (OMG, 2013) has been specified by the OMG and is the action language for UML. It is based on Foundational UML (fUML). There is no official editor implementation.

Alloy (Jackson, 2002) is a model finder and solver based on the Z notation (ISO/IEC 13568:2002, 2002) instead of UML. The author compares it to UML in sections 4.1 and 6.4 and states that "Alloy is similar to OCL, the Object *Constraint* Language (OCL) of UML"[2]. It provides a graphical and textual notation but does not support any UML diagrams. It is licensed under the MIT license and does not provide access to source code.

AUML (Winikoff, 2005) is an extension to UML SEQ diagrams. Winikoff defined a textual notation for AUML that has been included in the Prometheus Design Tool[3]. It provides a PNG export but does not provide a mechanism to import or export a model.

Clafer (Zayan, 2012) is a modeling language for CLS diagrams and constraints. It can be used for feature modeling. The online tool[4] provides no graphical view but offers a GraphViz export.

DCharts (Feng, 2004) specifies a meta-model in AToM[3] [5] and a graphical and textual notation. The textual notation is the leading one and the graphical implemented only partially (Feng, 2004, p. 35). No tool or files could be found actually implementing the theoretical concept. We could not find an advanced textual editor with collaboration features for the self-defined language. The publication claims that there is a transformation from the meta-model to UML state charts.

Earl Grey (Mazanec and Macek, 2012) is a proof of concept for an accessible textual notation. The Eclipse implementation creates a model during editing but there is no export to other notations.

HUTN (Vieritz et al., 2014) is an OMG standard for text-based representation of MOF-based meta-models, which covers the UML meta-model. Humans can use it easier than XMI. There is no official reference implementation of an HUTN-based UML editor.

IOM/T (Doi et al., 2004) allows specifying protocols for agent communication. It covers AUML (Winikoff, 2005) sequence diagrams partially, which we consider as SEQ support. The notation seems to consist of two papers, the latest in 2007.

MetaUML (Gheorghies, 2015) is a DSL leveraging TeX in the background. It creates graphics in UML style but no UML models.

modsl[6] is a text to diagram sketch tool based on Java code specifications. The proposed default editing environment is Eclipse. It creates graphics in UML style but no UML models.

pgf-umlcd[7] and pgf-umlsd[8] are both based on PGF/TikZ. They leverage TEX interpreters. This has a major influence on its syntax and structure. They create graphics in UML style but no UML models.

PlantUML (Roques, 2015) is a textual notation to diagram tool. CLS diagrams can be exported as UML files for the StarUML and ArgoUML tools. Imports and synchronization mechanisms are not available. There are various standalone and integrated editor implementations.

Quick Sequence Diagram Editor[9] is a text to diagram sketch tool written in Java. It creates graphics in UML style but no UML models.

TCD (Washizaki et al., 2010) is an ASCII-art converter for CLS diagrams. It provides a conversion from and to the UML XMI representation. The implementation is not available.

TextUML (Chaves, 2015) exports standard UML models but does not provide a graphical view. Services such as Cloudfier[10] use it as alternative for graphical modeling tools.

tUML (Jouault and Delatour, 2014) focusses on modeling for validation and verification purposes.

[2]http://alloy.mit.edu/alloy/faq.html

[3]https://sites.google.com/site/rmitagents/software/prometheusPDT

[4]http://t3-necsis.cs.uwaterloo.ca:8094

[5]http://atom3.cs.mcgill.ca/

[6]https://code.google.com/p/modsl/

[7]https://github.com/xuyuan/pgf-umlcd

[8]https://code.google.com/p/pgf-umlsd

[9]http://sdedit.sourceforge.net/

[10]http://doc.cloudfier.com/creating/language/

The mentioned prototype is not available.

txtUML (Dévai et al., 2014) uses regular Java syntax for modeling. Java Annotations provide additional information. There is no dedicated textual or graphical editor but a Papyrus model can be exported.

UML/P (Grönniger et al., 2014) is a textual notation claiming to merge programming and modeling by enriching UML models with Java expressions. The Eclipse plugin provides textual and graphical editors but no import or export.

UMLet[11] (Auer et al., 2003) is a graphical UML sketch tool. It provides graphical UML shapes. A selected shape is shown in a textual view, which allows to modify the element. The textual view covers only the selected element. It create graphics in UML style but no UML models.

