TU WIEN Informatics

Conceptualization and Implementation of UML Sequence Diagrams in a GLSP-based UML Modeling Tool

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Business Informatics

eingereicht von

Simone Andreetto, BSc.
Matrikelnummer 01635069

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Assistant Prof. Dipl.-Wirtsch.Inf.Univ. Dr.rer.pol. Dominik Bork

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

TU Bibliothek Your knowledge hub

Wien, 6. Dezember 2023

Simone Andreetto	Dominik Bork

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

TU WIEN Informatics

Conceptualization and Implementation of UML Sequence Diagrams in a GLSP-based UML Modeling Tool

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Business Informatics

by

Simone Andreetto, BSc.
Registration Number 01635069

to the Faculty of Informatics

at the TU Wien

Advisor: Assistant Prof. Dipl.-Wirtsch.Inf.Univ. Dr.rer.pol. Dominik Bork

Vienna, 6th December, 2023

Simone Andreetto	Dominik Bork

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Simone Andreetto, BSc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 6. Dezember 2023

Simone Andreetto

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Danksagung

Ich möchte mich bei all jenen bedanken, die mir nicht nur während der Anfertigung meiner Diplomarbeit, sondern auch während meines gesamten akademischen Werdegangs zur Seite standen.

Ein herzliches Dankeschön geht an meinen Betreuer, Dominik Bork, für seine unermüdliche Unterstützung, sein aufschlussreiches Feedback und seine wertvollen Hinweise auf beachtenswerte Literatur. Die dynamischen und zum Nachdenken anregenden Diskussionen, die wir geführt haben, haben wesentlich zum erfolgreichen Abschluss dieser Arbeit beigetragen.

Darüber hinaus möchte ich mich bei der Business Informatics Group (BIG) dafür bedanken, dass sie mir während der Arbeit einen angenehmen Arbeitsplatz zur Verfügung gestellt hat. Diese Umgebung steigerte meine Produktivität und erleichterte die Zusammenarbeit mit meinem Betreuer. Mein Dank gilt allen Mitarbeitern bei BIG für ihre Beiträge.

Meinen Freunden, Kollegen und insbesonders Felix, Dominik und Lukas, eure Begleitung und Unterstützung während meiner Zeit an der TU Wien waren ein wesentlicher Bestandteil dieser Reise. Eure Anwesenheit hat mir viel Freude bereitet und mich bereichert, was ich sehr schätze.

Schließlich bin ich meiner Familie und Laura zutiefst dankbar. Ihre ständige Ermutigung und Unterstützung haben entscheidend dazu beigetragen, dass ich mein Studium erfolgreich abschließen konnte. Ohne Ihre Anwesenheit wäre dieser Erfolg nicht möglich gewesen.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

I want to extend my gratitude to all those who supported me, not only during the writing of my thesis but also throughout my academic journey.

A heartfelt expression of thanks goes to my thesis advisor, Dominik Bork, for his unwavering support, insightful feedback, and valuable references to noteworthy literature. The dynamic and thought-provoking discussions we engaged in significantly contributed to the triumphant culmination of this thesis.

Furthermore, I would like to acknowledge the Business Informatics Group (BIG) for providing me with a conducive workspace during the thesis period. This environment amplified my productivity and facilitated collaboration with my supervisor. My appreciation extends to everyone at the BIG for their contributions.

To my friends, colleagues, and especially Felix, Dominik, and Lukas, your companionship and support during my tenure at TU Wien have been an integral part of this journey. Your presence has added an enjoyment and enrichment that I deeply value.

Lastly, I am deeply grateful to my family and Laura. Your constant encouragement and support have been pivotal in enabling me to complete my studies successfully. Without your presence, this achievement would not have been attainable.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Das Sequenzdiagramm (SD) spielt eine entscheidende Rolle in der Softwareentwicklung und Unternehmensorganisation. Es definiert Prozesse und Systeminteraktionen mit präziser zeitlicher Semantik. Diese Präzision führt zu zusätzlicher Komplexität, was die Verwendung von SD-Modellierungstools erschwert. Die derzeit verfügbaren SD-Modellierungstools haben Schwierigkeiten, eine funktionale und dennoch benutzerfreundliche Palette von Interaktionsverhaltensweisen bereitzustellen. Diese Herausforderung führt zu verschiedenen Implementierungen und Modellierungsverhalten, was zu einer weit verbreiteten mangelnden Benutzerfreundlichkeit führt. Die Tools sind gegenwärtig stark von etablierten Technologien abhängig, was die Realisierung effizienterer Funktionen beeinträchtigt. Eine Migration zu zeitgemäßen Webtechnologien könnte fortgeschrittenere Modellierungsinteraktionen ermöglichen.

Diese Arbeit analysiert die bekanntesten SD-Modellierungstools und gibt einen systematischen Überblick über ihre Funktionalitäten und implementierten Verhaltensweisen. Darauf aufbauend wird ein SD-Modellierungstools mit den entsprechenden Interaktionen und Editierverhalten konzeptioniert, um die semantischen Anforderungen zu erfüllen und die Benutzerfreundlichkeit zu gewährleisten. Das Konzept wird in einem neuen SD-Modellierungstool umgesetzt, das auf der Graphical Language Server Platform (GLSP) entwickelt und den Open-Source-UML-Modellierungseditor BIGUML, der sich derzeit in der Entwicklung befindet, um die Unterstützung für SDs ergänzt. Das Tool verfügt über wesentliche SD-Modellierungsfunktionen und bietet zusätzlich spezielle Funktionen, die den Modellierungsprozess reaktiver und dynamischer gestalten. Es kombiniert robuste Modellierungsfunktionen mit dem Komfort webbasierter Features.

Die Implementierung der Konzepte wird auf der Grundlage der festgelegten Anforderungen bewertet, und das resultierende Artefakt durchläuft einen definierten Modellierungsprozess, um einen Vergleich mit den bestehenden Lösungen zu ermöglichen.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

The sequence diagram (SD) plays a significant role in software development and business organization. It defines processes and systems interactions with a precise temporal semantic. The temporal semantics bring about additional complexity, making SD modeling tools bothersome. Available SD modeling tools struggle to provide a functional yet user-friendly palette of interaction behaviors. This difficulty results in divergent implementations and modeling behaviors, causing widespread poor usability. The reliance of tools on long-established technologies further hinders more efficient tool functions and design. Switching to modern web technologies could unlock more advanced modeling interactions. Doing so as part of an open-source project allows the adoption of implementations in other diagrams and enables future adaptions and improvements of the tool.

This thesis analyses the most noted SD modeling tools and derives a systematic overview of their functionalities and implemented behaviors. Based on this, an SD modeling tool with its respective interactions and editing behaviors is conceptualized to fulfill the semantic requirements and ensure usability. The concept is realized as an artifact developed on the Graphical Language Server Platform (GLSP), adding the support for SDs to the open-source UML modeling editor BIGUML currently in development. The tool features essential SD modeling behaviors with the addition of specialized functionalities, rendering the modeling process more reactive and dynamic. It combines robust modeling functionalities and the convenience of web-based features.

The implementation of the concepts is assessed against the conceptualized requirements, and the resulting artifact undergoes evaluation through a defined modeling process, enabling a comparison with existing solutions.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 1

Introduction

This chapter outlines the problem addressed within this thesis and the driving motivation behind it. It introduces the domain of conceptual modeling and, specifically, sequence diagram modeling. It delves into the current challenges within current modeling practices and modeling tools. It specifies the posed research questions answered throughout the work and presents the methodological approach. Lastly, it provides an overview of the subsequent chapters with concise descriptions of their content.

1.1 Motivation & Problem Statement

Conceptual models play a significant role in software development and business organizations. Models are based on well-defined metamodel standards, such as the Unified Modeling Language (UML) [Gro17], which allows accurate modeling of processes and systems. Existing modeling tools are complete in their utility but often lack usability, making the modeling process cumbersome and reserved for specialized modelers [Hop18] while increasing the user's necessary time and cognitive load. This issue is especially prominent with semantically complex diagrams such as the sequence diagram (SD), making it the focus of this thesis.

SDs define interactions between objects along a temporal dimension in sequential order and find applications in software development and business organization. The SD conveys additional complexity compared to other diagrams due to the semantic meaning of the relative position of its elements, as visualized in Figure 1.1. The vertical sequence of elements indicates their sequential temporal order, and the relative object height and vertical distances convey the meaning of duration.

The most common conceptual modeling tools [Fakb] support SDs. However, the behavior implemented in response to changes in elements along the temporal dimension differs.

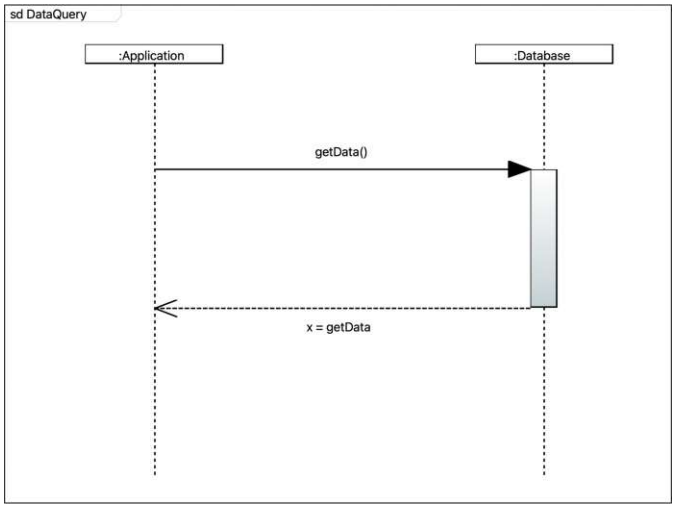


Figure 1.1: Exemplary SD for data query

Research must identify and systematically collect their functionalities and behaviors, evaluate their implementations, and conceptualize a unified behavior.

The interaction with the vertical ordering and distances is crucial when creating or editing SDs. If and how a modeling tool behaves in response to new elements, the movement and removal of elements affect the tool's usability and make it a central question of this thesis. Most available tools do not react to changes, thus requiring the user to manually adjust the diagram, resulting in a tedious and repetitive job of micromanaging the diagram's layout. Possible dynamic behaviors to assist the modeling process include rearrangement and shift in the order and position of elements. Some tools implement a rudimentary approach to such behaviors, but no tool provides a complete solution. Conclusively, the modeling tools do not meet the users' expectations of modern software, which is also shown by the efforts in the research of automated layout functionalities, which currently are missing or limited [NPA16].

1.2 Aim of the Work

These findings suggest the need for a more reactive SD modeling tool to reduce the users' untrivial workload, such as manually readjusting the model's layout after changes. This requires identifying concepts of behaviors that provide a less work-intensive modeling experience while ensuring the semantic requirements of the model.

New technologies appear vital to facilitate a feasible implementation that improves existing solutions. This thesis explicitly explores using the GLSP to enable advanced modeling interaction. Evaluating whether the new technology benefits the final solution compared to the proven and matured implementations forms the concluding question of this thesis.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

The elaborated issues can be best summed up by answering the following research questions:

- RQ1: What editing behaviors and interactions do current SD modeling tools offer?
- RQ2: Which behaviors significantly increase manual interactions by the user and encumber the SD modeling process?
- RQ3: Which concepts for tool behavior can reduce the necessary modeling user interactions and, in that respect, improve the efficiency of the SD modeling tool?
- RQ4: What advantages and disadvantages, concerning the prospect of new functionalities and efforts of implementation, does the implementation of GLSP-based modeling bring in comparison to existing tools?

These questions guide the research process of the thesis that follows them in its structure. Chapter4 describes the exploratory work on existing SD editors and their described functionalities to provide insights into RQ1. To respond to RQ1, we select the most notable and currently used modeling tools and evaluate them in chapter 5 on their functionalities through a consistent modeling process. This provides concrete insight into their editing behaviors and further supports the statements concerning RQ1.

Through the evaluation of the selected SD modeling tools, their modeling behaviors are evaluated and graded based on their usability to uncover shortcomings and cumbersome behaviors, as asked in RQ2. The evaluation provides a foundation of well-proven modeling behaviors and a selection of cumbersome interactions with the need for improvement.

In chapter 6, the editing behaviors with a significant negative impact on usability are redesigned with new concepts of interactions to improve efficiency and ease of use (RQ3). The collection of the redesign behaviors, in combination with the existing well-established and proven interactions, forms the redesigned concept for SD modeling.

Chapter 7 guides the implementation of the new modeling concept as an artifact. This chapter highlights the chosen architecture's technical advantages and implementation efforts that impact the response to RQ4. The implemented artifact is evaluated using the initial evaluation process to compare to the existing SD modeling tools implemented in the second to last chapter 8. This final evaluation was conducted in chapter 8. We applied the established evaluation method for the existing modeling tools in chapter 4 to our created artifact. The outcome allowed us to objectively compare our artifact with existing tools and uncover possible shortcomings for future improvements.

1.3 Methodology

The used methodologies follow the methods described by the design science framework and the guidelines for design science research (DSR) [HMPR04]. The methods concern the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

artifact creation, the problem's relevance, the design evaluation, the research contribution, the research rigor, the design search process, and the research communication.

To ensure applicability and generalizability as required from research rigor, the concept and the application interaction behaviors are defined in a general manner without referencing technology-specific methods and feasibility. Past knowledge is introduced in the research as the current state-of-the-art implementations as proposed in the rigor cycle [Ala07]. This thesis is divided into phases following the structure established by the expected results:

1.3.1 Modeling Literature & Tools Analysis

The thesis starts with research on relevant works on sequence diagram modeling and modeling tools. We collect and analyze available catalogs of modeling tools and existing evaluations of modeling tools, focusing on sequence diagrams. The design search process starts with collecting the existing sequence diagram tools' modeling methods and implementations. As part of the rigor cycle, [Ala07], this initial phase aims to research and reference the available knowledge base to provide grounding truth and verify and ensure research contribution.

1.3.2 Existing Tools Evaluation

A reference SD modeling process is defined and performed with each SD tool to achieve an objectively comparable evaluation of the individual SD tools and categorize their behaviors. The modeling process is initialized with an empty editor view and ends once the defined SD is modeled. The modeling process contains the strictly necessary steps, such as creating diagram elements. These are extended with additional intermediary steps, such as removing, moving, and replacing existing components to replicate a natural user behavior and consider a broader range of interactions. The modeling process qualitatively evaluates the usability of the individual modeling tools by ensuring consistent testing of the functionalities across all the editing tools and systematically uncovering differences between the tools. If one specific step is not supported by a modeling tool, the step is skipped or adapted with a comparable alternative to allow the continuation of the process. Of particular interest is how the modeling editors handle object interactions along the temporal dimension of the diagram. The outcomes are collected in a tabular fashion, and the tools' functionalities are categorized based on their implementation.

1.3.3 Conceptualization

The derived insights from this research serve as a foundation for further conceptualizing ideal sequence diagram editing behaviors and provide a baseline for implementing the artifact. The concept assembles the most effective and efficient approaches from other tools and defines variations. The created concept from the initial phase undergoes an ex-ante criteria-based evaluation. It is evaluated on fulfilling the requirements posed by

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

the definition of SDs. This ensures a solid foundation to start development and reduces additional costs and modifications further down the implementation process.

1.3.4 Artifact Implementation

The development process employs creating individual behavioral methods and solutions for the tool, which are tested for their functionality against the requirements defined by the concept. As part of the design process testing cycle [Sim01], the solution's quality is tested compared to the existing sequence diagram tools. We focus on its usability and intuitiveness of interaction along the modeling process of sequence diagrams. This is described in greater detail as part of the design evaluation. The divide and conquer approach is adopted to simplify the sequence diagram tool's development process. The tool is decomposed into its behaviors and functions as individual problems and combined into a complete artifact. The developed artifact undergoes functional testing to ensure its functionality and the satisfaction of the defined requirements. This step happens continuously throughout the development process in incremental growth, from testing individual functionalities to testing the tool in its entirety, as proposed in Hevner's design cycle [Ala07].

1.3.5 Artifact Evaluation

The finalized artifact is evaluated on its utility and usability. The diagram's utility is tested by its capability of representing all the relevant diagrams extracted as examples from UML@Classroom [SSHK15], which covers most of the tools' required functionalities. The developed artifact undergoes functional testing to ensure its functionality and meets the defined requirements in the in chapter 6.

As part of the design cycle, this step occurs continuously throughout the development process in incremental growth, from individual functionalities to testing the tool. Since usability is a subjective measurement, this evaluation is conducted through a defined modeling process with a predefined set of questions reflecting on the experienced interaction. We conducted it as a descriptive evaluation based on informed arguments following the defined modeling scenario and the related reflections, as described by Hevner [HMPR04].

1.4 Structure

This thesis is structured in the following chapter:

Chapter 2 provides supporting background information on the discussed topics and domain to ensure a clear understanding by the reader. This chapter introduces a general view of the domain of model engineering and the general motivation behind it. We provide an overview of the Unified Modeling Language (UML) and focus on the sequence diagram and its possible applications.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Chapter 3 outlines related research on modeling tools and SD-specific modeling tools, which cataloged or evaluated them. Furthermore, it investigates existing research on available modeling solutions, outlines previous endeavors related to tool implementations, and explores research that delves into and showcases the adoption and implementation of GLSP.

Chapter 4 collects available SD modeling tools from existing collections out of related works. The tools are categorized and then selected based on criteria defined in the chapter.

Chapter 5 introduces and outlines the evaluation process for SD modeling tools, which is applied to selecting available SD modeling tools. The methodology and its results offer in-depth insights into the existing landscape of interactions and solutions. Additionally, they highlight the unique limitations and disparities among these tools.

Chapter 6 establishes the requirements for an SD modeling tool and illustrates the concept for improvements by outlining the most beneficial interactions and tools to support SD modeling practices. It focuses on applications and interactions specific to the SD, some of which are extended as concepts generally applicable to all diagram types.

Chapter 7 describes the implementation of the SD modeling editor, focusing on SD-specific requirements and encountered challenges. This chapter provides additional insights into working with and expanding BIGUML.

Chapter 8 evaluates the developed artifact by applying the established evaluation schema from chapter 5. It reflects on current shortcomings and possible improvements.

Chapter 9 provides the thesis's conclusion, summarizing the achievements and the answers to the research questions posed.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 2

Background

This chapter provides supporting background information on the discussed topics and domain to ensure a clear understanding by the reader. We introduce a general view of the domain of model engineering and its general motivation, focusing on SDs and their use cases in the domain. A detailed description of the assembling components of SDs is provided as a foundation for the conceptualization part of this thesis. As our work includes the implementation of an artifact, we provide technical background information on the utilized technology and architectural stack.

2.1 Unified Modeling Language - UML

The sequence diagram is part of the collection of diagrams defined within the Unified Modeling Language (UML) by the Object Management Group (OMG). The UML has been in development since 1995 (UML 0.8), was majorly revised in 2005 with v2, and is in its latest iteration with the v.2.5.1 specification, released in December 2017, which is utilized throughout this thesis [OMG17].

UML is a general-purpose, developmental modeling language and an industry-wide accepted modeling standard. As found in the 2.5.1 UML specification: "The objective of UML is to provide system architects, software engineers, and software developers with tools for analysis, design, and implementation of software-based systems as well as for modeling business and similar processes." [Gro17] UML is widely used to represent system specifications at different abstraction levels.

To achieve this objective, UML defines 14 different diagrams divided into structure and behavior diagrams:

Structure Diagrams

1. Class Diagram

2. Component Diagram
3. Deployment Diagram
4. Object Diagram
5. Package Diagram
6. Profile Diagram
7. Composite Structure Diagram

Behavior Diagrams

8. Use Case Diagram
9. Activity Diagram
10. State Machine Diagram
11. Sequence Diagram
12. Communication Diagram
13. Interaction Overview Diagram
14. Timing Diagram

Out of the 14 diagrams, the most known and utilized diagrams include the Use Case Diagram, the Class Diagram, and the Sequence Diagram.

2.2 UML Sequence Diagram

The sequence diagram plays a significant role in representing dynamic systems and their interactions and is, therefore, part of the interaction diagrams, a subset of the behavior diagrams. The sequence diagram models the behaviors of multiple objects and object groups in a single scenario/use case; the behavior consists of a sequence of messages exchanged between the objects [Fow04].

2.2.1 Structure

To provide a general understanding of an SD, we offer a concise overview of its critical components visualized in Figure 2.1 and Figure 2.1. Each component is summarised based on its syntax, semantics, and notation. The following notations describe the key components, and the respective visualizations follow the UML specifications [Micb] with the incorporation of supporting materials [Fakb, Gee17], guiding modelers through the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

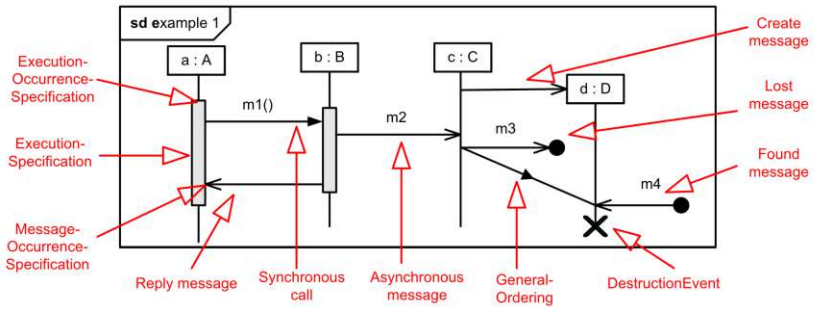


Figure 2.1: Example of SD with messages (Fig. 2 of [MW08])

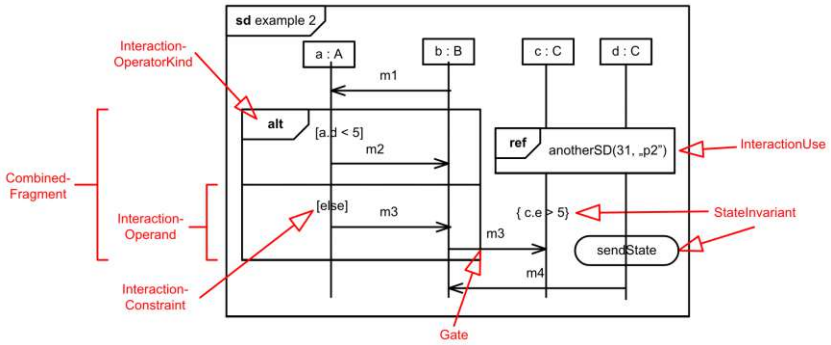


Figure 2.2: Example of SD with combined fragment (Fig. 3 of [MW08])

individual components, and their use in the SD. While the described notation follows the UML specification, the visualization can vary across editing tools [Bel04, Par, Sta].

Interaction or interaction frame is the overarching compartment containing all further components. It is represented as a rectangle with a solid outline, which in the upper left corner has a pentagon containing the label beginning with "sd", the interaction name, and parameters.

Lifelines are symbolic representations of entities or actors participating in the interaction. These entities are depicted as vertical lines, each bearing an identifying label at the top containing their name, class, and stereotype. Lifelines are represented as a rectangular node containing the lifelines label, with a vertical line under it representing its lifetime. To improve recognisability, some stereotypes allow for an alternative notation of the lifeline's header. For instance, an actor can be visually represented by a stick figure replacing the lifeline node. The length of the lifetime represents the duration of life of the lifeline. Terminated lifelines are marked with an x at their destruction point.

Messages are depicted as horizontal arrows connecting lifelines, denoting the exchange of information or instructions. Messages can take on different forms, including synchronous (indicated by a solid arrow end), asynchronous (depicted by a lined arrow end), replies (represented by dashed lines), or self-referential (indicating a message sent from an entity

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

to itself). Reply messages signify the response from a calling entity back to the calling entity. These messages portray the flow of control as it returns to the originating lifeline. Messages of kind create and destroy messages, respectively create and terminate lifelines. The final differentiation of messages is by their type; lost and found messages opposing to completed messages do not have to have a lifeline as their source or target. The duration of a message can be visualized by drawing it diagonally rather than horizontally, with the height difference visualizing the duration.

Execution Specifications are vertical bars or rectangles atop a lifeline, signifying the temporal span during which an entity is actively processing a specific message. They effectively encapsulate the time when an entity executes a particular task. The representation of execution specifications is optional and open to the modeler.

Combined fragments offer a means to articulate conditions and constraints governing the sequencing of messages and interactions within the diagram. Its interaction operators, such as loops (for iteration) and conditional branches (for decision-making), define it with their corresponding interaction operands. The combined fragment may have guards, a Boolean expression represented within square brackets that guards an operand in a combined fragment.

InteractionUse references another sequence diagram within the current one. It simplifies complex diagrams and enhances their modularity. It is depicted as a rectangle with a frame labeled with the name of the referenced interaction.

2.2.2 Significance and Use Cases

SDs hold significant importance in software engineering as they provide a visual representation of system behavior, facilitating a clear depiction of interactions between various objects or components within a software system. This visual clarity is instrumental in understanding complex system behavior, facilitating communication among project stakeholders, and fostering collaborative design and development efforts. SDs play a role in the early detection of design flaws, helping to identify potential issues in the system's architecture and facilitating design improvements before implementation. Their significance lies in enhancing the overall quality of software systems by promoting systematic and well-structured development.

The use cases of sequence diagrams span various stages of the project development lifecycle. In the requirements analysis phase, sequence diagrams capture user interaction scenarios, thereby defining system behavior from the user's perspective. In the design and architecture phase, they aid in visualizing system components and their interactions, ensuring consistency in system design and promoting a comprehensive understanding of the system's structure. Furthermore, sequence diagrams find utility in testing and validation, as they can generate test scenarios, verify system behavior against specifications, and facilitate efficient debugging. They also play a vital role in documentation, helping create detailed system documentation that enhances system maintenance and knowledge transfer. Lastly, sequence diagrams can aid in performance optimization by identifying

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

bottlenecks and optimization opportunities, ultimately improving the efficiency of the software system [Micb, SSHK15].

2.3 Graphical Language Server Platform - GLSP

GLSP builds on the foundation of the Language Server Protocol (LSP) [BL23]. LSP and, in turn, GLSP are founded by a client-server architecture to separate concerns in the functionalities of an IDE. Previous monolithic IDE approaches handled all the required functionalities internally, resulting in large and complex software solutions.

2.3.1 LSP

"The Language Server Protocol (LSP) defines the protocol used between an editor or IDE and a language server that provides language features like auto complete, go to definition, find all references, etc. " [Micc]

Microsoft developed LSP for its Visual Studio Code IDE and was later 2016 standardized as an open standard in collaboration with RedHead and CodeEnvy [Mica]. Since its release, the expansion and adoption rate has consistently grown, with a clear upward trend. Over 200 language servers and nearly 50 language clients were developed [BL23]. The LSP was developed to push the modularity and simplicity of Visual Studio Code (VS Code). The IDE can be rendered language-agnostic while obtaining its language-specificity through the standardized protocol and dedicated language servers for each language. The upside is that ideally, for each language, one LSP suffices, which all IDEs can access. The LSP synchronizes client and server through standardized communication. The protocol consists of a bidirectional JSON-RPC. Besides Visual Studio Code, LSP is utilized by several other modern IDEs such as Eclipse IDE, Atom, and Theia [Mica].

2.3.2 GLSP

As stated, LSP is limited to textual content. However, the interaction with graphical models is increasingly relevant in software development and business modeling. Visual representations are more straightforward and faster to read and allow for more intuitive interaction. Graphical language development holds added complexity due to the rendering and manipulation of the graphic notation symbols. This motivated researchers to replicate the success of LSP in graphical modeling solutions [REIWC18]. One of the resulting solutions is the GLSP mentioned above.

GLSP is an open-source project hosted by the Eclipse Foundation and part of the Eclipse Cloud Development project [Ecla]. GLSP is an extensible open-source framework for building custom diagram editors based on web technologies [Ecl22]. GLSP builds on the same fundamentals of LSP, allowing the separation of language specifics and modeling software. The diagram client and server are separated and communicate using the defined protocol. The resulting flexibility allows the editors to be integrated into dedicated web

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

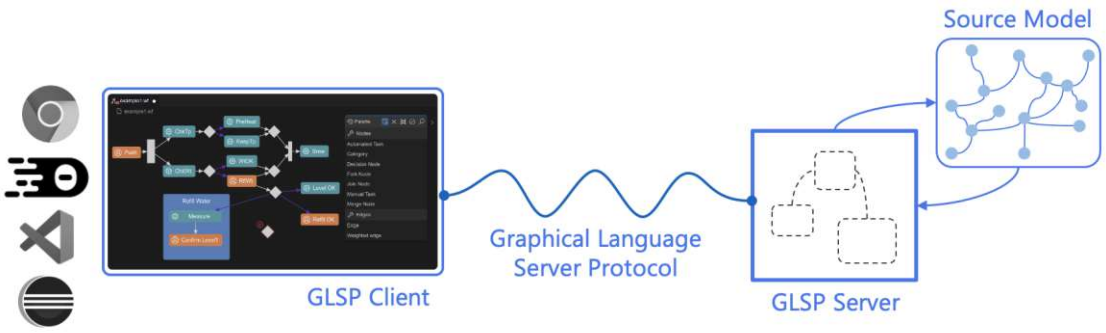


Figure 2.3: GLSP architecture overview [Ecla]

applications, common web IDEs such as VS Code, Eclipse Theia, and the traditional Eclipse desktop IDE [BLO23, MB23].

The GLSP architecture, as visualized on an overview level in Figure 2.3, consists of an extensible client framework, a server framework, and the communication enabling language server protocol (LSP) for diagrams.

The server framework handles the heavy lifting, such as loading, interpreting, and editing according to the rules of the modeling language. It is responsible for model management, the model logic, validation, and applying changes to the model(s). It supports the Eclipse Modeling Framework (EMF), based on which many modeling languages and their language-specific logic are already implemented.

The client framework handles the interaction of the user with the model. Interactions can create actions that are either handled by the client (zoom, move viewpoint, selection) or forwarded to the server for changes on the model (create elements, move elements, etc.). The results of changes on the server side are returned to the client as an updated model to be rendered. The diagrams are rendered using Eclipse Sprotty, SVG, and CSS, which allows for extensive customization of the feel and look of the tool.

The GLSP protocol is an extension of the LSP protocol, allowing communication over an extensible JSON-RPC between the client and server. Model interaction and changes are transmitted as actions enveloped in messages between the client and server.

The source model represents the data of the diagram being modeled and is the by the user ultimately created artifact in combination with its visual representation. The GLSP framework puts no restriction on the source formats, supporting a variety of EMF models, JSON files, databases, etc.

The openness and flexibility of the GLSP framework allow a wide possibility of use cases [BLO23], finding applications in commercial [Buc] and open-source projects [Imia].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

CHAPTER 3

Related Work

In this section, we discuss relevant related work. SD modeling tools-specific material is limited to [NPA16, Lim05]; the relevant material significantly increases by extending the researched material to UML modeling tools. Some of these works explicitly reference SDs as supported diagrams or evaluate their basic SD modeling functionality. The UML modeling tools-related works focus on different aspects, either the categorization and collection of available UML modeling tools, the evaluation of specific tools but their general metrics and modeling functionalities, or a combination of the two aspects.

The categorization and research of existing tools is a significant topic of interest and provides users and researchers with valuable datasets of possible tools. The logical follow-up question to an extensive list of tools is which tool to use. Therefore, evaluating the tools to identify the best tools to use and recommend has value to the community.

The literature review on modeling tools evaluations [KL22] analyzes the existing related works, including 41 papers from 2000 to 2020. The literature reviews the papers' evaluation methods and summarises the most frequent MBSE tools. Kalantari's work provides a valuable collection of the available UML modeling tools and highlights them with higher interest and relevance in the modeling and research community. The paper provides a summary of proposed and applied evaluation methods and the requirements for UML modeling tools, which will be taken as valuable input for this paper and used more specifically for SDs. Our work aims to expand existing works by focusing on a specific UML model, the sequence diagram (SD). We can evaluate the temporal modeling interactions particular to this model by focusing on the SD.

3.1 Modeling Tools Catalogs

The research on available modeling tools for SDs builds and extends the efforts of related works already establishing a collection of available UML modeling tools. The works

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[Ozk19], [EES11], and [ALS19] feature extensive collections of available general and UML modeling tools.

[Ozk19] features a tabular listing of 58 tools with the covered UML diagrams' specifications. The paper further analyses the tools for their feature support. For our initial cataloging purposes, the tool listing is of particular value. Eichelberger's updated version [EES11] of his original work [EES09] provides a collection of 62 UML modeling tools evaluated in detail on their compliance level with the UML specification, which provides a valuable quantitative comparator across the modeling tools. Angner provides listings of 32 [AL17] and 31 [ALS19] modeling tools from the perspective of its users, these being professors and students. The survey provides quantitative insights into utilizing individual tools in the academic setting in which the survey was conducted. In addition, the works survey the ease of use of the tools and the educational value and challenges of UML modeling. In addition to scientific research material, listings of possible UML modeling editors are published and maintained online.

3.2 Modeling Tools Evaluations

The works above already encompass the evaluation of the tools to some extent. This section lists related works that put increased focus on evaluating the tools. These works evaluate a reduced set of modeling tools through different means of evaluation [KL22].

The paper [PC20] analyses the usability of UML modeling tools by comparing two widely used UML modeling tools, MagicDraw and Papyrus. They apply an empirical approach by analyzing screen recordings of modeling behaviors for class diagrams. The paper looks into how students create models and how the tools support the task from three perspectives: modeling strategy, the modeling effort, and the modeling obstacles. The paper found no notable difference between the evaluated tools due to essential shortcomings in both tools. The paper offers potential improvements, which, in addition to addressing known issues, encompass the necessity for providing supplementary guidance to new users.

In the work of Ṇikiforova et al. [NPA16], the authors conduct an in-depth analysis of existing SD modeling tools focusing on the SD element's layout and the automatic layout functionalities. The authors defined a summary of twelve criteria defining a "good" diagram layout, Based on which the modeling tools were evaluated. The disappointing evaluation results highlight the need for better layout functionalities in sequence diagrams.

Funes et al. [FDSP05] differentiates its work from the mentioned works as it focuses on defining a clear structure of requirements for evaluating UML modeling tools. The paper recognizes the growing number of available tools and the need for means of evaluation. The designed evaluation requirements encompass features a UML tool should support. These reach from general functionalities as import/export features to UML diagram-specific ones.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

3.3 Modeling Tools Implementations

This section lists existing works documenting previous efforts at implementing new modeling tools and modeling behaviors. The following works provide early insights into necessary considerations and possible challenges of implementing a modeling tool.

Limyr [Lim05] provides valuable insights and perspectives specific to SD modeling behaviors. The author systematically analyzes the editing operations of a sequence diagram editor. Limyr provides a foundation for possible improvements in modeling behavior. These can be further refined and possibly implemented as part of this work thanks to the increased flexibility and capabilities of the development stack.

The paper by Auer et al. [ATB03] presents the modeling tool UMLet. It consists of a flyweight modeling tool that provides intuitive and explorative modeling to create UML sketches quickly. The interface features a novice approach by providing a palette of diagram elements, which can be dragged and dropped into the diagram. This is combined with textual modeling functionalities. The paper describes the design considerations of their tool and outlines the motivations to provide a flyweight tool over a fully loaded heavy-weight modeling tool.

3.4 GLSP development and implementation

In addition to researching and evaluating the current availability of modeling tools, we found existing research on the development and implementation of solutions and extensions utilizing GLSP.

The paper by De Carlo et al. [DCLB22] builds on the Eclipse GLSP platform through advanced model visualization and interaction functionalities. The work presents a concept and a prototypical implementation of Semantic Zoom and Off-Screen Elements. "We believe such advanced functionalities pave the way for a prosperous modeling future and spark innovation in modeling tool development." [DCLB22, 1]

The paper by Bork et al. expresses the vision for more flexible modeling tools that offer the feature-rich functionalities of established modeling tools while building on web technologies that provide added flexibility regarding user experience, accessibility, reuse, and deployment. The paper presents the potential of GLSP as a platform for developing modeling tools. The paper provides a technical foundation of GLSP and focuses on its flexible application [BLO23].

The paper by Sarioğlu et al. identifies the lack of accessibility in available modeling tools and envisions more inclusive and disability-aware conceptual modeling. Motivated by their research on existing tools and their lack of accessibility, they expand on the Eclipse GLSP platform through the development of keyboard-only modeling behaviors and interaction to also include users with physical disabilities to engage in conceptual modeling [SMB23].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 4

Modeling Literature & Tools Analysis

There is a wide selection of existing tools for the creation of UML diagrams, from legacy software to modern web tools. This chapter aims to collect and provide an overview of the currently available tools for modeling SDs. This involves consulting existing works cataloging UML modeling tools [Ozk19, EES11, ALS19, KL22], and research additional sources online [Wik23, Jec, Cab22]. Table 4.1 lists various cataloging sources of UML modeling tools.

Source	Quantity of cataloged tools
[EES11]	62
[Ozk19]	34
[AL17]	32
[ALS19]	31
[Wik23]	48
[Jec]	102
[Vpj22]	30
[Cab22]	20+

Table 4.1: Modeling tools catalogs

The merging of the mentioned sources in table 4.1 resulted in a heterogeneous list of over 100 UML modeling tools. This collection forms the foundation for selecting relevant tools for further analysis but requires further filtering of tools to ensure tool relevance to our work. At this stage, the collection contains a heterogeneous selection of modeling tools, not all meeting our relevance criteria, defined in section 4.1. The tools are of different types, graphical and textual. They are created with other use case purposes as general drawing tools for sketching or semantic correct modeling. The tools vary in recency.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Some of these are legacy systems that have been outdated for years, while others are actively used and developed. Finally, not all tools of the collection have support for SD. A series of assertions and restrictions on the tools are required to provide a list of relevant modeling tools.

4.1 Tools Criteria and Categorization

To research and organize the available relevant modeling tools, we established a selection of criteria. We categorize and select the tools based on their support for relevant diagram types, the tools' recency and currently available user support, the public accessibility to the tool, the typology of the tool, and the tool's adherence to syntax specifications such as UML.

4.1.1 Supported diagram types

Modeling tools support various types of models/diagrams, 14 of which are specified by the UML specifications. While most UML-specific modeling tools support all these 14 diagram types, some only cover the most common and more straightforward diagram types, such as class diagrams. Since this work handles the modeling interactions of SD, all tools lacking support for SD can be regarded as not relevant to this study.

4.1.2 Modeling tool recency

Tool recency indicates when the tool was last updated through development efforts, the date of the most recent release, and the last supported software environment and hardware. During the research of modeling tools, many tools were found to be outdated, meaning used in the past were not maintained over time and fell into a natural state of degradation, are not supported by current hardware and software anymore, were officially dismissed or replaced, or wholly vanished including their vendor's/developer's page. For example, tools relying on Flash lost support due to the developing stack and architecture not being maintained. This created some difficulty in understanding whether the tools themself were still relevant in the community. The fast evolution of software environments and the missing updates of modeling tools motivated the addition of the latest tool version and the last update date available to our tools list. This transparently communicates the recency of the tool to the reader and supports the possible tool selection process.

4.1.3 Modeling tool public accessibility

While most tools are publicly available through their open-source license, free trial license, or freemium license, some do not offer free access. This was mainly the case for the products from IBM, which are targeted at enterprises rather than consumers and provide no public access. This can be seen as a limiting factor to this work.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

4.1.4 Modeling tool syntax

The researched tools vary in their adherence to the UML syntax. The tools range from "unrestricted drawing tools" to "syntactic accurate modeling tools."

Some tools only provide elementary shapes representing lifelines, messages, combined fragments, etc., which graphically follow the UML notation. The user freely arranges these elements on the canvas without any assistance from the tool to generate syntactically valid diagrams. These "unrestricted drawing tools" provide no constraints regarding their use or positioning concerning the UML grammar rules, giving the user complete freedom with full responsibility for creating a grammatically correct diagram. These drawing tools tend to support various diagrams, not only UML. They attract a broad user base, motivating the reliance on web technologies and justifying the implementation of additional features such as online real collaboration. The advantage of these tools lies in their open distribution and ease of use. Most run within a browser for online modeling and are platform-unspecific, working on most devices. Such model drawings are usually quickly assembled and serve as sketches and means of visualizing the given model. Often, they are utilized as support for planning or as documentation material. The drawback of these tools is the lack of a semantic model for technical application.

Alternatively, more specialized tools follow the syntactic UML specification [Gro17], which ensures the creation of models accurate to the syntax. The models consist of the visualization of the model itself and an underlying model defined explicitly by a standardized modeling language. These models can be utilized for visualization and documentation. Still, they can be applied actively in the development process with methodologies such as code generation out of the model, opening up additional use cases. Regarding the syntax correctness of the model, better guidance was provided by the editor, which provided clear rules for the use of the SD components. These rules constrain the movement of elements and the correct dependency of components. The level of constraints and the layout guides vary across editors of this typology. "Syntactic accurate modeling tools" are defined by their underlying model, usually stored as an XML. Their applications go beyond the sole visualization of a graphical model, allowing them to take a central role in the developing process.

4.1.5 Modeling tool typology

The final aspect relevant when researching UML modeling tools is the tool type that defines how users interact with the model. The tools are categorized into textual modeling tools and graphical modeling tools. Textual modeling tools generate diagrams based on plain text language, with the most utilized language being PlantUML as depicted in Figure 4.1. In most textual modeling tools, users only interact and edit the diagram's textual description, which defines the visual representation. Interaction with the visual diagram is not possible.

In contrast, graphical modeling editors, as shown in Figure 4.2, require the users to interact with the diagram itself and not with the underlying files. They usually provide

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

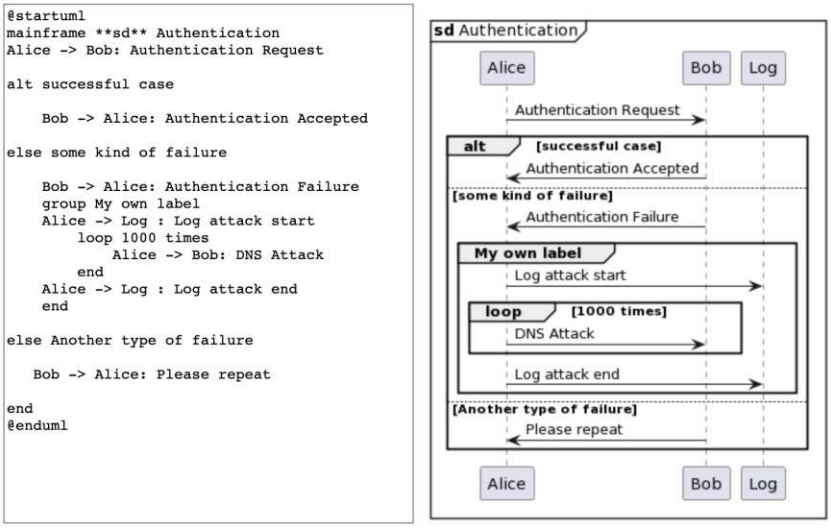


Figure 4.1: SD example in PlantUML

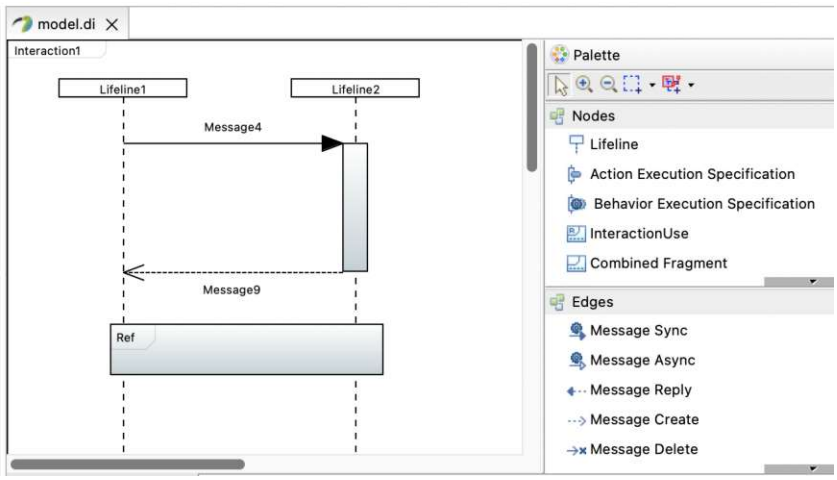


Figure 4.2: SD example in Papyrus

a palette of elements and tools the user can interact with to create the desired diagram. The components consist of nodes and edges that can be placed on the modeling canvas. The editing tools allow the user to navigate the diagram and interact with diagram components.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

4.2 Existing sequence diagram modeling tools selection criteria

To provide a valuable catalog of relevant modeling tools, an initial selection of possible tools was conducted based on tool base requirements in the tool as mentioned above variations. The following criteria were selected.