UMLGraph (Spinellis, 2003) uses Java source files and customized JavaDoc comments to create diagrams. It creates graphics in UML style but no UML models.

uml-sequence-diagram-dsl-txl[12] is a command-line based text to diagram sketch tool written in the transformation language TXL. The Eclipse IDE plug-in was not available. The table lists the mentioned features of the guide[13]. It creates graphics in UML style but no UML models.

Umple (Lethbridge, 2014) is a model-to-code generator with textual notations. UML elements not relevant for code generation such as aggregations are omitted. The online tool synchronizes the textual and graphical notation.

USE (Zayan, 2012) aims for specifying systems with including OCL constraints. The official tool does not provide an editor but textual and graphical views.

AWMo (Del Nero Grillo and de Mattos Fortes, 2014)[14] is a Web application targeting the collaboration of blind and sighted users. The Web tool does not work, there is no included documentation. The characteristics have been determined based on the source code and available presentations and the paper. They define their own simplistic meta-model inspired by CLS diagrams for their proof of concept. Collaboration is realized via store and load mechanism, which maps to Import and Export in the table.

blockdiag[15] has the subprojects seqdiag[16] and actdiag[17]. Both are written in Python and convert textual diagram descriptions to graphics. The syntax is Graphviz's DOT format. The code and release activities are taken from seqdiag only being representative. It creates graphics in UML style but no UML models.

Finite State Machine Diagram Editor and Source Code Generator[18] has an own XML Schema Definition, which defines their textual language called FsmML. Conforming XML documents can be Imported and Exported. Links to model elements are realized via String matching.

js-sequence-diagrams[19] is a text to diagram sketch tool written in Java Script. It is inspired by the commercial WebSequenceDiagram. It parses plain text and can report basic parsing errors. Its shared with an own license title as simplified BSD. It creates graphics in UML style but no UML models.

nomnoml[20] is a text to diagram sketch tool written in Java Script. The syntax is oriented at the graphical UML shapes. It creates graphics in UML style but no UML models.

WebSequenceDiagrams[21] is a text to diagram sketch tool written in Java Script. It creates graphics in UML style but no UML models. A free alternative is js-sequence-diagrams.

yUML (Harris, 2015) is a text to diagram sketch tool. It creates graphics in UML style but no UML models.

5 Discussion

This section discusses the presented results of the survey. Section 5.1 discusses observations based on the results. Section 5.2 discusses threats to validity.

5.1 Findings

The analyzed notations can be divided into two groups: 14 notations focuses on generating diagrams for documentation purposes but do not consider UML modeling. These notations mainly regard the text to graph change propagation and focus on graphical export. 17 notations focus on modeling or further analysis. The relative share of web tools in the first group (5) is twice as much as in the second group (3). Most scientifically published notations mentioned in (Luque et al., 2014a) are in the first group as well.

[11] `www.umlet.com`

[12] `http://www.macroexpand.org/doku.php`

[13] `http://www.txl.ca/eclipse/TXLPluginGuide.pdf`

[14] `http://garapa.intermidia.icmc.usp.br:3000/awmo/`

[15] `http://blockdiag.com/en/`

[16] `https://bitbucket.org/blockdiag/seqdiag`

[17] `http://blockdiag.com/en/actdiag/index.html`

[18] `http://www.stateforge.com/`

[19] `https://bramp.github.io/js-sequence-diagrams/`

[20] `https://github.com/skanaar/nomnoml`

[21] `https://www.websequencediagrams.com/`

Table 2: Characteristics of analyzed textual UML notations. Characteristics are: not extractable (-), given (✓), or not given (×). Layout information is: mixed (*m*) or separated (*s*). The License is: open (*O*) or closed (*C*) source.