Tool availability: The tools should be available to the public either freely, as is the case for open-source software, or provide a trial version for a commercial product. The tools cannot be a company's internally developed tool or tools restricted to enterprise licensing with limited or no public access. Only publically available tools are evaluated.

Diagram type support: From the tool description, it should be clear that support for SD is given.

Popularity within the community: The tool has to be known and adopted within the domain by being mentioned in multiple in Table 4.1 mentioned tool catalogs, featuring an active following on development sites such as GitHub or showing a wide adoption rate through downloads or active users.

Hardware and Software support: The tool must be supported by generally available hardware and operating systems. More specifically, it should function on the current version of either macOS(12.6+) or Windows(10+) on their respective supported computers.

Recency and support: To avoid outdated tools, the last release should be within the last ten years.

Tool typology: Since this thesis is interested in the interaction behaviors and usability of these tools, we consider tools of different natures. These include tools that only focus on the graphical modeling of SD, as well as tools that create the underlying semantic model. We included "constrained drawing tools". While these do not provide a model, they give intuitive and practical interaction methodologies. They are built on web technologies and employ the new interaction method.

4.3 Tabular Summary

After filtering the catalog of modeling tools based on the criteria mentioned above, we end up with the following table of available diagram editors. We found 28 modeling tools that match our requirements and could qualify for the evaluation in the next chapter. These selected models have the highest overlap across the listed cataloging efforts and, therefore, can be assumed to be the most commonly utilized.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Table 4.2: Collection of available modeling tools

#	Tool	Release	Version	Deployment	Availability	OS	Typology
1	**Astah UML**	06.2023	9.1	local	commercial, free student license	Windows, Linux, MacOS	modeling
2	Cacoo	-	-	web	trial and commercial	-	drawing
3	Creately	-	-	web	free and commercial	-	drawing
4	**Draw.io**	-	-	web	free	-	drawing
5	edrawmax	07.2023	12.5.1	web, local	free and commercial	Windows, Linux, MacOS, iOS	drawing
6	**Enterprise Architect**	-	-	local	trial and commercial	Windows	modeling
7	**GenMyModel**	-	-	web	free	-	modeling
8	Gliffy	-	-	web	free and commercial	-	drawing
9	**Lucidcharts**	-	-	web	free and commercial	-	drawing
10	**MagicDraw**	12.2021	2021x	local	trial and commercial	Windows, Linux, MacOS	modeling
11	Mermaid	08.2023	10.3.1	web	open source	-	textual
12	miro	-	-	web, local	free and commercial	Windows, Linux, MacOS, Android, iOS	drawing
13	Modelio	03.2023	5.3.1	local	open source	Windows, Linux, MacOS (up to 4.1.0)	modeling
14	**Papyrus**	06.2023	6.5.0	local	open source	Windows, Linux, MacOS	modeling
15	PlantUML	07.2023	1.2023.10		open source	-	textual
16	Rational Software	07.2022	9.7.1.1	local	commercial	Windows, Linux, MacOS	modeling
17	sequencediagram.org	03.2023	9.6.0	web	free	-	textual
18	**smartdraw**	-	-	web	free and commercial	-	drawing
19	**StarUML**	01.2023	5.1.0	local	trial and commercial	Windows, Linux, MacOS	modeling

Continue on the next page

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Table 4.3: Collection of available modeling tools (cont)

#	Tool	Release	Version	Deployment	Availability	OS	Typology
20	**Tracemodeler**	2008	1.6.2	local	commercial	Windows, Linux, MacOS	modeling
21	Umbrello UML Modeller	12.2022	2.37	local	open source	Windows, Linux, MacOS	modeling
22	UMLet	03.2023	1.15.1	local	free	Windows, Linux, MacOS, VS Code, Eclipse	drawing and textual
23	UMLetino	03.2023	15.1	web	free	-	drawing and textual
24	**UModel**	04.2023	2023r2	local	trial and commercial	Windows	modeling
25	**Visio**	-	-	web, local	freemium/ commercial	Windows	drawing
26	**Visual Paradigm**	05.2023	17.1	local	trial and commercial	Windows, Linux, MacOS, VS Code, Eclipse, IntelliJ IDEA, NetBeans	modeling
27	**Visual Paradigm Online**	-	-	web	free and commercial	-	drawing
28	websequencediagrams	-	-	web	free and commercial	-	textual

In the list, we differentiate between web-based tools and desktop applications. Web-based tools are modeling tools that run within the browsers and are provided as a service by their vendors. Web modeling tools require no setup on the local machine, meaning they can be accessed from most devices with browser support, and users can start modeling immediately. Overall, there appears to be a clear trend toward using browser-based editors. However, most of the widely used editors are unrestricted drawing tools [Lim05], meaning they employ elements of the diagram type but don't ensure syntactically correct diagrams. Desktop application-based modeling tools require download and installation on the local machine, with operating system and hardware requirements. The installation process on the device can cause significant additional effort for the user and could raise issues even before being able to model. This limitation was also encountered for some of the tools, which could not be evaluated. The evaluated tools were further selected regarding popularity, availability, and current support, resulting in a collection of 14 tools, highlighted in bold in table 4.2.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 5

Existing Tools Evaluation

The following chapter aims to provide a detailed insight and understanding of each tool's modeling functionalities and behaviors. The tools are evaluated based on the modeling process defined in this chapter to report on a comparable modeling experience. A comprehensive library of modeling tools supporting sequence diagrams is established within the industry. However, a clear industry-leading product cannot be identified, as this strongly depends on the individual use case and preferences.

We differentiate between the core modeling features of the tool and features and behaviors that affect the modeling of and interaction with the diagrams. The general features ease the use of the software itself as well as the integration of its use in a workflow. Examples of such features are the export/ import functionalities and cooperation on a model. For completeness' sake, the support for such features is added to the categorization process as it is experienced throughout the research but is not the focus of this work. The features of interest are the editing abilities of the modeling tool itself, particularly:

- How does the user interact with objects in the diagram?
- What constraints are in place to guide the modeling process?

An exemplary SD modeling process is created and applied to the SD modeling tools to achieve an objectively comparable evaluation of the individual SD tools and categorize their behaviors. The modeling process contains all the steps to create individual components to achieve the final diagram. However, a strict modeling process might not be representative of the modeling behavior in a practical, real-world application. Modeling does not simply consist of replicating a provided diagram but involves additional intermediate steps brought up by its explorative and iterative nature. The diagram may change dynamically throughout its creation process; individual elements might be moved, replaced, or removed. Following this reasoning, the strict modeling process is extended

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

with additional intermediary steps, such as removing, moving, scaling, and replacing existing elements to replicate a natural user's behavior and consider a broader range of user interactions. The modeling process serves to evaluate the usability of the individual modeling tools objectively. It ensures consistent testing of the functionalities across all the tools and systematically uncovers their differences.

5.1 Modeling Process Definition

This section defines the modeling process, which is applied throughout the tool evaluations. To ensure a consistent and replicable modeling process, we have to define its expected outcome, in our case consisting of an SD, as well as the steps required to model it.

5.1.1 Model Outcome

The resulting SD has to meet specific requirements, the first of which is to contain all critical components defined by the UML specification of SD. The most essential elements of an SD as defined by Seidl [SSHK15] consist of *Lifelines, Destruction events, Combined fragments, Synchronous messages, Response messages, Asynchronous messages, Lost messages*, and *Found messages*.

These are the most commonly utilized elements and cover most of the applications of sequence diagrams. However, to model more complex interactions, the UML Specification provides additional elements which in turn should be supported by the tool: *Interactions, Create messages, Execution Specifications, Time-consuming messages / Messages with a duration, Interaction operands, InteractionUses*, and *Gates*.

As a starting point for our reference SD utilized in the evaluation of the modeling tools, we selected the example diagram from Figure 5.1 illustrating the Spring and Hibernate transaction [Faka]. The diagram adheres to the OMG UML 2.5 specification and illustrates transaction management with exception handling. Textually, the diagram describes the following process:

> "When some business method is called, it could be intercepted by Spring Transaction Interceptor. This behind the scene interceptor creates Hibernate session and starts Hibernate JDBC transaction, so that business method will run in the context of a new transaction. Business method execution could complete (successfully or not) without throwing any exception, or by throwing some Java runtime (unchecked) exception or some business (checked) exception. If business method did not throw any exception or thrown some business (checked) exception, transaction interceptor will try to commit transaction. When Hibernate JDBC transaction is flushed (to store data permanently), this operation might fail, e.g. because of some database constraint violation. In this case transaction will be rolled back (even though business method execution was successful). If business method thrown some

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

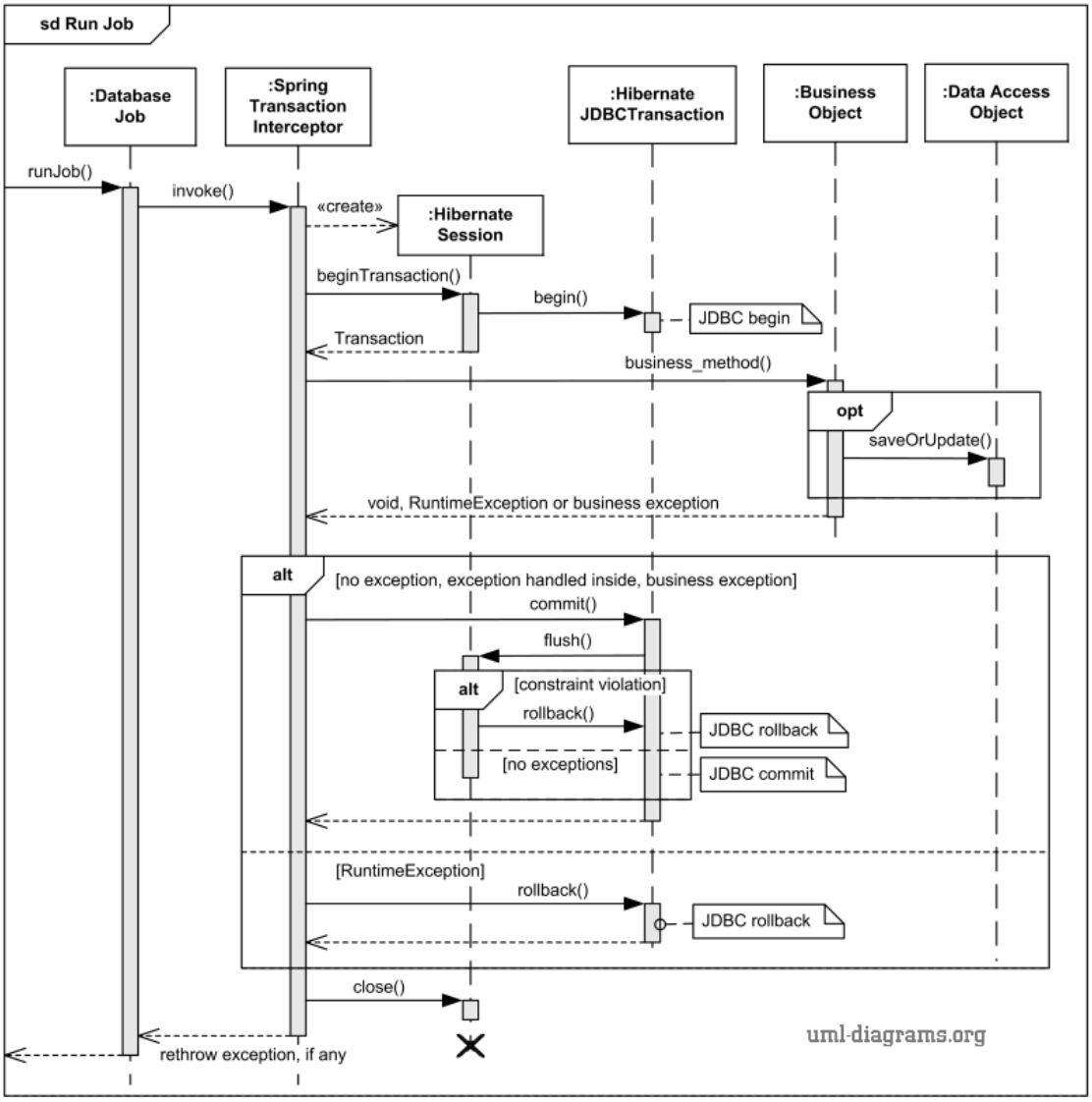


Figure 5.1: Spring and Hibernate Transaction example from [Faka], created with Microsoft Visio 2007-2016 using UML 2.2 stencils

> Java runtime (unchecked) exception, it is an indicator that business method failed, and transaction interceptor will rollback the transaction. At the end, Hibernate session will be closed by the Spring Transaction Interceptor." [Faka]

We selected this diagram since such transactions between multiple objects greatly benefit from a visualization with an SD and are a common use case for SDs. The added complexity of the exception handling makes this a suitable candidate for our evaluation. However, to evaluate all the mentioned components of an SD, we are expanding the reference diagram

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

with the missing diagram components. The diagram requires the addition of an instance of the following components: *Asynchronous message, Message to self, Lost and found message, Message with a duration, Duration constraint, Interaction use*, and *Lifeline stereotype notation*.

By extending the original diagram in Figure 5.1, we obtain our reference SD Figure 5.2, representing the expected outcome of the modeling process in our evaluation.

5.1.2 Modeling Behaviors

The modeling process used to assess the tools concludes upon the attainment of the resulting SD. To do so, a user could choose many ways to achieve the same result by different means. The user can follow the most efficient modeling approach regarding time and amount of interactions/clicks if the outcome is already well defined and the user is familiar with the tool. If, however, the resulting SD is not predefined at the time of modeling, a more iterative modeling behavior is applied, which results in additional steps or model variations throughout the modeling process. These consist of additional *creations, selection, deletion, scaling*, and *renaming* of diagram elements. Our final modeling process consists of a sequence of necessary steps with added optional modeling behaviors to represent a user's real-world modeling behavior better.

5.1.3 Complete Modeling Process

By combining the defined resulting SD with the required and additional modeling behavior, we define the complete modeling process. For each component, we need at least one creation behavior.

The modeling process might vary depending on the tools support and handling of components. Before setting up and conducting the modeling process for evaluation, we must recognize that some editor functionalities can significantly impact the modeling experience, requiring addition to the process. The significant influence on the modeling process is the tools handling execution specifications and response messages. Some tools create execution specifications and response messages automatically by creating synchronous messages. Alternatively, these have to be added manually.

As a starting condition for the modeling evaluation, each instance starts with a new project and a blank canvas. Table 5.1 lists all required steps to create the defined reference model in Figure 5.2 with the addition of the mentioned supporting interactions.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

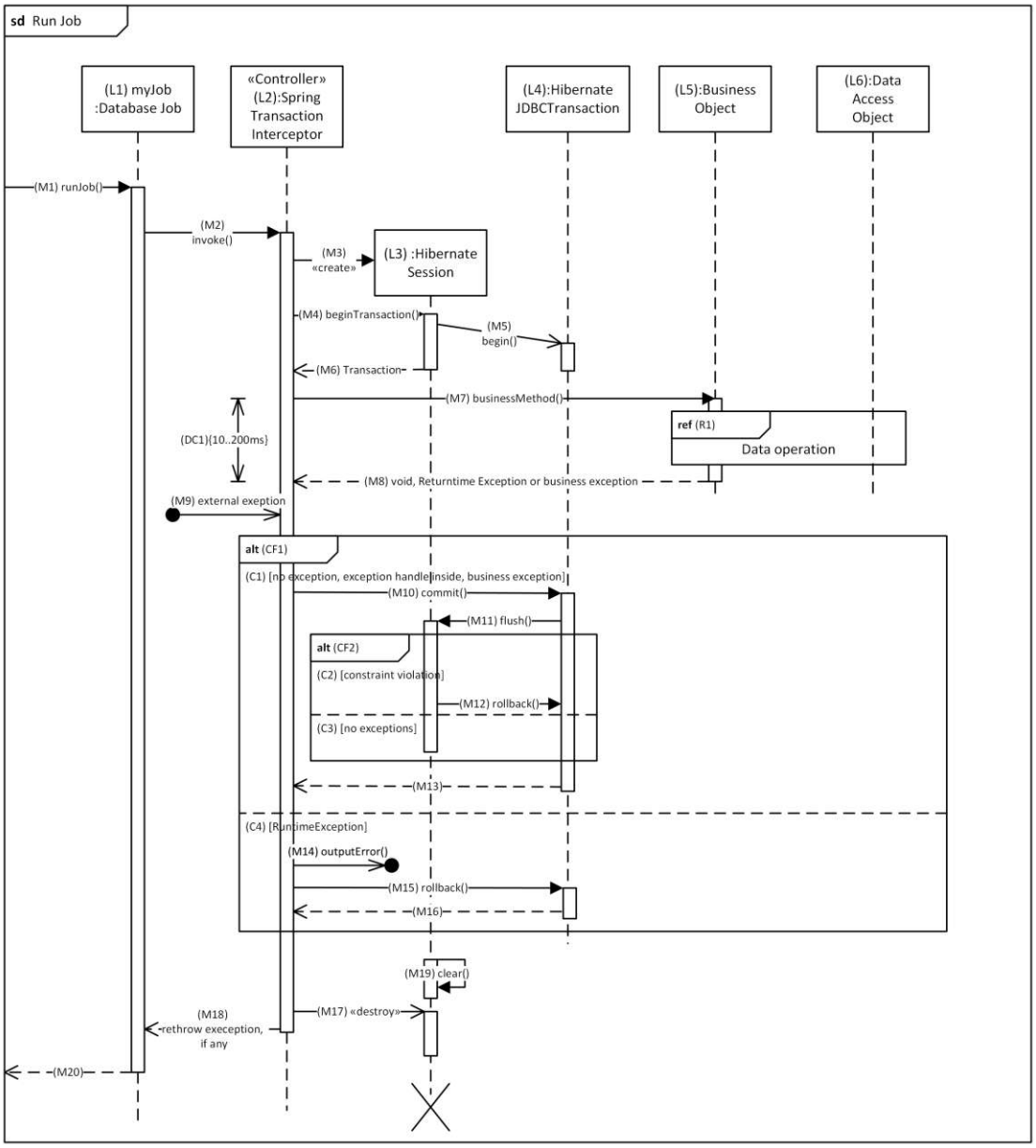


Figure 5.2: Adapted Spring and Hibernate Transaction example created with Microsoft Visio 365 using UML 2.5 stencils

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Table 5.1: Modeling process

#	Comp.ID	Comp. Type	Type	Description
1	SD	model	create	Create new sequence diagram
2	I1	interaction	create	Create interaction(I1) named "runJob"
3	L1	lifeline	create	Create a lifeline(L1) named "myJob_temp" with class "Database Job"
4	L1	lifeline	edit	Edit lifeline(L1) name to "myJob"
5	L1	lifeline	move	Move the lifeline L1 horizontally, and then back to its position
6	L1	lifeline	move	Move the lifeline L1 vertically, and then back to its position
7	L1	lifeline	scale	extend and shorten the length of the lifeline
8	L3	lifeline	create	Create a lifeline(L3) with class "Hibernate Session" to the right of L1
9	L2	lifeline	create	Create a lifeline(L2) with class "Spring Transaction Interceptor" and stereotype "Controller" to the right of L3
10	L2, L3	lifeline	move	Move L2 to the left of L3
11	L4	lifeline	create	Create a lifeline(L4) with class "Hibernate JDBCTransaction"
12	L5	lifeline	create	Create a lifeline(L5) with class "Business Object"
13	L6	lifeline	create	Create a lifeline(L6) with class "Data Access Object"
14	M1	message	create	Create the synchronous message(M1) named "runJob" from the interaction gate to lifeline "myJob"
15	M2	message	create	Create the synchronous message(M2) "invoke" from L1 to L2
16	M3	message	create	Create the creation Message(M3) from L2 to create lifeline L3
17	M7	message	create	Create the synchronous message(M7) named "businessMethod()" from L2 to L5
18	M4	message	create	Create the synchronous message(M4) named "beginTransaction" from L2 to L3 above of M7
19	M5	message	create	Create the asynchronous message(M5) with duration named "begin()" from L3 to L4

SD = model, I = interaction, L = lifeline, M = message, CF = combined fragment, DC = duration constrain, IU = interaction use

Continue on the next page

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Table 5.2: Modeling process (cont)

#	Comp.ID	Comp. Type	Type	Description
20	IU1	interaction use	create	Create the interaction use (IU1) referencing another interaction named "Data operation". Create the additional interaction if necessary.
21	M9	message	create	Create the found asynchronous message(M9) named "external exception" from L2 to L3
22	DC1	duration constraint	create	create a duration constraint (DC1) between M7 and M8 with duration 20..100ms
23	CF1	combined fragment	create	Create combined fragment(CF1) of type "alt" over L2, and L3
24	CF1	combined fragment	move	Move the combined fragment CF1 horizontally
25	CF1	combined fragment	move	Move the combined fragment CF1 vertical
26	CF1	combined fragment	scale	Widen CF1 to also cover L4
27	M10	message	create	Create the synchronous message(M10) named "commit()" from L2 to L4
28	M11	message	create	Create the synchronous message(M11) named "flush()" from L4 to L3
29	M12	message	create	Create the synchronous message(M11) named "rollback()" from L3 to L4
30	CF2	combined fragment	create	Create combined fragment(CF2) of type "alt" over L3, and L4, containing M12
31	C2, C3	operator	create	Add a second operator and rename the conditions (C2,C3)
32	C4	operator	create	Add a second operator to CF1 and rename the condition(C1) to "RuntimeExcepetion"
33	M18	message	create	Create the destruction Message(M17) from L2 to L3
34	M17	message		Create the asynchronous Message(M17) to sef on L3
35	M7	message	delete	Delete message M7
36	M7	message	undo	Undo the deletion of M7

SD = model, I = interaction, L = lifeline, M = message, CF = combined fragment, DC = duration constrain, IU = interaction use

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

5.2 Modeling Process Evaluation Criteria

The following section defines questions based on which the interaction with the sequence diagram modeling tools is evaluated. In the previous chapter, Table 5.1 described each step in the creation process of the reference SD shown in Figure 5.2. In this section, we define a set of questions referencing individual process steps in Table 5.3. These questions are meant to guide the reflection on the behaviors and interactions of the editing tool during the modeling process. The answers to the questions can be expanded with additional observations regarding the modeling behavior for each step.

Table 5.3: Evaluation questions on interaction and tool behaviors

Step	Question
3	Is the role:class naming scheme supported?
	Are multiple types of lifelines supported?
4	Are labels editable on the diagram or in the properties?
5	Is it possible to move lifelines horizontally?
	Is the movement fluent or in steps (grid)?
6	Is it possible to move lifelines vertically?
	Is the movement fluent or in steps (grid)?
9	Is the stereotype notation supported?
10	Do the lifelines overlap? Is the overlap automatically resolved (by moving L3 to the right)?
14	Are response messages (M18) automatically created?
	Are execution specifications created and visualized?
	Can execution specifications be added manually?
	Are the function brackets "()" added automatically or manually?
	Where can messages be created?
	Is the creation location constrained?
	Does the position follow a grid/row layout?"
16	Does the created lifeline align to the message height?
18	Does the creation of the message cause the following elements to shift down?
	Does the tool create additional space for the new element between existing elements?
	Is there a margin constraint/enforcement between the existing and the newly created message?
19	Are messages with duration supported by the tool?
23	Does the combined fragment interact with other elements of the diagram?
	Is the covering of the lifelines recognized?
24	Are there constraints in the movement?
	Does the movement move other elements?
	Does the CF1 movement minimize the overlay of objects?

Continue on the next page

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Table 5.4: Evaluation questions on interaction and tool behaviors (cont)

Step	Question
25	Are there constraints in the movement? Does the movement move other elements? Does the CF1 movement minimize the overlay of objects?
26	Is the change in lifeline covering recognized?
30	Do combined fragments drawn over messages also contain them?
33	Does the target lifeline scale back to the height of the destruction message? Does L3 automatically receive a destruction symbol at its end? How does the tool handle M17 below M18?
34	Does the tool support messages to self?
35	Does the deletion of the messages cause a vertical movement of the other elements? Is the freed-up space reduced?

5.3 Tools Evaluation

In this section, we evaluate the individual tools by conducting the modeling process mentioned above and collecting our observations following the established questions. Observations of behaviors going beyond the fixed questions are noted to ensure the collection of all implemented approaches. This chapter features the most interesting interaction behaviors uncovered during the evaluation process.

5.3.1 Astah

Astah provides a medium level of modeling interaction and dynamic modeling support. It features toggleable execution specification visibility, togglable interaction frame visibility, and the option for a grid. The resulting SD from our evaluation is shown in Figure 5.3.

Lifelines support the assignment of class and stereotypes with respective stereotype iconography. In their movement, lifelines are vertically locked and horizontally free to move with no overlap resolution. To support lifeline layout, the editor features guides showing symmetric lifeline spacings. In the case of lifeline creation, the lifeline appropriately follows its creation message.

The editor automatically adds execution specification and, by default, the return message of a newly created asynchronous message. Messages define the position of their execution specification. Messages are vertically freely movable with no regard to overlap. The editor has no support for messages with duration. Moving a message also moves messages on its execution specification, reducing efforts for the user. It does, however, not move all subsequent messages. For such interaction, Astah provides "gap mover" and "gap expander" tools, which, according to their name, adapt to vertical and horizontal gaps by dragging/pushing all components towards or away from the gap. Messages can also

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

have their target and source reassigned. Execution specifications can be scaled but are limited to their content size. Removing execution specifications also removes their related messages. Destruction messages must be the last component, and the destructed lifeline is destroyed at its height with the destruction symbol.

When placing, moving, or scaling the components interaction use and combined fragments, the editor assists the user by highlighting the covered lifelines. Combined fragments cannot be moved horizontally, are constrained by their covered lifelines, and scale with lateral movement. The vertical movement of combined fragments is unconstrained and moves their containing message ends. The target lifeline scales back to the height of the destroyed message. L3 does not automatically receive a destruction symbol at its end.

A unique feature of Astah is the visualization of offscreen lifeline names when vertically cut off and on cut-off messages from and to them as seen in Figure 6.2b.

Overall, Astah offers a fairly responsive modeling experience with some automation to reduce the user's workload, such as creating execution specifications and response messages. However, the overlap of lifelines and messages is not hindered, and it is the users' responsibility to curate the diagram's layout and readability.

5.3.2 Draw.io

Draw.io is an unrestricted graphical tool without regard for the UML SD syntax. It provides a rudimentary set of shapes matching the UML SD notation but none of the syntactic guides or constraints. Messages are not required to connect to lifelines. Instead, they can be anchored to any shape. Some anchors are available on execution specifications to connect messages to. The editor provides a canvas on which the shapes, such as execution specifications, lifelines, and combined fragments, can be placed and aligned freely by the user. The containment relation has to be configured manually for individual shapes. A shape representing a combined fragment does not explicitly exist and, therefore has to be constructed by the user.

Draw.io shows limited support for SDs-specific shapes, such as the combined fragments, lost messages, and duration constraints, which are not included in the tool palette, shown in Figure 5.4. Modifying an existing diagram, such as moving messages or a lifeline, results in a significant amount of adjustment efforts by the user. Overall, modeling sequence diagrams using Draw.io is work-intensive due to missing SD-specific shapes and the high amount of required manual shape alignment.

5.3.3 Enterprise Architect

Overall, the tool features a well-designed interaction. It provides intuitive layout support to ensure a well-modeled diagram. The editor visualizes the names of the lifelines above them, which is especially useful when viewing a lower part of the diagram and they are cut off.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

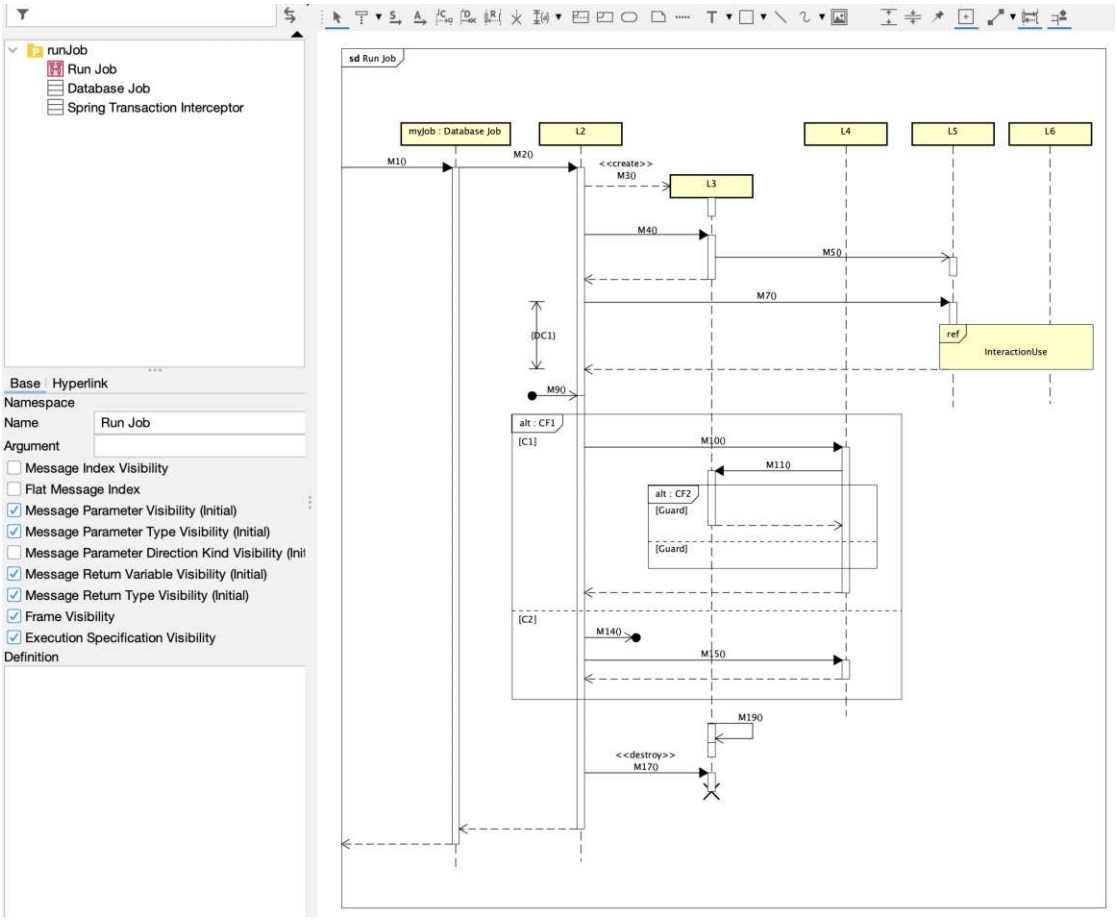


Figure 5.3: Astah

The movement and dependent movement of messages and components are thought out and well-implemented in the tool. Moving a message by default does not allow the message to be reordered or overlapped. This means the upward movement is limited by the message above it, while downwards it is freely movable. All components below follow the message movement. Reordering of messages is possible by holding a modifier key (Shift) while moving them. The editor mitigates the overlap of components by moving overlapping components and not allowing multiple messages at the same height.

A novice approach is the implementation of two different movement modes specific to combined fragments. The user can toggle between the 'freely move' mode and the 'content move.' In 'freely move' mode, combined fragments can be moved in any direction without affecting other components, just like a graphical overlay. Alternatively, the 'content move' mode allows only restricted vertical movement with no reordering of messages, but contained components and components below it move accordingly.

New users could require additional guidance or experimentation, as not all components

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

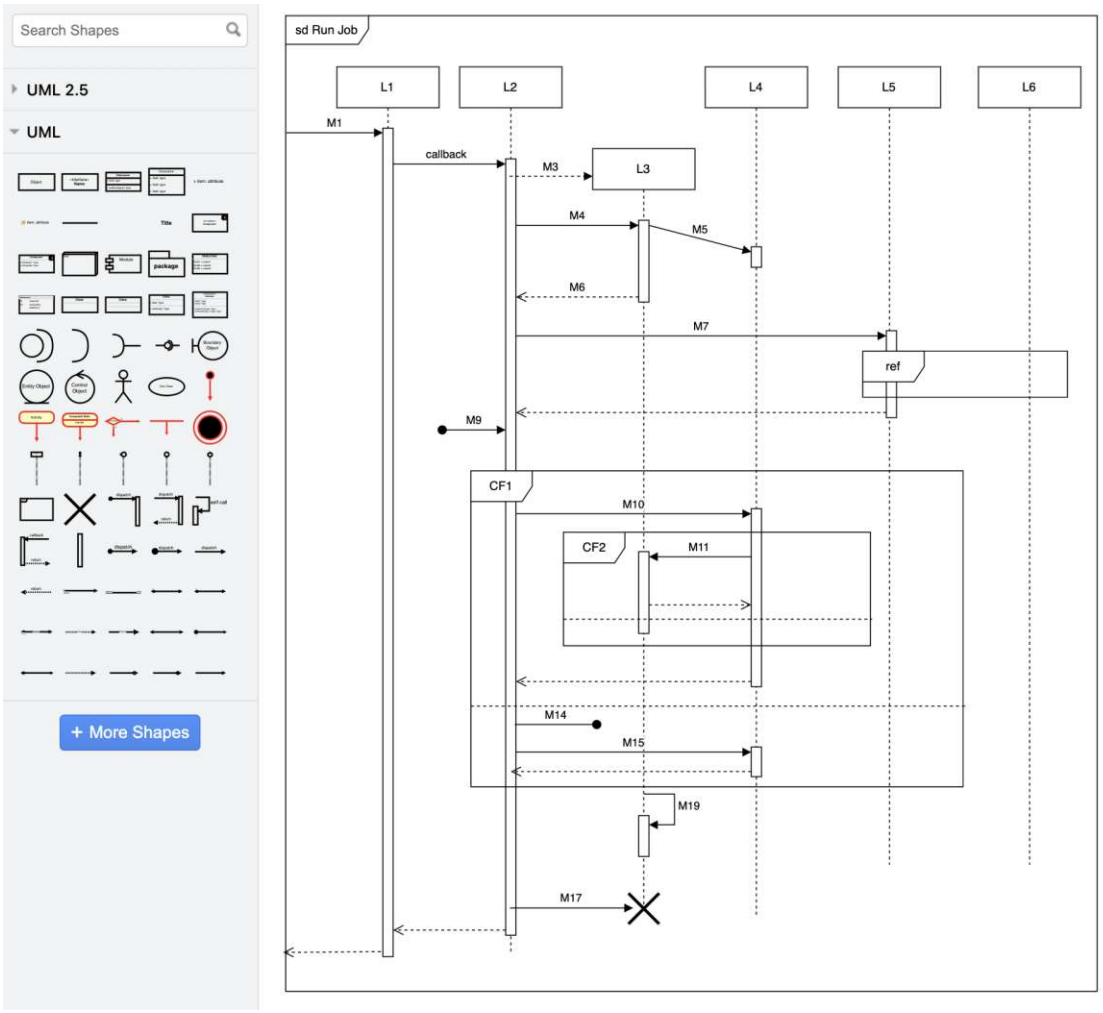


Figure 5.4: Draw.io

are directly accessible through the toolbox. Some features are accessed through the properties panels of components, such as the addition of operands to a combined fragment. The interaction use component is also not created through the tools palette but instead by dragging the referenced interaction from the context tab onto the diagram canvas. This can make it difficult to find specific actions and whether they are supported.

Beyond our use of modeling SDs, the tool provides a large selection of additional functionalities, such as simulation and analysis tools, which, depending on the user's needs, might be required features.

In conclusion, the tool provides an extensive feature set that supports an organized layout and an efficient modeling experience. It provides good support and tools to ensure an organized and readable diagram layout as shown in Figure 5.5. However, the tool is aimed more toward experienced modelers, as it requires additional time to master due to

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

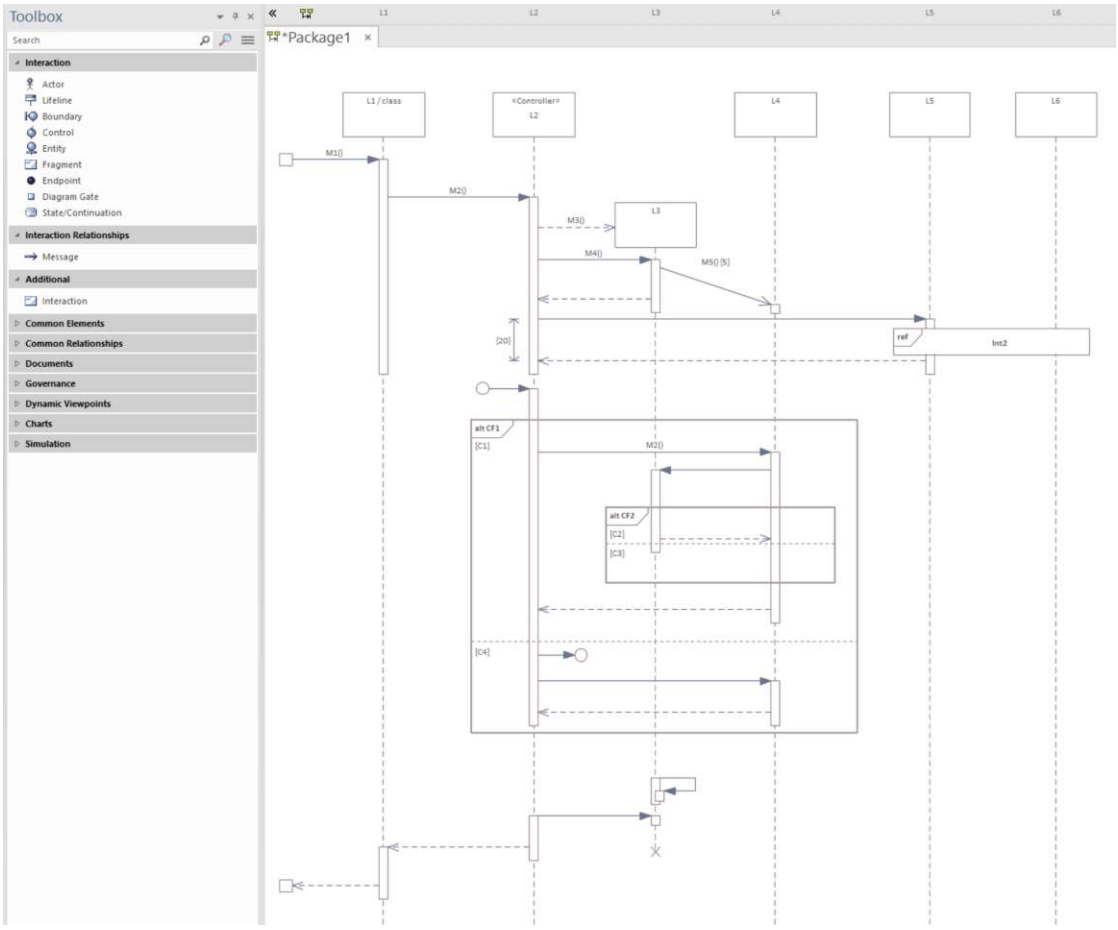


Figure 5.5: Enterprise Architect

its added complexity.

5.3.4 GenMymodel

The tool supports a subset of the component of a sequence diagram as shown in the palette in Figure 5.6. The tool employs some restrictions as defined by specifications, such as fixing the lifeline header height. The combined fragments don't seem fully implemented, as they do not interact with the rest of the model. This behavior aligns with the overall modeling experience, as there is little interaction and automated adaption to the diagram with changes throughout the modeling process.

For diagrams that are covered by the limited toolset, GenMyModel adheres well to the general SD semantics in comparison to other online modelers. While it only offers essential layout support when modeling, it does not impede the user through overengineered automated adjustments.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

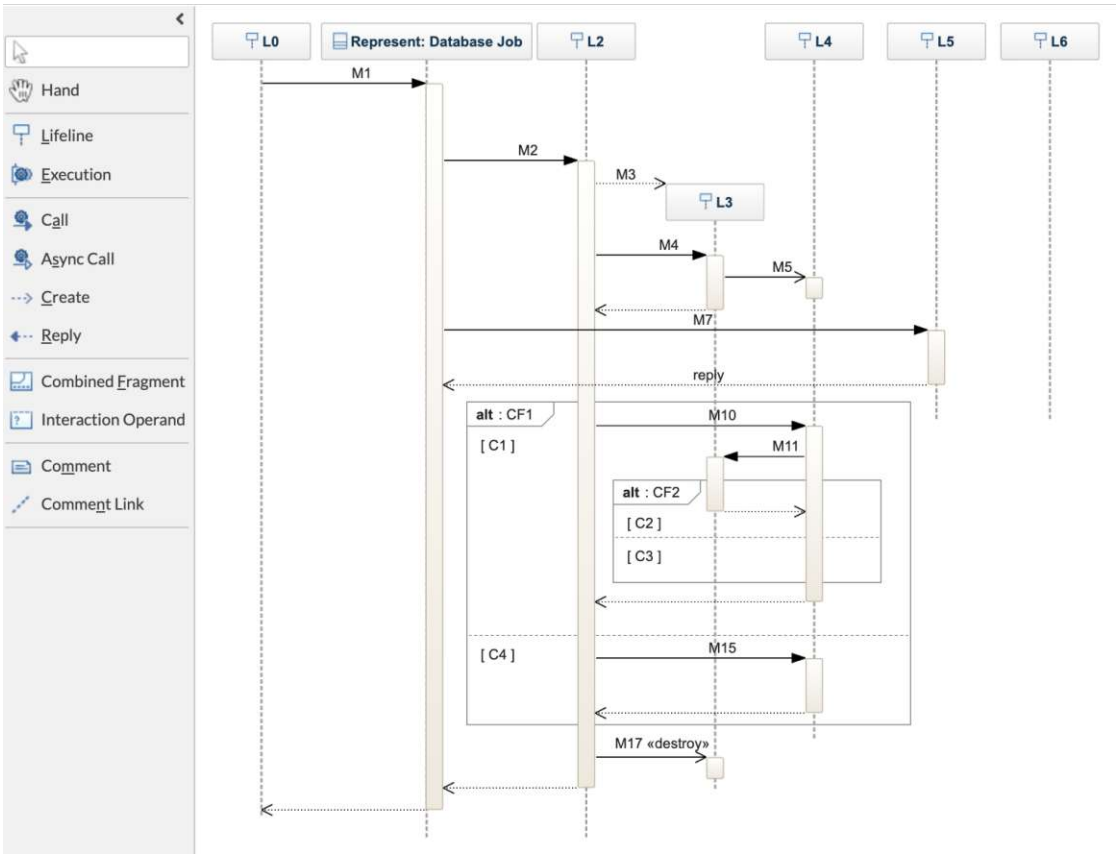


Figure 5.6: GenMyModel

5.3.5 Lucidcharts

Lucidcharts is an unrestricted graphical tool without regard for the UML SD syntax. The editor partially supports the shapes of the UML SD as seen in the Figure 5.7, but none of the syntactic guides or constraints. The shapes can be placed and aligned freely by the user. Lucidchart has a preassembled shape composition for combined fragments but lacks other components, such as a duration constraint. The primary observation on this tool not being designed or adapted for SD is that timelines of lifelines are not enforced to be vertical but consist of routable edges. The user can enforce some limitations by grouping shapes, which the possibility to create custom-assembled shapes and store them as a library.