	Language (mandatory)				Implementation															
					Recent Activity					Format Exchange		Editor								
												Textual							Graph.	
												Visualization			Navigation					
Notation	UML Support (mandatory)	Sketch Support	Layout Information	Similar to UML Graphics	Release	Ticket	Commit	License	Change Propagation	Export	Import	Syntax Highlighting	Folding	Refactoring	Code Completion (Syntax)	Code Completion (Element)	Outline	Goto	Editable	Layoutable
Alf	CLS,ACT,PKG	✓	×	×	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Alloy	×	×	×	×	22.02.15	×	×	O	×	DOT, XML	ALS	✓	×	×	×	×	×	×	×	✓
AUML	SEQ	×	×	×	05.09.14	20.11.14	-	-	T2G	PNG	×	✓	×	×	×	×	×	×	-	-
AWMo	CLS	×	×	×	×	×	05.11.13	C	T2G, G2T	×	×	-	-	-	-	-	-	-	-	-
blockdiag: seq-,actdiag	SEQ,ACT	✓	m	✓	01.01.15	10.09.14	22.08.15	O	T2G	PNG,SVG,PDF	×	-	-	-	-	-	-	-	-	-
Clafer	CLS,OBJ	×	×	×	28.07.15	11.03.15	28.07.15	O	×	own,DOT,Z3Py, ALF,ChocoJS	×	✓	×	×	×	×	×	×	-	-
Dcharts	STM	✓	×	×	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Earl Grey	CLS,SEQ,STM	×	×	×	25.05.12	09.04.12	23.05.12	O	×	×	×	✓	✓	✓	✓	✓	✓	✓	-	-
Finite State Machine DE	STM	✓	×	×	02.04.15	-	-	O	T2G, G2T	own	own	✓	×	×	×	×	×	×	✓	×
HUTN	all	✓	×	×	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
IOM/T	SEQ	×	×	×	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
js-sequence-diagrams	SEQ	✓	m	✓	17.05.15	14.08.15	27.07.15	O	T2G	SVG	×	-	-	-	-	-	-	-	-	-
MetaUML	CLS,STM,ACT, UC,CMP,PKG	✓	m	×	23.06.15	23.08.15	07.08.15	O	T2G	×	×	-	-	-	-	-	-	-	-	-
modsl	CLS,COM	✓	×	×	11.08.09	09.11.09	11.08.09	O	T2G	PNG, JPG	×	✓	✓	✓	✓	×	✓	✓	-	-
Nomnoml	CLS,OBJ,STM, UC,PKG	✓	m	✓	×	14.08.15	03.08.15	C	T2G	PNG	×	-	-	-	-	-	-	-	-	-
pgf-umlcd	CLS	✓	m	×	28.04.14	22.08.15	31.05.15	O	T2G	×	×	✓	×	×	×	×	×	×	-	-
pgf-umlsd	SEQ	✓	m	×	24.02.14	09.02.15	24.02.14	O	T2G	×	×	✓	×	×	×	×	×	×	-	-
PlantUML	CLS,OBJ,SEQ, STM,ACT,UC, CMP,DEP	✓	m	✓	05.08.15	23.10.14	05.08.15	O	T2G	UML,SVG,EPS, TXT,HTML	×	-	-	-	-	-	-	-	-	-
Quick Sequence Diag. Edit.	SEQ	✓	×	×	11.02.13	22.01.15	01.02.15	O	×	PDF,JPEG,SVG, SWF,EMF,GIF, JPEG,(E)PS	×	-	-	-	-	-	-	-	-	-
TCD	CLS	✓	×	✓	-	-	-	-	IE	UML	UML	-	-	-	-	-	-	-	-	-
TextUML	CLS, STM	×	×	×	20.08.15	02.08.15	20.08.15	O	×	UML	×	✓	×	×	✓	×	✓	×	-	-
tUML	CLS, STM, COS	×	×	×	-	-	-	-	T2G, IE	UML	UML	✓	✓	×	×	×	✓	✓	×	×
txtUML	CLS, STM, ACT	×	×	×	14.07.15	-	-	-	T2G	UML	×	✓	✓	×	×	×	✓	✓	-	-
UML/P	CLS, OBJ, SEQ, STM, ACT	✓	s	✓	-	-	-	C	T2G	×	×	✓	✓	×	✓	×	✓	✓	✓	×
UMLet	CLS, OBJ, UC, PKG	✓	m	×	19.03.15	13.08.15	12.08.15	O	×	BMP,EPS,GIF, JPG,PDF,PNG	UXF	✓	×	×	×	×	×	×	✓	✓
UMLGraph	CLS, SEQ	✓	m	×	28.10.14	29.10.14	29.10.14	O	T2G	PNG,fig,PS,GIF EMF,SVG,JPEG	×	-	-	-	-	-	-	-	-	-
uml-sequence-diagram-dsl-txl	SEQ	✓	m	✓	29.08.09	×	×	×	T2G	XML, Code	×	✓	✓	×	✓	×	✓	✓	-	-
Umple	CLS, STM, COS	✓	s	✓	24.02.15	07.02.15	20.08.15	O	T2G, G2T	UML,ALS,USE, EMF,UXF,Code, TUML,yUML	×	✓	×	×	×	×	×	×	✓	✓
USE	CLS	×	s	×	04.08.15	19.09.14	-	O	T2G	PDF	×	✓	×	×	×	×	×	×	✓	×
WebSequence-Diagrams	SEQ	✓	m	✓	-	-	-	×	T2G	×	×	-	-	-	-	-	-	-	-	-
yUML	CLS, ACT, UC	✓	×	✓	-	-	-	-	T2G	PNG,PDF,SVG, JPEG,JSON	×	-	-	-	-	-	-	-	-	-