The modeling experience relies heavily on the user manually creating shapes and layout tasks. There is little support provided by the editor when adjusting the diagram in the form of general shape alignment and snapping. The lack of primary SD shapes creates additional efforts from the user.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

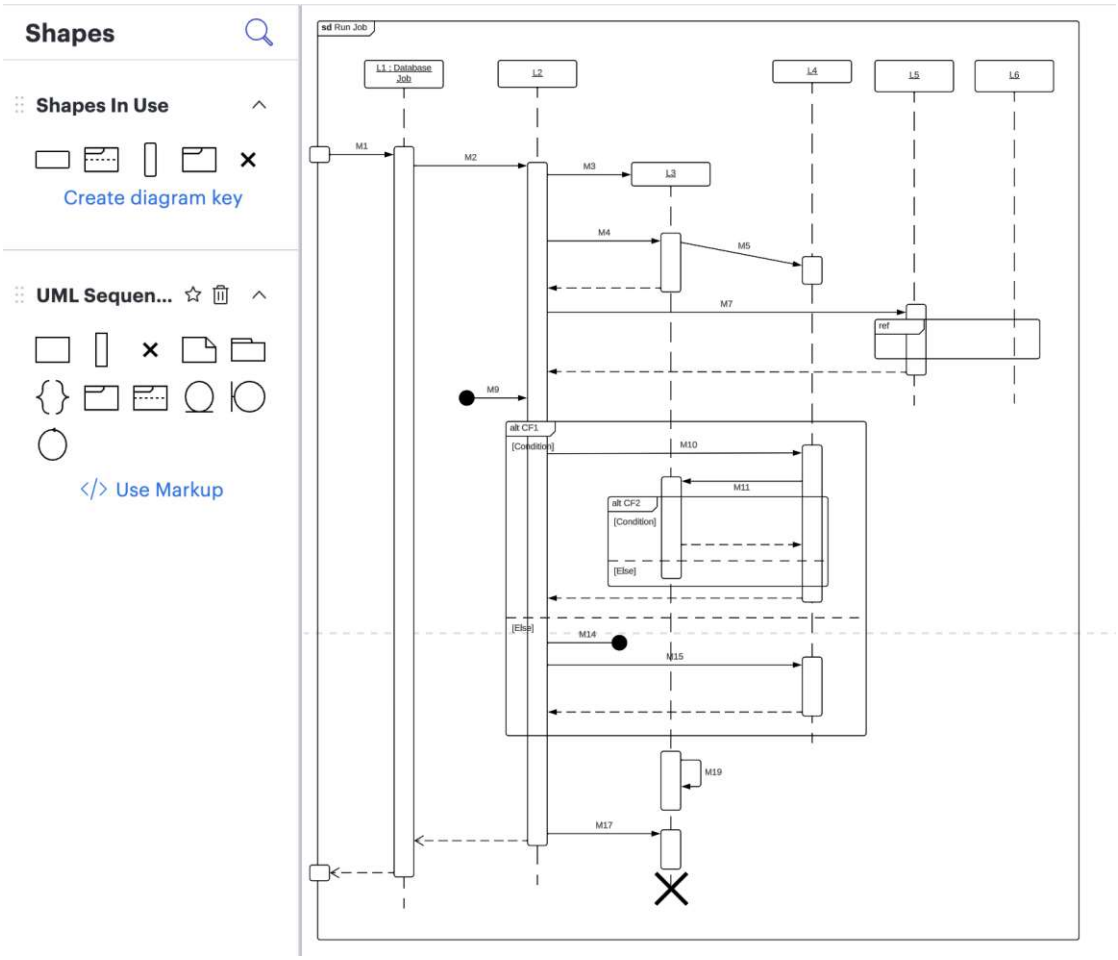


Figure 5.7: Lucidchart

5.3.6 MagicDraw

Lifelines are fixed in height, and their length is automatically adapted. The user can move them horizontally. However, overlaps are automatically resolved by the editor, which enforces a margin between lifelines for a clean layout. The editor avoids overlaps, ensuring a clean layout and distribution of the lifelines, thus positively impacting the modeling experience. The resolution of overlaps is not carried over to the vertical movement of messages and combined fragments, which are freely movable along their lifelines. To support the user, moving a message causes the following and related messages to move along and replicate the movement. This allows us to edit the diagram on the top without breaking the layout further down the timelines.

A unique feature of MagicDraw is the visualization of covered lifelines by combined fragments. The intersection between a combined fragment and the covered lifeline is marked with a dot, as seen in Figure 5.8. The coverage of lifelines is defined on the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

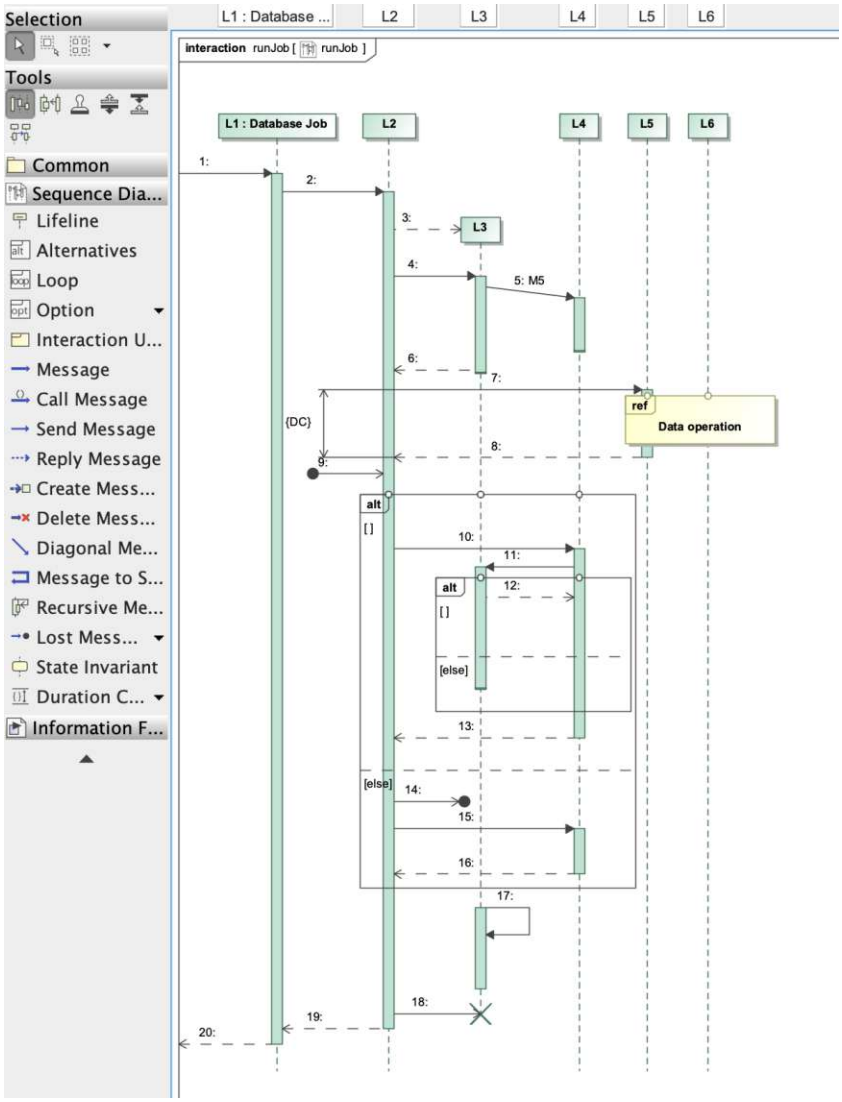


Figure 5.8: MagicDraw

initial drawing of the combined fragment or modified through the combined fragment properties. The combined fragments are limited in horizontal movement, but they can be widened to cover additional lifelines. This is not reflected in the model nor marked with the dot mentioned above.

A combined fragment can be placed into another one by dragging it into it. The content of combined fragments moves with it. Overall, the interactions with combined fragments feel logical and intuitive.

Like Astah, MagicDraw features a tool to reduce and expand gaps.

Overall, MagicDraw provides an intuitive and fluent modeling experience with a well-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

balanced set of features to support modelers. The reactive behavior of the tools does not feel overwhelming and reduces the workload for the user.

5.3.7 Papyrus

Like other editors, its lifelines can only be moved horizontally, with no overlap resolution. Their length is manually changed. The editor generates execution specifications and response messages. The user can move messages or execution specifications by moving either. The movement of lower components can be configured in the settings. The tool features configurable functionality, as shown in Figure 5.9, to react dynamically to moving messages by also moving all the following components. However, the behavior was unreliable during the evaluation. Some messages remained restricted and did not follow the movement, which disorganized the diagram layout and can bring confusion and frustration to the user.

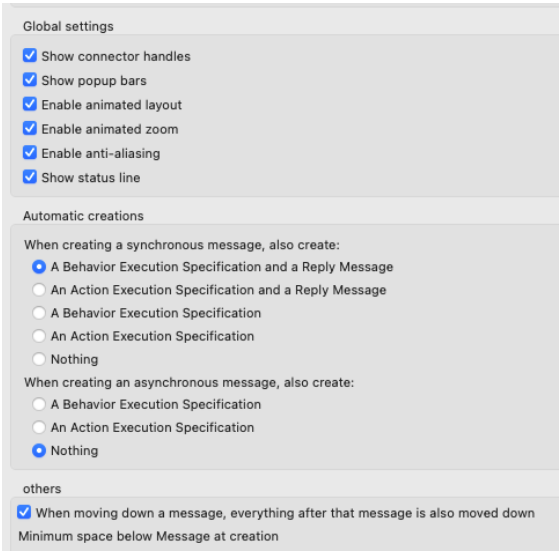


Figure 5.9: Papyrus SD editor settings

In the editor, combined fragments are freely movable horizontally as well as vertically, regardless of the containment of messages, and combined fragments are modeled correctly. One limitation of interacting with combined fragments is that the content within the combined fragments does not relocate in tandem with its container, and adjustments in movement or scaling do not automatically update the coverage of lifelines.

We were only partially able to go through the evaluation process as the program would freeze and stop responding. We reached step 29 of Table 5.1, culminating in the SD shown in Figure 5.10. The missing interactions were evaluated in isolated diagrams of lower complexity.

Papyrus features a wide selection of tools and interactions and is often called the standard in the domain. From testing the tools, it is somewhat frustrating to operate. Additionally,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

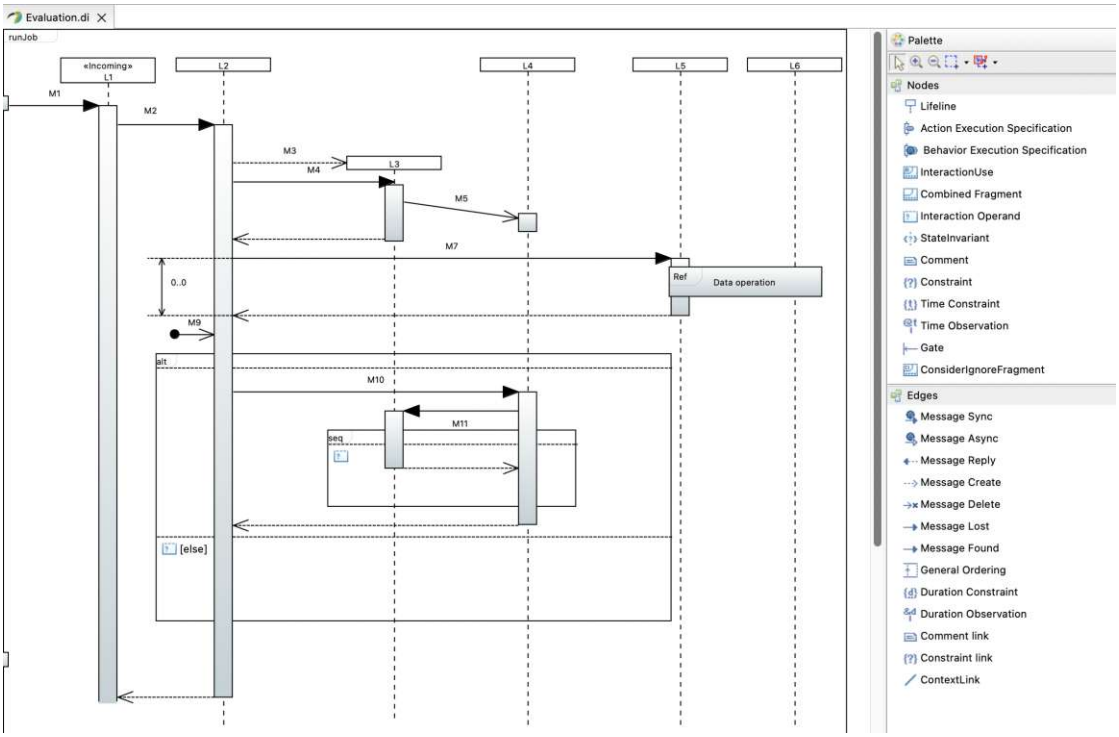


Figure 5.10: Papyrus

there appear to be occasional graphical issues in which the diagram does not entirely refresh from changes and struggles with complex diagrams.

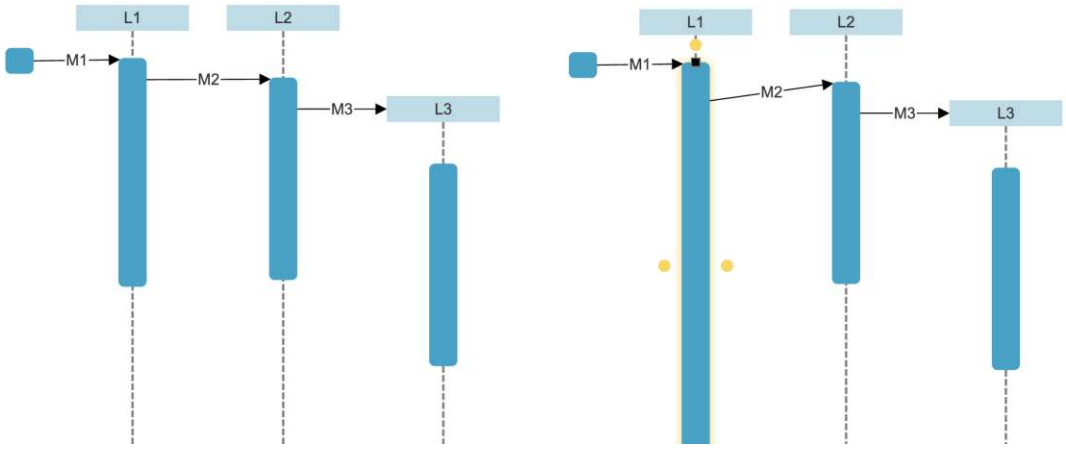
5.3.8 SmartDraw

SmartDraw represents an example of an editor with support for the shapes of the UML SD but none of the syntactic guides or constraints. The editor provides a canvas on which the shapes, such as execution specifications, lifelines, and combined fragments, can be placed and aligned freely by the user.

The evaluation of SmartDraw was terminated preemptively, as the modeling interactions required significant adjustments by the user. The editor does not provide presets for specific message types, such as async and response messages. These have to be manually styled by the user. As another example, the anchors along execution specifications are defined on a position relative to their height. Scaling the execution specification causes all connected messages to shift in position, shown in Figure 5.11. The workaround to this behavior consists of omitting the anchoring of most components, mitigating any relative movement. Overall, the diagram is very sensible to change; minor tweaks can impact many other elements. For instance, vertically moving a message moves its source and target execution specification and their connected messages.

From the partial evaluation, we can deduce that the editor requires significant con-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.



(a) Before scaling (b) After scaling

Figure 5.11: SmartDraw Execution Specification scaling

figuration by the user to achieve a satisfactory result. Diagrams do not respond in a constructive way to user changes, causing additional tweaks. Concluding, the diagram is not a suitable tool for larger SDs.

5.3.9 StarUML

The editor seems lacking in following specifications, aiding the user through constraints and automated movement of related components.

It requires the user to adjust all elements in case of required changes. Combined fragments show no interaction with other components. They can be moved freely and contained messages do not follow. The interaction frame is not integrated into the diagram and can even be removed. The whole tool seems more like an advanced drawing tool, featuring barely any constraints. However, through disciplined modeling, it produces well-readable diagrams, as shown in Figure 5.12 The relationships tab is available but only integrated for the message's source and target relationships. The tool features no covered or containment by CFs.

5.3.10 Trace Modeler

Despite the absence of updates since its v1.6.2 release in 2009, Trace Modeler holds significance for our research due to its unique approach compared to other editors. Its underlying philosophy aligns with the creator's vision of delivering a straightforward and efficient editor to save time. One notable feature is its immediate and automatic layout functionality, which guarantees the creation of structurally sound diagrams while alleviating the user from grappling with shape and layout complications. [Ing17]

The editor Tracer Modeler features a very constricting approach. The components can only be placed and snapped to well-defined locations. As recognizable in Figure 5.13,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

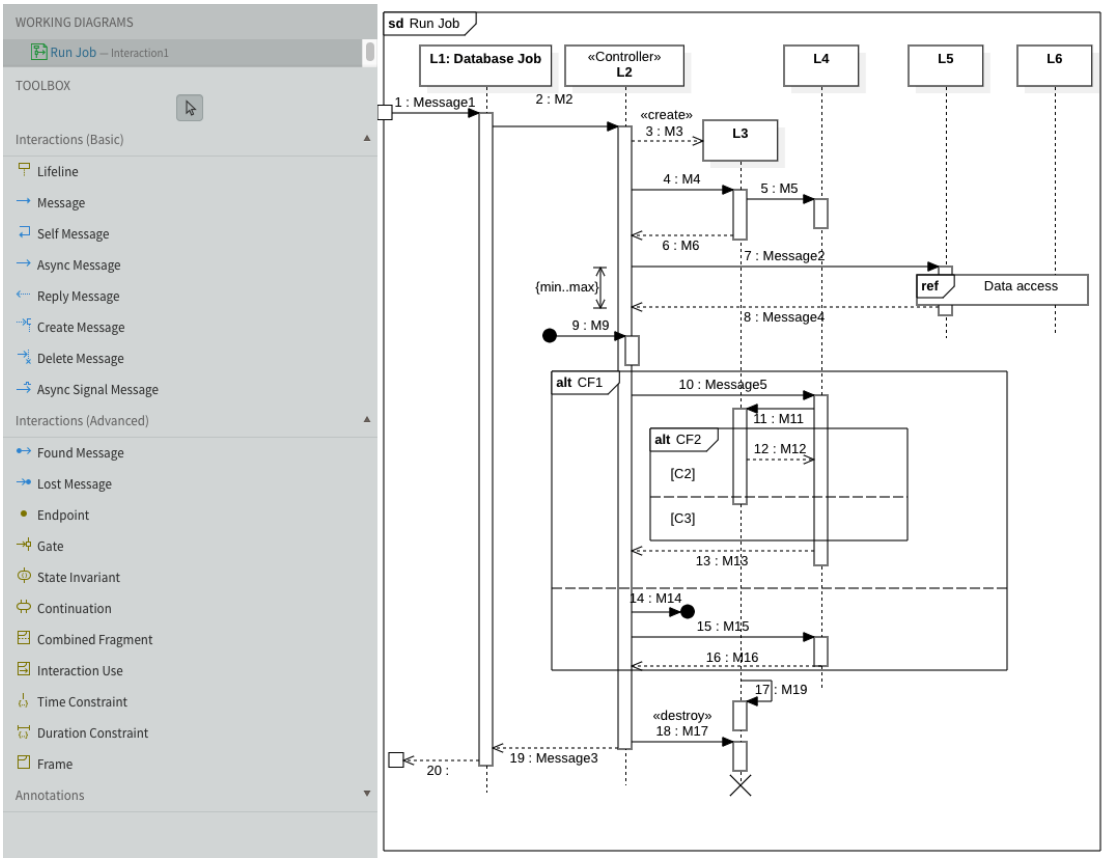


Figure 5.12: StarUML

lifelines have fixed spacing to each other. They can be reordered through horizontal movement, causing the remaining lifelines to move accordingly. The same is the case for messages in the vertical orientation. Messages stack vertically from top to bottom with the same spacing. From the editor's design, having multiple messages at the same height is impossible.

Through the rigorous layout and modeling constraints, the Trace Modeler editor can compute the control flow of the SD. The resulting interesting feature is the control flow as seen in Figure 5.13.

5.3.11 UModel

The editor provides an efficient modeling experience in some regards. Surprisingly, the lifelines can be moved vertically and aren't locked. However, it resolves overlays in lifelines. The editor generates execution specifications with new messages and optionally also reply messages. The layout of messages is primarily up to the user, with no resolution of overlaps or enforcement of margins between messages. The organization of messages is supported through toggleable moving behavior through which all following dependent

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

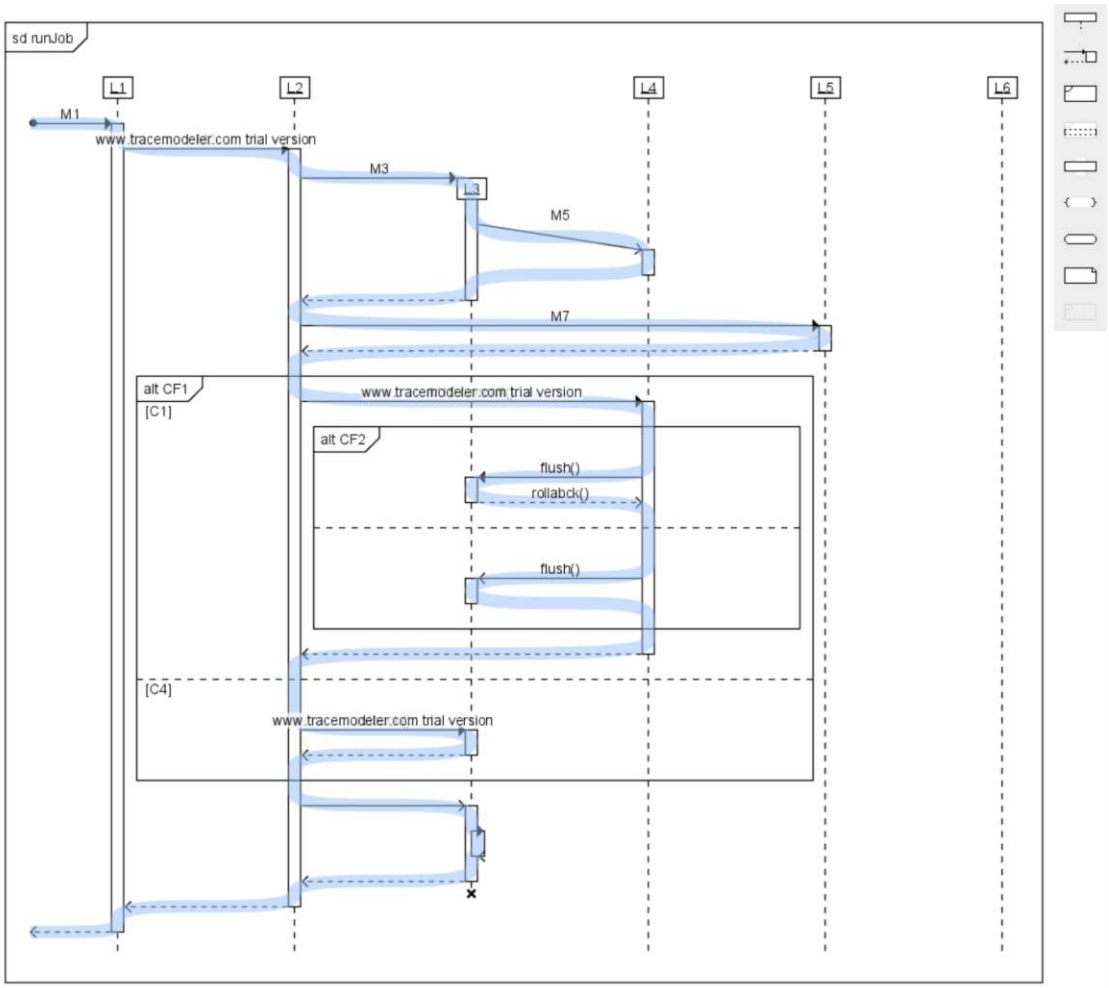


Figure 5.13: Trace Modeler Sequence Diagram evaluation

messages can be moved concurrently. Execution specifications are automatically scaled following their respective messages as shown in Figure 5.14. UModel handles combined fragments as overlays over the diagram; they move independently from messages and lifelines but can be contained in combined fragments. The coverage relationships are generated in the exported XMI model.

5.3.12 Visio

Visio can be classified as an unrestricted graphical tool; it incorporates the graphical components of the SD, such as lifelines and combined fragments, but not the component relationships and resulting behaviors of SD modeling. Moving components is not restricted, nor are components moved in dependency on each other due to relation, besides containment relations of execution specifications in combined fragments. The tool allows

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

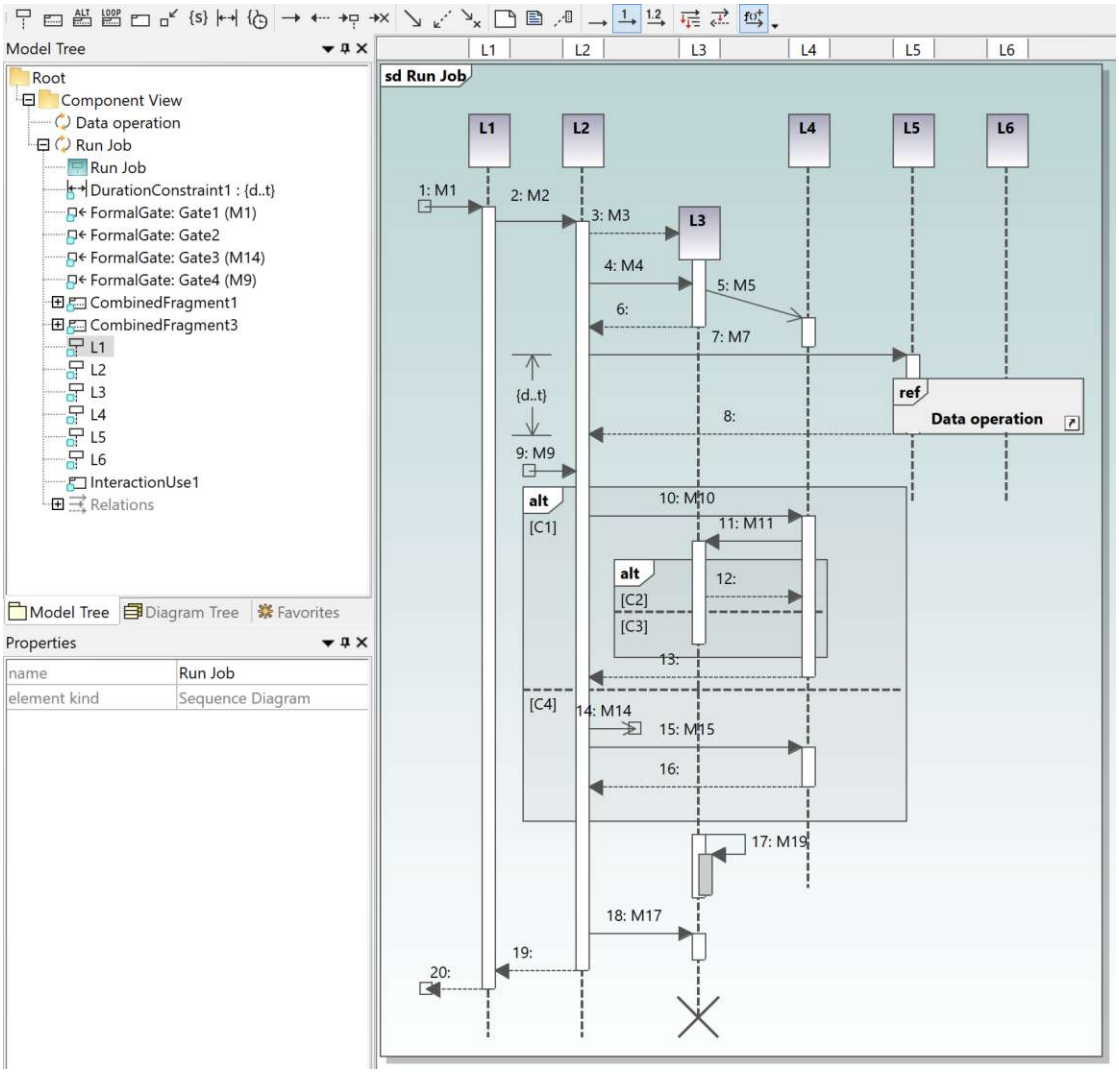


Figure 5.14: UModel

users to place components such as lifelines and execution specifications and connect them via messages as connecting edges. The local version of Visio differentiates itself by allowing more SD-specific components to be imported through stencils, which include preassembled graphical components such as the Duration Constraint. The supportive modeling behaviors by the tool consist of messages anchoring to lifelines or execution specifications. Execution specifications do not reliably anchor to lifelines, which proved inconvenient when moving lifelines.

Overall, the Visio editor provides a reliable and fast solution for sketching and visualizing simple diagrams. More complex diagrams such as ours require disciplined modeling to achieve the desired results as shown in Figure 5.15. The user can benefit from the tool's

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

seamless integration with other Microsoft 365 products. The unrestricted editing gives freedom to the user but no syntactic support. The tool does provide layout support through the general alignment of diagram objects, but working with the tools still requires significant adjustments of the placement of the shapes by the user to achieve the final result.

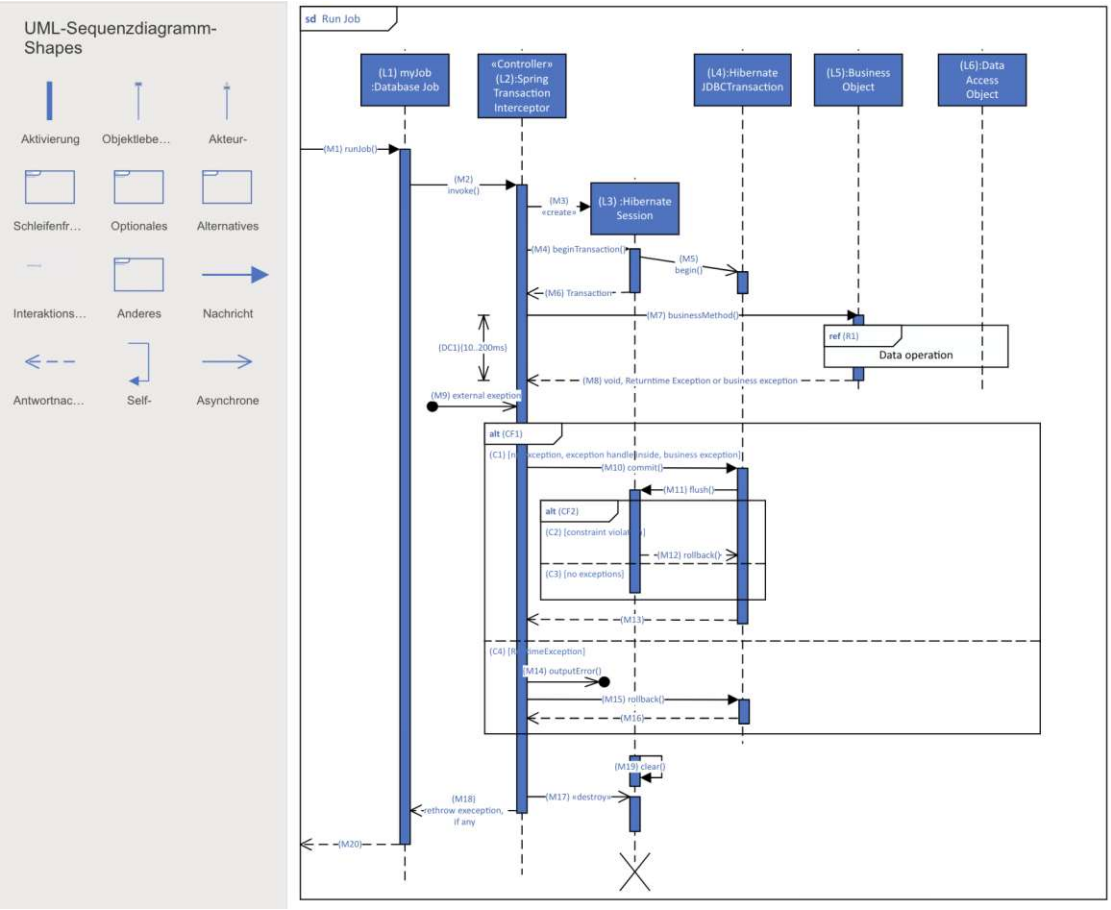


Figure 5.15: Visio

5.3.13 Visual Paradigm

Lifelines are locked at a fixed height. Created lifelines match the height of their creation message, and the lifeline length is adapted to the content.

The editor generates execution specifications automatically, and their visibility can be turned off. Response messages are created manually. Messages can be moved freely along their lifeline without affecting other components besides the scaling of their source execution specification. Therefore, overlaps of messages are also not resolved by the editor; one exception is if messages are copied/pasted, which causes the editor to insert

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

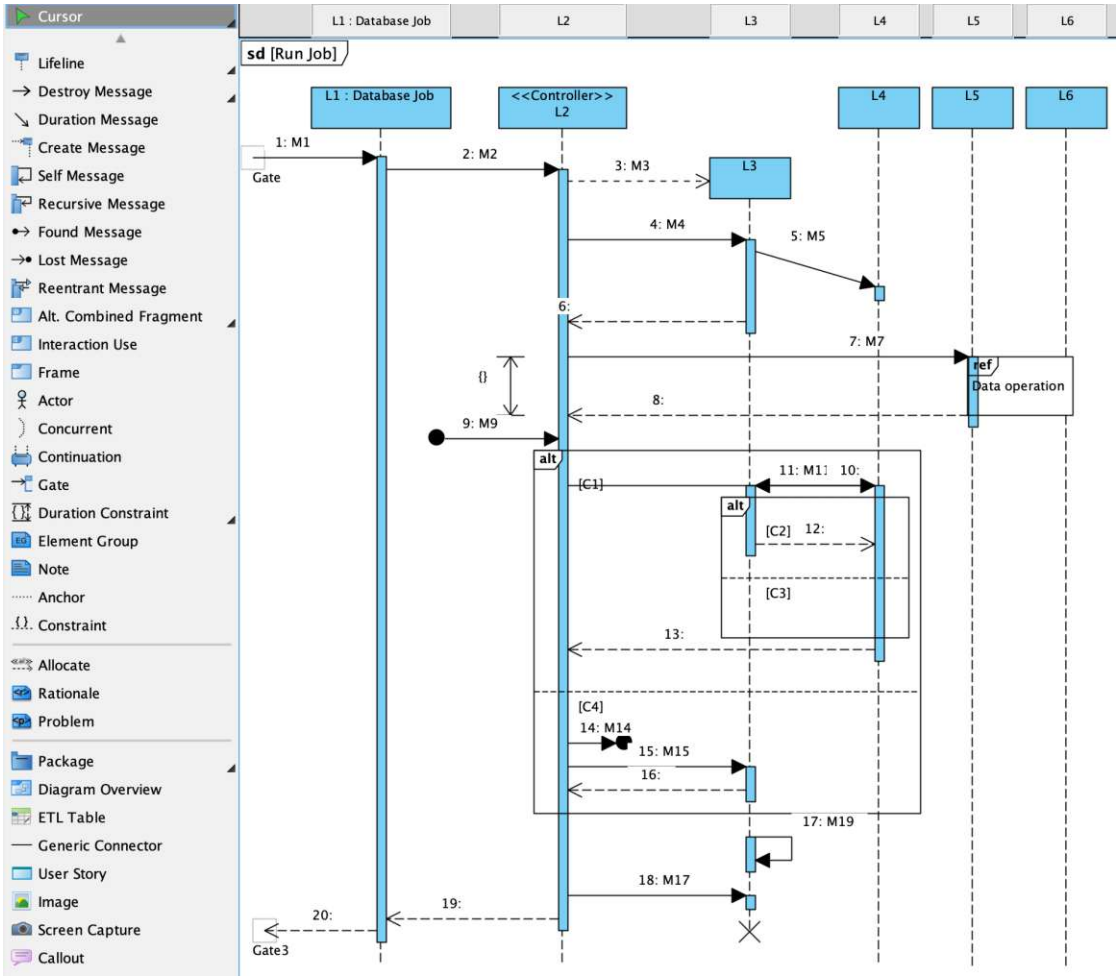


Figure 5.16: Visual Paradigm

them into the diagram and move downward all the following components.

The editor makes a fair impression as can be recognized from the resulting diagram in Figure 5.16. It shows some semantic attention, and handling combined fragments is reasonably well implemented as the containment of elements is handled.

5.3.14 Visual Paradigm Online

Visual Paradigm Online brings the desktop modeling experience of Visual Paradigm to the browser but lacks many of the modeling supporting features. The online version is not a syntax-aware editor but a drawing tool with few constraints. It has similarities to draw.io's functionality and interface but with added specificity to UML modeling and modeling support. In comparison to other available online editors, it incorporates SD-specific modeling, such as the automatic creation of execution specifications and

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

components having fixed features and behavior according to the specifications. For example, lifelines are vertically locked, and their length is manually set. Like the desktop version of Visual Paradigm, the messages can be moved freely, and occurring overlaps are not resolved.

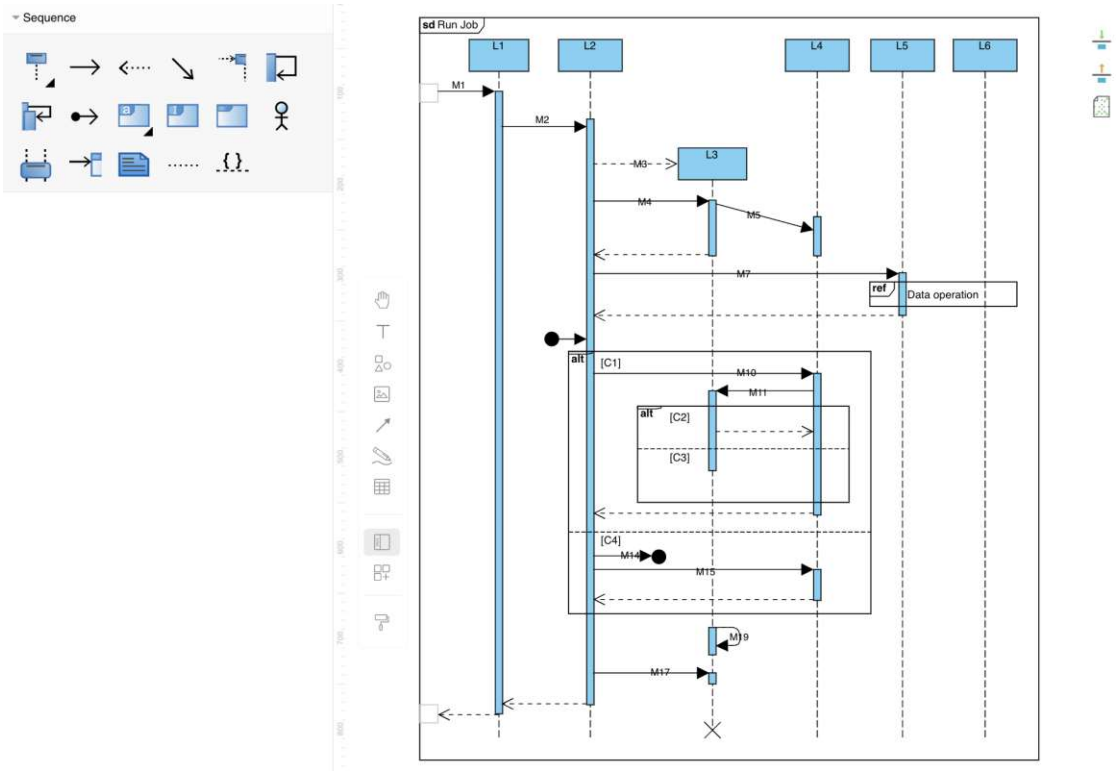


Figure 5.17: Visual Paradigm Online

Overall, the tool provides more modeling support and adherence to the SD specifications than other online editors. However, it falls short in comparison to more elaborated desktop solutions. The tool necessitates user adjustments, like aligning reply messages and scaling execution specifications, to generate a neat and organized diagram. While suitable for creating quick and simple diagrams, for more intricate ones like ours 5.17, the modeling experience can become somewhat laborious.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

5.4 Evaluation Summary

This subsection outlines the most critical aspects and differences across the evaluated tools. The following chapter elaborates on these aspects while conceptualizing an ideal SD modeling experience.

The evaluation showcased the variation throughout the different tools. The tools offer different degrees of modeling support and constraints, going from free shape positioning editors to very restrictive and structurally fixed editors. From the tool evaluation, we compiled the modeling constraints and interaction behaviors commonly employed by the tools, as summarized in Table 5.5.

Table 5.5: Common Modeling Constraints & Interaction Behaviors

#	**Behaviors**
1	*Lifelines* are locked at a fixed height unless they are created
2	The vertical movement of messages causes lower, dependent *messages* to move along
3	*Reply messages* are automatically created for synchronous messages
4	The *interaction frame* is fixed in place and scales with its content
5	*Execution specifications* are automatically created and bound to their respective messages
6	*Duration constraints* move and scale with their associated messages
7	Moving a *combined fragment* also moves the containing messages, execution specifications, and combined fragments
8	The overlap of *lifelines* is resolved by enforcing a horizontal margin between them
9	The overlap of *messages* is resolved by enforcing a vertical margin between them
10	The visualization of *execution specifications* can be turned off
11	The label of cut-off *lifelines* is shown in a toolbar over the modeling view

The implementation of the listed modeling constraints and interaction behaviors differs from one tool to another. We assessed the support for each of these aspects across the modeling tools. In Table 5.7, the modeling tools are presented in ascending order based on their support for modeling constraints and interaction behaviors. The results illustrate variations among the available modeling tools. These findings serve as the basis for our categorization of the tools into five general categories, as outlined in Table 5.8.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Table 5.7: Modeling Constraints & Interaction Behaviors Support

Behaviors	Draw.io	Lucidcharts	smartdraw	Visio	Visual Paradigm Online	GenMymodel	StarUML	UModel	Visual Paradigm	Papyrus	Astah	Enterprise Architect	MagicDraw	Tracemodeler
	Modeling Tools													
1	○	○	○	○	●	●	●	○	●	●	●	●	●	●
2	○	○	○	○	○	◐	○	●	○	◐	◐	●	●	●
3	○	○	○	○	○	●	○	●	○	●	●	○	○	●
4	○	○	○	○	●	◐	●	●	●	●	●	●	●	●
5	○	○	○	○	●	●	●	●	●	●	●	●	●	●
6	○	○	○	○	○	○	○	○	●	●	●	●	●	○
7	○	○	○	○	○	○	○	○	○	○	◐	●	●	●
8	○	○	○	○	○	○	○	●	○	○	○	○	●	●
9	○	○	○	○	○	○	○	○	○	○	○	◐	○	●
10	○	○	○	○	○	○	●	○	●	●	◐	●	●	●
11	○	○	○	○	○	○	○	●	●	○	●	●	●	●

●= *supported*, ◐= *partially supported*, ○= *not supported*

From evaluating the tools, we can observe a relation between the deployment of tools and their adherence to specifications and layout support. In our categorization in Table 5.8 we find web-based tools only in categories one and two, while categories three to five are populated with desktop applications. This clearly shows a lack of web-based modeling tools specialized for modeling.

Table 5.8: Modeling Tools Classification

1	2	3	4	5
Freely placeable collection of shapes of the SD	Anchoring of lifelines, alignment of execution specifications to lifelines, and anchoring of messages. No resolution of overlaps	Optional automatic generation of execution specifications and response messages. Some support on message layout.	The tool resolves overlap on the lifelines and supports the user in the message layout. Combined fragments interact with other components and group their content.	The layout is fully managed by the tool; no overlap is possible.
Draw.io Lucidcharts SmartDraw Visio	Visual Paradigm - Online GenMyModel	StarUML UModel Visual Paradigm Papyrus Astah	Enterprise Architect MagicDraw	TraceModeler

Throughout the evaluation, we encountered behaviors that negatively impacted the modeling process, resulting in frustrating or repetitive modeling practices.

One major contributor to the work effort was working with the more generalistic drawing tools. These tools provide no constraints to how the SD components are placed, meaning

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

even execution specifications would not be bound to lifelines, requiring the user to do this manually multiple times. This is why evaluations of such tools were not carried out to their full extent and were concluded when excessive repetition occurred without the emergence of new behaviors. The tools are viable for contained, straightforward diagrams due to their instant setup but fall behind in modeling time efficiency and comfort with more complex diagrams.

A factor contributing to frustration was inconsistent component snapping and alignment. Some tools try to adapt the diagram dynamically to ease the modeling process but fail to do so. Throughout the modeling process, the user should remain in control of the diagram, understanding how the tool behaves and modeling accordingly. Having to fix unwanted dynamic snapping manually or scaling behaviors of the tool brings significant frustration to the user if required repeatedly.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

CHAPTER 6

Conceptualization

This chapter discusses the modeling interactions and behaviors an ideal SD modeling tool should provide. It is ideal in the sense of minimizing manual interactions by the user that encumber the SD modeling process. The concept for tool behavior should reduce the necessary modeling user interactions and, in that respect, improve the efficiency of the SD modeling tool. The users should be able to focus on the diagram's content rather than on the orderly layout of lifelines, messages, etc.