Only 23% of the remaining notations of (Luque et al., 2014a) that are also present in our survey do not focus on graphics generation. This shows that the delivery method of the editor has no significant effect on the editing experience and applicability.

Notations that focus on sketching graphics often disregard formal UML rules and having a UML model. This leads to some drawbacks: First, the results of the textual modeling cannot be processed in tool chains. This is true for all notations that do not provide export formats other than graphics. Second, interpreting the generated graphic might be hard because determining the semantics of the elements requires spatial thinking. Non-UML graphics such as the conditional node in the PlantUML activity diagram [22] make this even harder. Third, the (semantic) relation of graphic elements is inaccessible for machines and visually impaired people. These drawbacks limit consistency and collaborative editing. We only identified five notations providing UML-compaptible model export.

19 notations cover less than three UML diagram types. The most supported diagram type is the CLS diagram (20) followed by the SEQ (13) and STM diagram (12). TIM, PRO, and INT diagrams are only supported by the HUTN. 15 do not provide editing environments at all. Based on our results, notations are specialized and there is no notation that supports collaborative editing in textual and graphical notations of more than five UML diagrams. MetaUML (6) and PlantUML (8) come with the highest UML diagram type coverage and provide an implementation but no state-of-the-art editing environment. We could not identify a one-fits-all notation in our survey.

Almost half of the reviewed notations (14) have recent activity in the year 2015. Therefore, we expect improvements in the syntax and implementation of the notations in the near future.

5.2 Threats to Validity

We address four common threats to internal validity: incomplete selection, inconsistent measurements, biased experimenter, and incomplete information.

We addressed *incomplete selection* with two additional phases that check the completeness of search results. During the survey, we found a total of 31 textual UML notations. We found half of them in the SLR phase. In the Quality Assurance phase, we found four new papers and three new notations. The first phase did not reveal three of these papers (Jackson, 2002; He, 2006; Doi et al., 2004) because their main contribution was not about a textual UML notation. Therefore, they did not clearly indicate that they also cover a textual UML notation in their title or abstract. The remaining paper (Dévai et al., 2014) is not indexed by the search engines that we used. The major new result of the completion phase was the textual UML tool list (Cabot, 2015) provided by Jordi Cabot, a professor with research interests in model-driven software engineering at the ICREA research institute. We found eleven new notations compared to the previous phase. We did not find ten of them in earlier phases because those phases are focused on scientific notations and the found notations have not been scientifically published. The remaining notation (Del Nero Grillo and de Mattos Fortes, 2014) is not indexed by the search engines we used. Therefore, we consider the keywords of the SLR phase and the whole notation finding process to be successful and appropriate.

The Complement phase did not include an extensive search strategy because we focus on scientifically published notations in this survey. We complement the intensive search strategies of the previous phases with the most common notations used in industry. To achieve this, we imitate the common search strategy that covers the very first results only because they are the most popular ones. We included all notations of previous surveys in the analysis. In total, we found 14 new notations compared to previous surveys: Alf, Alloy, AUML, Clafer, Dcharts, HUTN, IOM/T, Nomnoml, pgf-umlcd, pgf-umlsd, tUML, txtUML, UML/P, and uml-sequence-diagram-dsl-txl.

We addressed *inconsistent measurements* and *biased experimenter* with a rigorous review protocol and instructions for the characteristics extraction. The characteristics for the notations can be determined in an objective way. Mazanec et al. (Mazanec and Macek, 2012), however, used subjective characteristics such as *readability* or *simplicity* and did not mention how they have been determined.

We addressed *incomplete information* by using multiple information sources. We characterized all 31 notations by extracting information from the papers, and mining websites and source code (if possible). The former is the standard approach during a SLR but the two latter allow filling the gaps left by the scientific papers. Especially, the project's activity and editor features are most commonly not covered by publications. Only Alf, Dcharts, HUTN, and IOM/T did not provide sufficient information to determine these characteristics.