Finding the right balance between user freedom and guidance is a challenge that can significantly impact the modeler experience, and it defines the user base and use case of the modeling tool [Fow04]. This results in a tradeoff between user freedom and modeling guidance. On the one side, graphical tools with little regard for SD syntax rules give the user complete freedom in creating the diagram and allow a creative, unconstrained thinking process. On the opposing side, tools such as TraceModeler 5.3.10 give strict guidance on the diagram syntax, lifting the modeler from that responsibility.

From a user perspective, general graphical tools imply additional work to ensure the diagram's correctness and the layout micromanagement, as the model doesn't dynamically react and adapt to changes based on the component's relations. The opposite extreme, a restrictive and reactive tool, can cause frustration to the user if automated movement and snapping behaviors become overwhelming to the point in which the modeler doesn't feel in charge anymore and is hindered from realizing his diagram.

With this concept, we strive to provide a modeling experience that builds on constraints and supporting features to guide a syntactically correct modeling process and support a well-structured layout while giving the user modeling choices and freedom.

In our concept, we differentiate between general tool requirements, which describe how and in which environment the tool should be usable. General modeling requirements find application to most diagrams, and SD-specific component modeling requirements focus on the required SD components and their interactions.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

6.1 General Tool Requirements

It's essential to recognize that the modeling experience and user interaction represent only a portion of the overall user experience of a modeling tool. Additional supporting factors play a pivotal role in shaping the user's overall experience with a modeling tool, which can significantly influence the appropriate tool selection and must be considered. How frequently a modeling tool is utilized is strongly dependent on how well it can be integrated into the user's overall workflow and working environment.

6.1.1 Deployment

The ease of deployment of a given software contributes to the software's adoption. By today's standards, users expect a seamless use of the software through direct access over the browser with no installation, integration in their existing IDEs, or a seamless installation on different workstations and operating systems.

Many of the tools mentioned above necessitate installation on a user's machine, introducing additional barriers to entry during the initial setup phase. The effort required for tool installation may not be justified depending on how frequently a user engages in modeling UML diagrams. Furthermore, the installation process imposes specific operating system requirements, which might not be readily accessible or compatible with the user's system.

The modeling tool should be accessible from various systems, such as mobile devices and traditional workstations, and make little restrictions on the supported operation systems. This has been successfully achieved by integrating the editor into a browser-based editor.

6.1.2 Collaboration

Given the growing emphasis on collaboration in contemporary work practices, the modeling tool must allow easy accessibility to multiple users [DI16]. These users should have simultaneous access to a shared diagram, enabling seamless real-time collaboration. These functionalities are rarely supported by the more traditional locally running modeling tools and instead actively developed in web-based editors, which partly contributes to the trend toward browser-based editors. Therefore, the conceptualized modeling editor should support the real-time collaboration of multiple users on a single diagram in a remote setting.

6.2 General Modeling Requirements

These general modeling requirements are not SD-specific and can be applied to any diagram. They are intended to support modelers in their practices by mainly supporting an organized diagram layout through editing tools. We collected a selection of generalizable interaction behaviors by evaluating the existing modeling tools chapter 5.

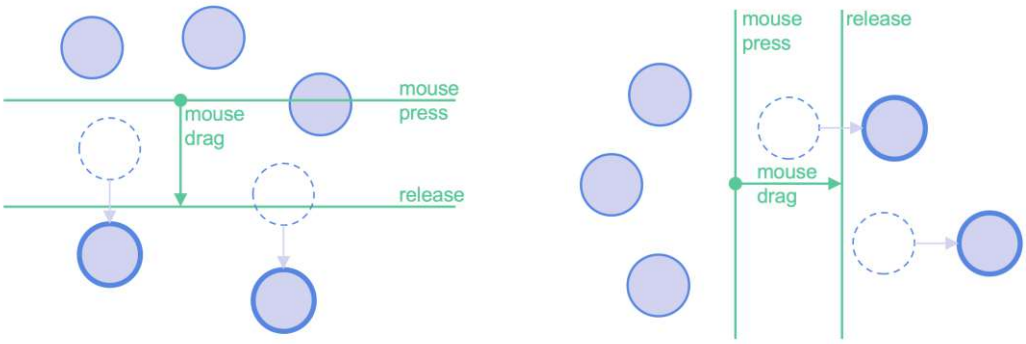
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

6.2.1 Positional Selection and Movement

When working on extensive diagrams with many components, the available diagram space must be managed sensibly to ensure an organized and consistent distribution of components across the page. A well-readable diagram avoids clusters and distributes information evenly across the page, giving consistent margins between elements. It is challenging to maintain an organized diagram when modeling, as gaps form when removing components, and clusters of components grow through later additions of components.

The editors Visual Paradigm Online, Visual Paradigm, MagicDraw, and Astah provide features to manage the positioning of multiple components to create or reduce space. These features allow the movement of all components on the vertical or horizontal axis, depending on their position, facilitating the effortless creation or reduction of space between components.

This editing functionality proves especially useful when components need to be added in the middle of the diagram and require components below it to be moved down. The opposite use case would be removing a component in the center, which leaves a gap in the diagram; closing this gap requires moving all the components below back up. Compared to manually selecting and moving all the components, one key benefit of this feature is the movement's limitation to only the vertical or horizontal orientation.



(a) Vertical (b) Horizontal

Figure 6.1: Shifting Feature Concept

The current implementation of this functionality relies on two features: one for pushing and another for pulling. The key distinction lies in the selection of elements. The pushing feature selects and moves the elements situated in the direction of the drag, while the pull feature selects and moves the elements situated in the opposite direction of the drag. From our testing, we determined that when dealing with diagrams, only the relative position of components is significant. This implies that, for example, when additional horizontal space is required, it makes no difference whether the elements to the right or left are moved away. This leads to our conclusion that we can consolidate it into a single feature instead of two. For this feature, it is sufficient to move the right or lower

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

components to create or reduce space; the components above and to the left do not need to be selected or moved.

This feature, as visualized in Figure 6.1, allows the user to click and hold anywhere on the canvas. From that position, the feature recognizes the orientation by moving the mouse horizontally or vertically while holding it. In the case of a vertical selection, all objects below are selected. In the case of a horizontal selection, all components to its right are selected. As part of this feature for vertical selection, the last interaction is to move the mouse up or down, resulting in the selected components moving in that respective direction by the distance moved from the initial cursor.

6.2.2 Automated Layout

An alternative approach to ensuring a well-structured and organized layout of the final diagram is the implementation of automatic layout functionality, meaning the user can freely model the diagram without concern for the positioning and scaling of components and relies on the editor to adapt the diagram automatically. Such behavior and implementations have been evaluated in [Hoo13, SHvH18].

6.2.3 Shortcuts

Shortcuts or hotkeys activate a specific feature by pressing a defined key or combination of keys. This can speed up the modeling as the user can instantly access features, such as creating a new component, without activating the feature through a palette. This reduces the amount of clicks and mouse movement [EDDT11]. Shortcuts should be available for the most common actions to support the user. For the SD, these would be the creation and movement of messages.

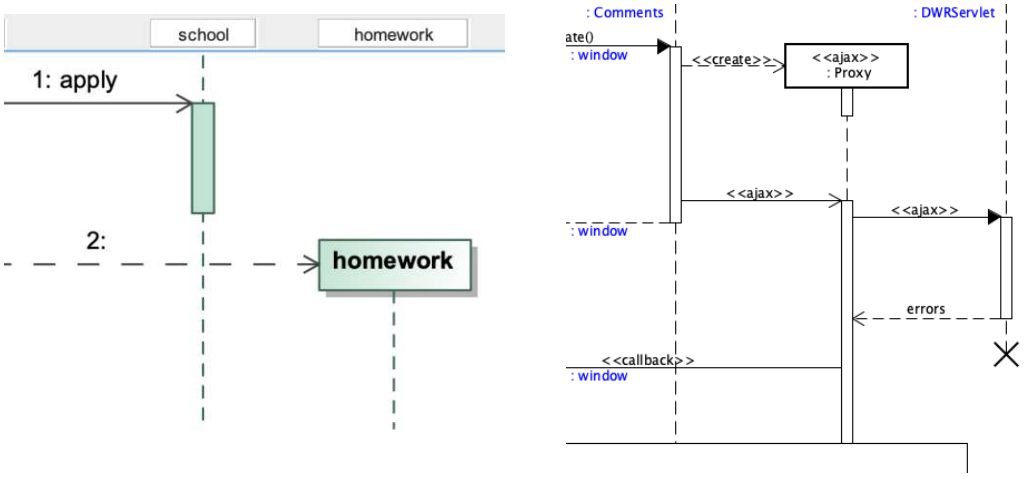
6.2.4 Semantic Selection

Various selection features are available in modeling software, encompassing individual component selection, multiple component selection (typically achieved by holding down the shift key), and the area selection feature. However, users might find additional selection features based on the model's semantics beneficial. Under semantic selection, we define a selection feature in which, upon selecting an individual component, the semantically dependent elements are automatically added to the selection. The dependencies of objects are defined according to the diagram type.

In the context of SD, the sequencing of components is a crucial aspect of the diagram's structure. For users, identifying all components sequentially dependent on specific elements could prove advantageous, especially when temporary repositioning becomes necessary. Notably, both MagicDraw and Umodel offer a toggleable functionality that allows dependent messages to move along with the selected message.

It's essential to clarify that this functionality should not be equated with moving all messages below a particular time point. While these components occur chronologically after

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



(a) MagicDraw (b) Astah

Figure 6.2: Visualization of off-screen lifelines

that point, they are not necessarily interdependent. Implementing such a feature involves a recursive evaluation identifying and selecting the recursively dependent components.

6.2.5 Off-screen elements

Another feature to consider in the SD editor concept is handling off-screen elements. Visualizing off-screen elements can benefit the navigation of larger models and positively impact the user experience of the editor [DCLB22].

As noted in the evaluation, this feature is supported by only a few SD modeling editors. In the case of SD, the most critical information for the user of off-screen elements is identifying the partially seen lifelines and the source or target of partially seen messages. An exemplary visualization of such an implementation can be experienced in MagicDraw, EnterpriseArchitect, Visual Paradigm, Tracemodeler, Umodel, and Astah. Most of these editors always visualize the label of each lifeline in an additional information box on top of the canvas as shown in Figure 6.2a. Astah provides the information on off-screen lifelines and messages in blue writing as shown in Figure 6.2b.

6.3 SD Components Requirements

In this section, we look at interactions specific to the SD and its components. From the evaluation in chapter 5, we collected the common recurring behaviors across the SD modeling editors in Table 5.5. These form the foundation for the concept and are further expanded and specified to each SD component throughout this conceptualization phase.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

6.3.1 Interaction

The interaction, also referred to as the interaction frame, encloses the SD diagram. Some modeling tools entirely forfeit its visualization, some make its visibility togglable, while some model it as an object part of the diagram. SDs are modeled as individual diagrams within each file, as SD depicts an interaction. Therefore, we can support the user by setting up each new diagram with an empty interaction, which is not deletable and does not allow additional interactions. Furthermore, following the SD syntax, all components should be contained within the interaction. The size of the interaction should dynamically adapt to its content.

The defined requirements are collected and summarised in Table 6.1.

Table 6.1: Interaction Requirements

#	Requirement
1	The SD contains a single interaction
2	Interaction scales automatically to include all SD components
3	Empty interaction is created with a new model

6.3.2 Lifelines

Lifelines depict interaction partners and build up the foundation of SD. Messages, execution specifications, and combined fragments are anchored along them. Therefore, we clearly understand how they should behave based on the UML specifications [Gro17] and the general use of lifelines.

Lifelines consist of a rectangular header containing the role name and class of the lifeline and the timeline, drawn as a vertical dashed line. The head can also contain the specification for the lifeline's stereotype, enclosed in double-angle quotation marks. The typical stereotypes actor, boundary, control, and entity have their respective icons replacing the default rectangle.

The positioning of lifelines is well-defined in SD. Lifelines are anchored at a fixed height within the interaction frame and can only be moved horizontally. Lifelines created within the interaction are placed at their respective initiation position aligned with their creation message. The length of a lifeline should scale to contain all of the occurrence specifications, combined fragments, and execution specifications along it. Destructed lifelines have to terminate at the location of their destruction message.

The order of lifelines can also be deducted from general modeling approaches. The SD is read from left to right and from top to bottom. The lifeline with the first message should be on the very left, and the following lifelines should be ordered as they participate in the interaction.

To ensure an organized layout of the diagram, lifelines should not be able to overlap, and some spacing between lifelines should be ensured for ease of readability. The user

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

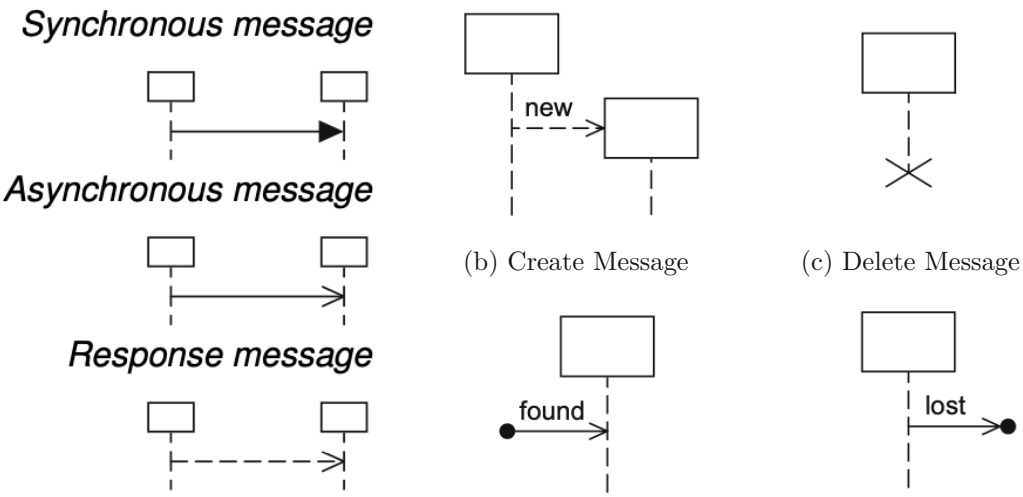
can be supported with the diagram layout by the editor functionality to evenly space out lifelines across the page, voiding the need to do it manually. Table 6.2 collects and summarises the defined requirements.

Table 6.2: Lifeline Requirements

#	**Requirement**
1	Lifelines are locked at a fixed height unless they are created
2	The length of a lifeline should scale based on its content
3	Destructed lifelines have to terminate at the location of their destruction message
4	The overlap of lifelines is resolved by enforcing a horizontal margin between them
5	Lifelines should be horizontally ordered according to their moment of interaction
6	The label of cut-off lifelines is shown in a toolbar over the modeling view

6.3.3 Messages

Messages are a core component of SD. They visualize the communication between lifelines and are depicted as directional edges.



(a) Message Kinds (b) Create Message (c) Delete Message (d) Found Message (e) Lost Message

Figure 6.3: Message Sorts and Kinds [SSHK15]

In SDs, messages vary in kind and sort, which has implications for their syntax and notation, shown in Figure 6.3. To accurately model SDs, the editor needs to support all their variations. In message kind, we differentiate *found messages*, *lost messages*, and *complete messages*. The message sorts consist of *synchronized call*, *asynchronised call*, *asynchronised signal*, *create message*, *delete message*, and *reply message*. Generally, messages within an SD are instantaneous and horizontal. The SD specification [Gro17] defines messages with a duration to be depicted as lines going downwards diagonally.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

As Messages make up most of the components of an SD, maintaining an organized layout of messages impacts the overall layout diagram and its readability. A set of criteria specific to messages can be established for a well-organized SD. Throughout the diagram, the crossing of messages should be minimized, messages should be displayed in the order they are sent, and messages should be situated horizontally (except explicitly drawn with duration) [NPA16].

The established requirements are collected and summarised in Table 6.3.

Table 6.3: Message Requirements

#	Requirement
1	By default, messages should be created as horizontal lines with no duration
2	The user should be able to reorder messages
3	Moving a message should maintain its being horizontal
4	Support for messages with duration
5	Messages should have vertical margins between them to ensure good readability
6	When adding a message, the components below should move down to provide space and sufficient margin between them
7	Message overlap should be handled by the editor
8	Message crossing should be minimized

6.3.4 Execution Specification

One specific aspect in which the implementation varied across modeling tools is if and how execution specifications are handled. Following the UML 2.5 specification, execution specifications indicate the execution of a behavior or an action with a start and finish event. It is visualized as a rectangle over the lifeline, with its height indicating the behavior duration Figure 6.4. Overlapping execution specifications are visualized as overlapping rectangles, shown in Figure 6.4.

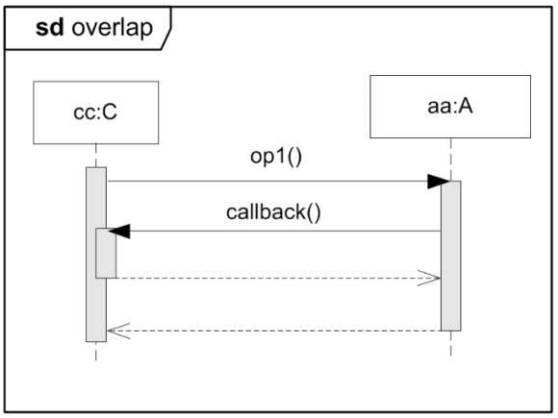


Figure 6.4: Notation of Execution Specifications [Gro17]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

The modeling of execution specifications appears to be optional based on the modeler's preference and model use case. This is also reflected in the modeling tools, as their support for execution specifications varies.

Some tools generate them automatically and require them, not allowing them to be removed without removing the messages. Another implementation allows the user to trigger a diagram-wide visualization of execution specifications. In StarUML and MagicDraw, this boolean property is called "show Activations", while in Astah, it is "Execution Specification Visibility". Others, eg. Papyrus, allow interaction with them, allowing them directly to create and remove them freely.

Ideally, the decision on how to handle execution specifications should be configurable by the user. The automated generation of execution specifications is, in most cases, a useful functionality in combination with the option to disable them altogether. However, the user should have the possibility to influence individual execution specifications as well as be able to create isolated ones with no adjacent messages.

The defined requirements are collected and summarised in Table 6.4.

Table 6.4: Execution Requirements

#	**Requirement**
1	Execution Specifications are automatically created
2	Execution Specifications adapt their length to their respective message's position
3	Execution Specification stack when overlapping as in Figure 6.4
4	Execution Specifications can be hidden

6.3.5 Combined Fragments

The Combined Fragment features wildly diverging interaction behaviors across the evaluated editor. Some editors handle combined fragments as visual overlays without interacting with messages or other components. Other tools integrate them in the diagram by allowing the containment relation to messages and other combined fragments. This makes this component a particular discussion element and motivates closer inspection and reflection.

When modeling SD, combined fragments and their containing components can be viewed as a grouping of components and should move as such. Moving a combined fragment would also require moving all its contained components. Combined fragments and their container messages and components should be seen as a building block of an SD, and it makes sense to interact with them as such.

Depending on the user's interaction and modeling practices, it might be necessary to adapt the combined fragment without affecting its content. EnterpriseArchitect's multiple movement modes for combined fragments are an example of a novice implementation. This allows the user to move the combined fragment either freely or following the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

movement constraints of its content. This concept of toggleable mode, as implemented in EnterpriseArchitect, is unique to the combined fragment, making it not intuitive for the user to learn and use.

Building on the concept of multiple editing modalities, this could be expanded to be a global toggle through a dedicated button, key, or both. This would also make it applicable to move messages and other components. Therefore, it becomes a more integrated and recurring functionality of the modeling process, representing a more evident editing behavior for the user to use.

Combined fragments have to be aligned and scaled horizontally and vertically. Horizontally to cover the required lifelines and vertically to contain the messages and combined fragments.

In a modeling process, the user should be able to either draw a combined fragment and fill it with messages or be able to draw a combined fragment over existing messages. If the user adds messages above a combined fragment and its content, the combined fragment should move with the remaining components. If the combined fragment remains fixed, the user must manually adapt all combined fragments and align them over contained components.

We summarize the discussed aspects into a set of requirements in Table 6.1.

Table 6.5: Combined Fragments Requirements

#	**Requirement**
1	Components can be created within combined fragments
2	Moving a combined fragment vertically also moves its content
3	Moving a combined fragment can alternatively be moved freely without their content
4	Combined fragments scale automatically to include their content
5	Combined fragments scale with the horizontal movement of covered lifelines
6	Combined fragments can be drawn over existing components and will contain them
7	Messages cannot cross the combined fragments frame

6.3.6 Interaction Use

Interaction Uses are references to other interactions. They allow the simplification of large and complex diagrams by splitting them up. The Interaction Use component is depicted as a node like the interaction frame, containing the name of the referenced interaction at its center as shown in Figure 6.5.

Interaction Uses can be placed along lifelines or over Execution Specifications and can cover multiple lifelines. The placement of Interaction Uses is similarly implemented as the Combined Fragments. Some editors handle the components as a freely movable overlay, while other tools implement them anchored to the covering lifelines and interacting with

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

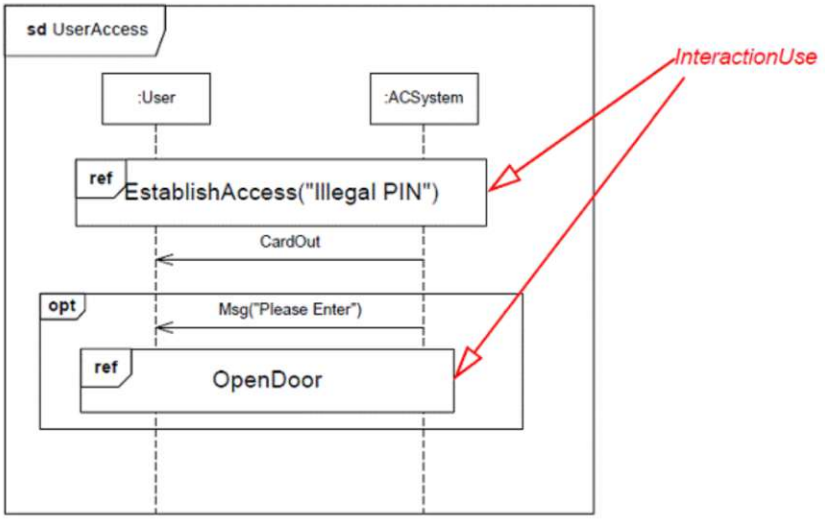


Figure 6.5: Interaction Use Notation [Gro17]

the surrounding components. The latter of these implementations better supports the user in the modeling process, as it requires fewer layout adaptions in a changing diagram. Like Combind Fragments, Interaction Uses must be aligned and scaled horizontally with their covering Lifelines.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 7

Artifact Implementation

This chapter describes the implementation process of the prototype of the sequence diagram modeling tool. It implements the modeling concept defined in chapter 6.