The external validity requires generalizable results. The survey results are applicable for scenarios that cover collaborative UML editing with textual no-

[22] `http://plantuml.com/activity2.html`

tations in general because the characteristics do not focus on a specific scenario. This is a benefit over the previous surveys (Luque et al., 2014a; Luque et al., 2014a) that focused on teaching UML to visually impaired people from industry applying UC and CLS diagrams. The fuzzy characteristics in (Mazanec and Macek, 2012) lead to a limited generalization and applicability.

6 Conclusions

UML is most commonly used for modeling tasks in various domains. It provides a graphical but no textual notation. The results show that the lack of a standard leads to many different and incompatible textual notations tailored for specific purposes. They largely differ in syntax, coverage, editing experience and applicability in teams. This diversity requires high effort to evaluate and select a notation for a specific purpose and environment.

The contribution of this survey is the identification and classification of textual UML modeling notations. Teams that plan to incorporate a textual modeling notation can save effort by using our survey because a) we provide a comprehensive list of available notations, b) we classified the notations with respect to characteristics that are important for their practical use in teams, and c) are usage scenario independent.

The applied review method ensures reproducible and reliable results, which is crucial for using our survey as a foundation for the approach selection. The review method covered a traditional SLR and two additional phases for ensuring the quality and completeness of the review. The SLR phase consisted of a keyword-based search in search engines relevant for the software engineering domain. The quality assurance phase applied one iteration of the snowballing approach as cross-check for the quality of our SLR phase. The complement phase completed the list of notations used in practice but without academic background. The two latter phases identified about the same amount of notations as the SLR showing the strength of our SLR method.

The classification is tailored to include the user's point of view and support the notation selection in teams. We presented each of the twenty categories in detail including objectively checkable conditions that cover the level of UML support, the editing experience, and the applicability in an engineering team. The categories are also a reference for future surveys in the area.

About half of the identified notations focus on the generation of graphics for documentation purposes based on notations on the code level. Many notations focus on discussions using purely graphic notations. They provide UML shapes and allow their arbitrary placement on a canvas. This prevents automated processing such as checking model validity or compliance to existing code. This prevents the usage in many context and can be taken into account during selection.

The restriction to graphical exports prohibits collaborative editing of graphical as well as textual representations in engineering teams although both have their advantages. The majority of notations only cover less than three UML diagram types. Most of them do not provide a state-of-the-art editor. This shows that users have to select notations based on their own usage scenario and decide on the supported UML diagram types and the required editor support. Practitioners can leverage our survey to prioritize their analysis of notations.

We could not identify a single comprehensive textual UML notation that comes with a state-of-the-art editing environment and standard import/export formats. This incompatibility suggests future work for researchers and practitioners as well: Researchers can analyze the gap in UML coverage or if there is a best of breed notation available for all diagrams. Practitioners can compare the available notations more easily and see which features can help users.

From our perspective, there is a need for notations that target proper UML modeling and have the ability to import and export standard UML models. This allows engineering teams to include the notation in existing tool chains and enables collaboration.

Acknowledgements

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

REFERENCES

Auer, M., Tschurtschenthaler, T., and Biffl, S. (2003). A flyweight uml modelling tool for software development in heterogeneous environments. In *Proceedings of EUROMICRO'03*, pages 267–272. IEEE Computer Society.

Cabot, J. (2015). Modeling languages – uml tools. `https://modeling-languages.com/uml-tools`. accessed 04. August 2015.

Chaves, R. (2015). Textuml toolkit. `http://abstratt.github.io/textuml/readme.html`. accessed 14. August 2015.

Del Nero Grillo, F. and de Mattos Fortes, R. (2014). Tests with blind programmers using awmo: An accessible web modeling tool. In *UAHCI'14*, pages 104–113.

Dévai, G., Kovács, G. F., and An, Á. (2014). Textual, executable, translatable UML. In *Proceedings of the 14th International Workshop on OCL and Textual Modelling*, pages 3–12.

Doi, T., Yoshioka, N., Tahara, Y., and Honiden, S. (2004). Bridging the gap between AUML and implementation using IOM/T. In *Proceedings of ProMAS'04*, pages 147–162.

Erb, S. (2011). Textual modeling of service effect specifications. `http://stephanerb.eu/files/erb2011a_Textual_Modeling_of_Service_Effect_Specifications.pdf`. accessed 04. August 2015.

Feng, H. (2004). *DCharts, a formalism for modeling and simulation based design of reactive software systems*. Master's thesis, School of Computer Science, McGill University, Montreal, Canada.

Gheorghies, O. (2015). Metauml - github. `https://github.com/ogheorghies/MetaUML`. accessed 14. August 2015.

Grönniger, H., Krahn, H., Rumpe, B., Schindler, M., and Völkel, S. (2014). Text-based modeling. *CoRR*, abs/1409.6623.

Harris, T. (2015). Create uml diagrams online in seconds, no special tools needed. `http://yuml.me`. accessed 14. August 2015.

He, Y. (2006). Comparison of the modeling languages alloy and UML. In *Proceedings of SERP'06*, pages 671–677.

ISO/IEC 13568:2002 (2002). Information technology – z formal specification notation – syntax, type system and semantics. Standard, International Organization for Standardization.

Jackson, D. (2002). Alloy: a lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.*, 11(2):256–290.

Jouault, F. and Delatour, J. (2014). Towards fixing sketchy UML models by leveraging textual notations: Application to real-time embedded systems. In *Proceedings of the 14th International Workshop on OCL and Textual Modelling*, pages 73–82.

Kern, H. (2014). Study of interoperability between metamodeling tools. In *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*, pages 1629–1637.

Khaled, L. (2009). A comparison between uml tools. In *ICECS'09*, pages 111–114.

Kitchenham, B. and Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering (version 2.3). EBSE technical report, EBSE-2007-01, Keele University.

Lethbridge, T. (2014). Umple: An open-source tool for easy-to-use modeling, analysis, and code generation. In *Proceedings of the Demonstrations Track of MoDELS'14*.

Luque, L., Brandão, L. O., Tori, R., and Brandão, A. A. F. (2014a). Are you seeing this? what is available and how can we include blind students in virtual uml learning activities. In *Proceedings of SBIE'14*.

Luque, L., Veriscimo, E., Pereira, G., and Filgueiras, L. (2014b). Can we work together? on the inclusion of blind people in uml model-based tasks. In *Inclusive Designing*, pages 223–233. Springer International Publishing.

Mazanec, M. and Macek, O. (2012). On general-purpose textual modeling languages. In *Proceedings of Dateso'12*, pages 1–12.

OMG (2013). Action language for foundational uml (alf). `http://www.omg.org/spec/ALF/1.0.1/PDF`.

OMG (2015a). Unified Modeling Language (UML) – Version 2.5. `http://www.omg.org/spec/UML/2.5/PDF`.

OMG (2015b). XML Metadata Interchange (XMI) – Version 2.5.1. `http://www.omg.org/spec/XMI/2.5.1/PDF`.

Open Source Initiative (2015). Licenses by name. `http://opensource.org/licenses/alphabetical`. accessed 04. August 2015.

Roques, A. (2015). Plantuml : Open-source tool that uses simple textual descriptions to draw uml diagrams. `http://plantuml.com/`. accessed 14. August 2015.

Safdar, S. A., Iqbal, M. Z., and Khan, M. U. (2015). Empirical evaluation of uml modeling toolsa controlled experiment. In *Modelling Foundations and Applications*, volume 9153 of *Lecture Notes in Computer Science*, pages 33–44. Springer International Publishing.

Spinellis, D. (2003). On the declarative specification of models. *IEEE Software*, 20(2):94–96.

Vieritz, H., Schilberg, D., and Jeschke, S. (2014). Access to uml diagrams with the hutn. In *Automation, Communication and Cybernetics in Science and Engineering 2013/2014*, pages 751–755. Springer International Publishing.

Washizaki, H., Akimoto, M., Hasebe, A., Kubo, A., and Fukazawa, Y. (2010). Tcd: A text-based uml class diagram notation and its model converters. In *Advances in Software Engineering*, volume 117 of *Communications in Computer and Information Science*, pages 296–302. Springer Berlin Heidelberg.

Wikipedia (2015). List of unified modeling language tools. `https://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools`. accessed 04. August 2015.

Winikoff, M. (2005). Towards making agent UML practical: A textual notation and a tool. In *2005 NASA / DoD Conference on Evolvable Hardware (EH 2005)*, pages 401–412.

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of EASE'14*, pages 38:1–38:10. ACM.

Zayan, D. O. (2012). Model evolution: Comparative study between clafer and textual uml. `http://gsd.uwaterloo.ca/sites/default/files/Model%20Evolution;%20Clafer%20versus%20Textual%20UML.pdf`. Project Report.