The tool builds upon and extends the current open-source UML modeling project BIGUML [bor23]. The BIGUML provides the core architecture of this project, on top of which the support for SD is implemented. At the current stage, BIGUML features support for the *class diagram*, *deployment diagram*, *information flow diagram*, *package diagram*, *state machine diagram*, and *use case diagram*. The implementation builds on the existing functionalities following its generalized concepts of nodes and edges where possible. Through the development process, some additional requirements specific to SD arose, which required some modification and additions to the underlying core functionalities. Whenever such additional functionalities were implemented or adapted, we ensured these were as generalized as possible to allow for application on any other model type, thus expanding the overall functionality of the BIGUML modeling capabilities.

The following sections provide a general implementation flow and project overview to guide the reader and support future implementors wanting to contribute to the project in their work. The section details implementations unique to SD and which challenges arose and describes the implementation of generalized interaction methods not specific to SD.

7.1 Project Architecture

From an implementation perspective, the BIGUML project [bor23] implements the GLSP architecture, which, as visualized in Figure 7.1, is divided into the three modules: *the model management*, *the server framework*, and *the client framework*. The GLSP protocol communicates between the GLSP client and the GLSP server.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

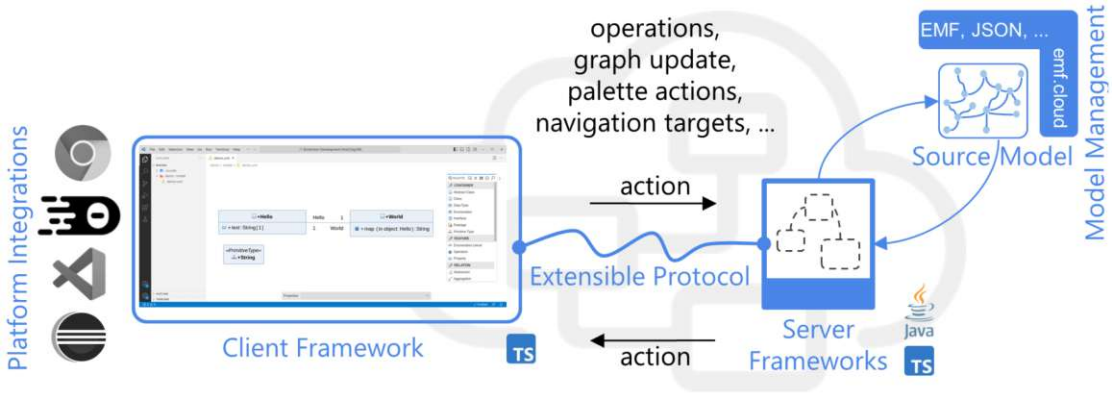


Figure 7.1: GLSP components and interactions [BLO23]

7.1.1 Model Management

Model management is overseen by the Model Server, responsible for handling the source model that defines elements, their properties, and their relationships. This source model serves as the underlying definition for the visualized diagram. The model server provides model access to the clients through model operations to fetch, create, update, save, and delete the models [Dos23].

For the project in the backend, the EMF.cloud model server is used to create and interact with the UML model following the EMF-based implementation of the UML metamodel from the Eclipse platform.

7.1.2 Server Framework

In the instance of BIGUML, the GLSP server is developed in Java. The server's architecture is further broken down to improve the separation of concerns and facilitate the extension of support for new diagrams and tools. Through the dependency injection, these components are loosely coupled as modules *core features, tool features*, and *diagram features* as visualized in Figure 7.2 [MB23].

This structure allowed us to bundle most of the implementations required to enable the modeling of SDs in their dedicated SD module within the diagram features. Throughout the development, some extensions and adaptions to the core and the feature module were necessary to cover SD-specific requirements not required by previous diagram types.

7.1.3 Client Framework

The Client Framework is responsible for rendering the user interface and defining the representation of model elements. It handles the user interaction and visualizes the interaction feedback. This module's implementation consists of adding styling definitions, rendering definitions, and the definition of new means to interact with the diagram. The

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

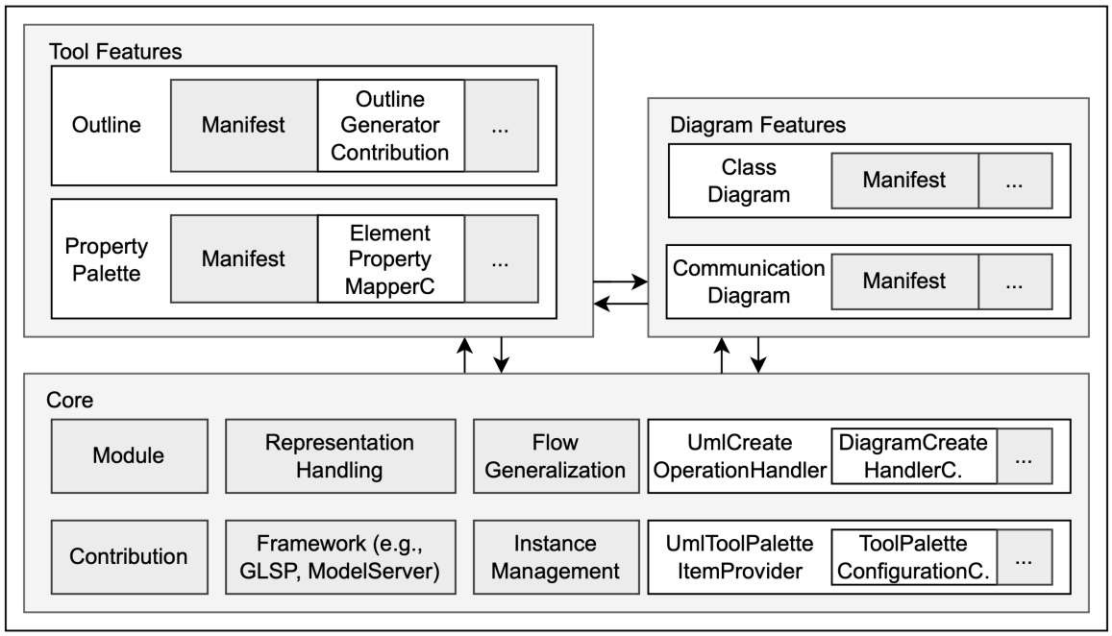


Figure 7.2: BIGUML server architecture [MB23]

client handles the user's interaction, such as key presses and mouse interaction, and maps them to the appropriate actions forwarded by the GLSP protocol to the server framework.

The GLSP client is implemented with TypeScript and integrates with Eclipse Theia and VS Code to provide the user interface. The visualization is implemented with Sprotty, an open-source diagramming framework [Eclb]. The diagram components are visualized as vector graphics generated as defined by their respective view. Views map the diagram model to its respective graphical representation following the JSX syntax to generate an SVG group styled with CSS.

7.2 Component Realization

The implementation applies a feature-driven development approach. The initial features being implemented were the components making up an SD, followed by features to improve interaction with the diagram. The order of implementation of the components followed the hierarchy structure of SD. We started with the *interaction* component, the *lifelines*, and a basic notation of *messages*. These form the foundation and basic capabilities of the SD. Support for *Execution Specifications*, *Combined Fragments*, and *Interaction Use* was subsequently added to the editor.

The implementation of each feature follows a similar implementation strategy. They are implemented following a bottom-up approach through the project architecture. We started with the element implementation for the model server to the client.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

7.2.1 Interaction

The SD is the depiction of an interaction, the interaction component as defined by the UML 2.5.1 specification [Gro17] contains all components of the sequence diagram as seen in the interactions abstract specification Figure 7.3. From the abstract specification, it becomes also evident that the diagram has a relatively flat hierarchical structure, with most components having a direct containment relation to the root interaction. Lifelines are not visualized in Figure 7.3 but are also contained in interactions as specified in Figure 7.7.

The sequence diagram interaction is depicted as a node as visualized in Figure 7.4 and scales dynamically to its content.

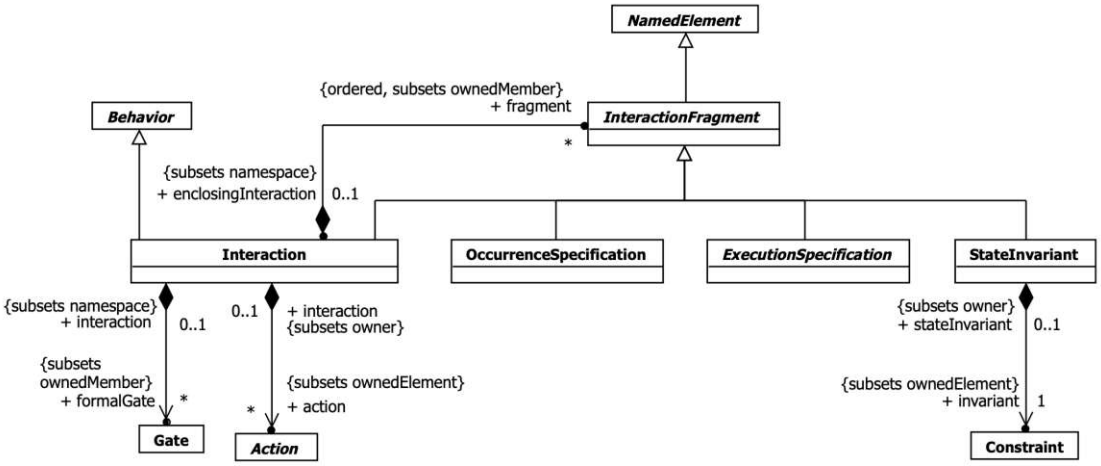


Figure 7.3: Interaction abstract syntax [Gro17]

Figure 7.4: Interaction representation

7.2.2 Lifeline

Lifelines are implemented as nodes containing a header with an extendable timeline below. The content of a lifeline aligns with the timeline. The timeline adapts its size to the content and can be extended manually by the user.

Lifelines are constrained to a fixed height within the interaction. They can only be moved horizontally. Vertical movement is constrained unless created within the interaction

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

through creation messages, in which case they align with the message's height as seen in Figure 7.5.

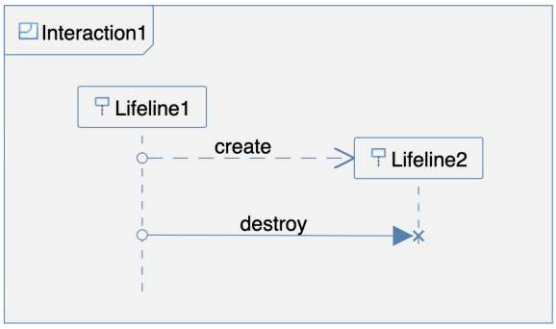


Figure 7.5: Lifeline representation

Lifelines can be created or destroyed by the respective message. As described in the concept in chapter 6, lifelines created within the interaction start at the height of their creation messages, and destructed lifelines are cut off at their Destruction Occurrence Specification as seen in Figure 7.5. The evaluation of a lifeline being created or destructed is implemented as part of the GLSP server depending on the messages of their *coveredBy* Message occurrence Specification as depicted in Listing 7.1.

Listing 7.1: Lifeline Creation & Destruction Check

```
1  private boolean isCreated(final Lifeline lifeline) {
2    return lifeline.getCoveredBys().stream()
3      .filter(f -> (f instanceof MessageEnd)
4        && ((MessageEnd) f).getMessage().getMessageSort() ==
             MessageSort.CREATE_MESSAGE_LITERAL
5        && ((MessageEnd) f).getMessage().getReceiveEvent() == f)
6      .count() > 0;
7  }
8
9  private boolean isDestructed(final Lifeline lifeline) {
10   return lifeline.getCoveredBys().stream()
11     .filter(f -> (f instanceof MessageEnd)
12       && ((MessageEnd) f).getMessage().getMessageSort() ==
             MessageSort.DELETE_MESSAGE_LITERAL
13       && ((MessageEnd) f).getMessage().getReceiveEvent() == f)
14     .count() > 0;
15 }
```

To support the rearranging of lifelines within the diagram, we implemented the logic for the modeling tool to detect overlapping lifelines and rearrange them accordingly, as defined by the fourth requirement for lifelines in our conceptualization. The tool's behavior is shown in Figure 7.6. The user can rearrange the lifelines by dragging one at the desired location shown in Figure 7.6a. The tool detects the order of the lifelines

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

and evenly distributes with a defined margin as shown in Figure 7.6b. This ensures an organized diagram throughout the modeling process without tedious adjustments by the user.

(a) Prior resolution (b) After resolution

Figure 7.6: Lifeline Overlap Resolution & Layout

7.2.3 Message & Message Occurrence Specification

Messages are represented as edges connecting lifelines along their timelines. The creation of messages along lifelines proved to be syntactically and technically challenging. The initial challenge concerned the modeling of messages along lifelines.

Following the UML syntax, as shown in Figure 7.8, messages are contained within an interaction and feature no direct reference to the lifelines. Instead, the connection is defined by the MessageEnds, which are the endings of a message and cover a specific lifeline. This is visualized by the abstract syntax of Messages in Figure 7.8 and Lifelines in Figure 7.7 as defined in the UML 2.5.1 specification [Gro17].

Technically, the GLSP diagram builds on the concept of nodes and edges, which connect to nodes. The position of the connection along a node can be defined through the placement of ports or through anchors [Imib]. These solutions define static predefined connection points, which do not meet the requirement for messages to be movable and added freely. Instead, we implemented messages through the implementation of Occurrence Messages Specifications as nodes created along the lifelines to represent the ends of messages. This implementation allowed messages to be created and moved freely along the lifelines.

This solution coherently aligned the syntactic and technical challenges. We resolved the syntactic challenge by modeling the Message Occurrence Specifications, which also resolved the anchoring difficulty by containing them within the lifelines, which means that the movement of the lifelines also moved the contained Message Occurrence Specifications.

In our implementation, the creation of messages now entails the creation of two Message Occurrence Specifications and their assignment as the source and target of the message.

Listing 7.2: Message Creation Compound Command

```
1 public CreateMessageCompoundCommand(final ModelContext context,
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

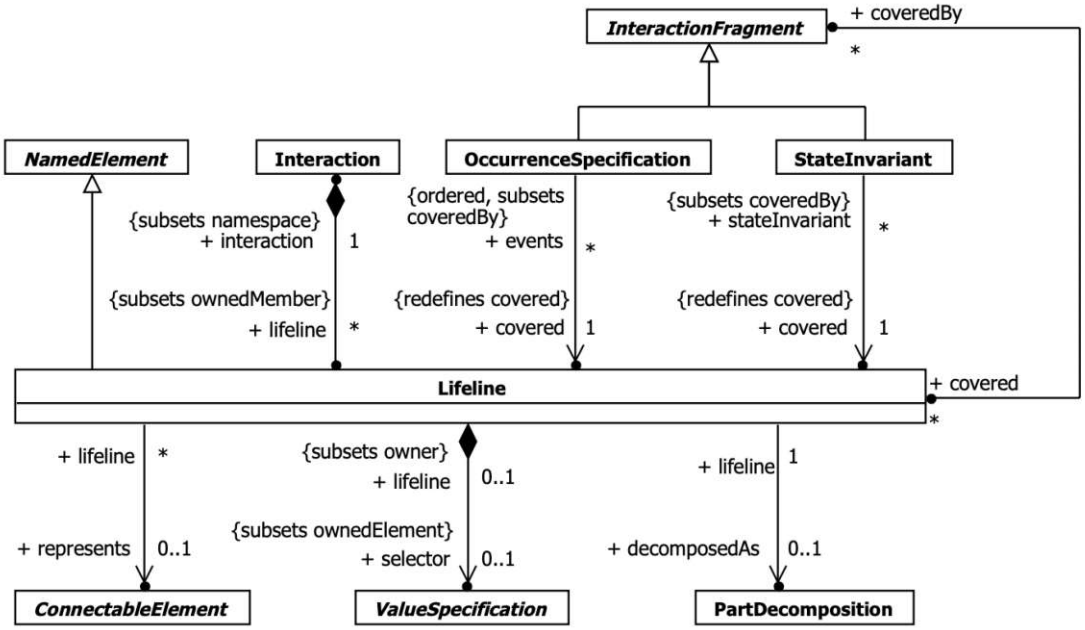


Figure 7.7: Lifeline abstract syntax [Gro17]

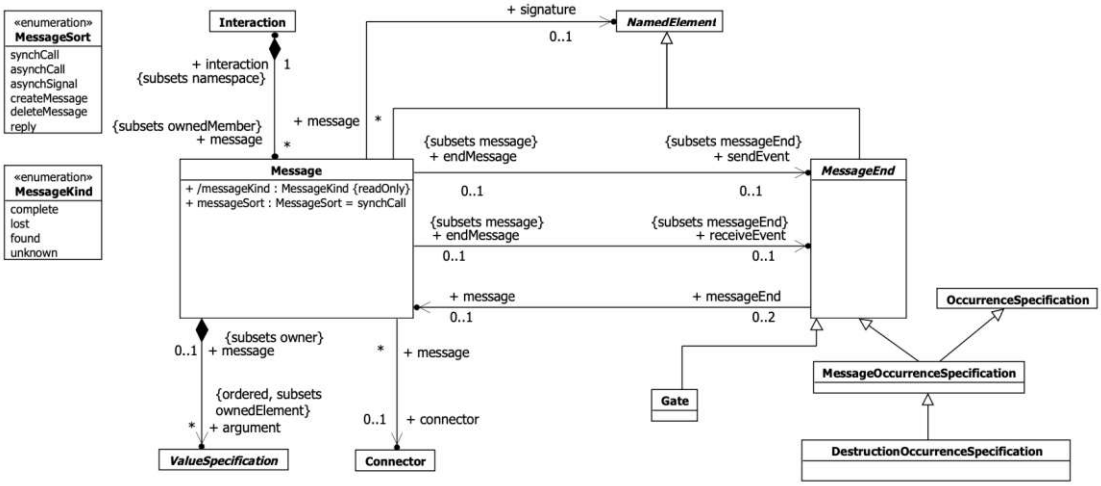


Figure 7.8: Message abstract syntax [Gro17]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
2       final Lifeline source, final Lifeline target, final GPoint
            sourcePosition, final GPoint targetPosition,
3       final UmlMessageSort sort, final UmlMessageKind kind) {
4       var createSourceCommand = new
            CreateMessageOccurrenceCompoundCommand(context, source,
5         sourcePosition);
6       this.append(createSourceCommand);
7       var createTargetCommand = new
            CreateMessageOccurrenceCompoundCommand(context, target,
8         targetPosition);
9       var command = new CreateMessageSemanticCommand(context,
            createSourceCommand::getSemanticElement,
10        createTargetCommand::getSemanticElement, sort, kind);
11      ...
12      this.append(command);
13      this.append(new AddEdgeNotationCommand(context,
            command::getSemanticElement));
14      ...
15      }
```

Creating the Message Occurrence Specifications requires the new message's position (lines 4 and 7) in Listing 7.2. This brought up the second challenge: The current BIGUML protocol did not include the input information for the source and target positions of edges, but only the source and target node objects.

The existing implementation in BIGUML for previously supported diagram types, such as class diagrams, did not require the position of edges but just their referenced nodes. This functionality is required for SD and was implemented on the client side by adding the *source position* and *target position* as additional arguments (lines 7-8) shown in Listing 7.3.

On the Model server side, these positions are required to create the respective Message Occurrence Specification nodes (lines 4 and 7) in Listing 7.2.

Listing 7.3: Edge Creation Operation

```
1  CreateEdgeOperation.create({
2     elementTypeId: this.triggerAction.elementTypeId,
3     sourceElementId: this.source,
4     targetElementId: this.target,
5     args: {
6        ...this.triggerAction.args,
7        sourcePosition: this.stringify(this.sourcePosition),
8        targetPosition: this.stringify(this.sourcePosition)
9     }
10 })
```

As stated in the established concept in chapter 6, to accurately model SD, the editor

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

needs to support all their variations of messages, meaning all kinds and sorts of messages as defined by the UML specification [Gro17].

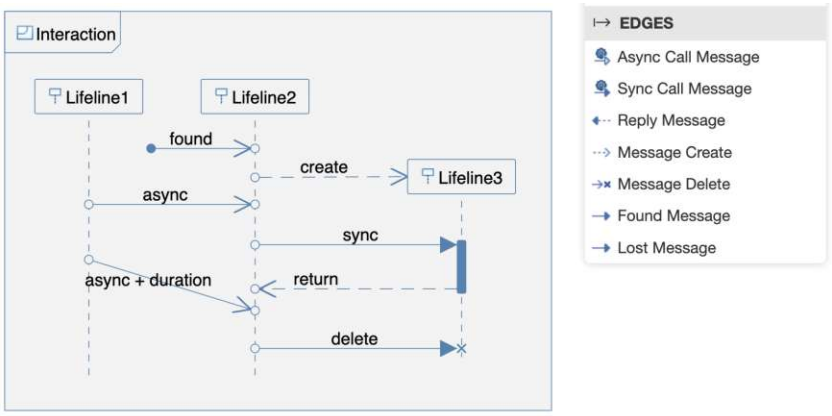


Figure 7.9: Messages

The peculiarity of messages of sort *create* and *delete* is their impact on their target lifeline. *Create messages* move the header of their created lifeline to their height as visualized in Figure 7.9. *Delete messages* instantiate a Destruction Occurrence Specification as their message end, at which height their target lifeline is cut off. The lifelines themself handle the behavior of scaling back to the destruction event.

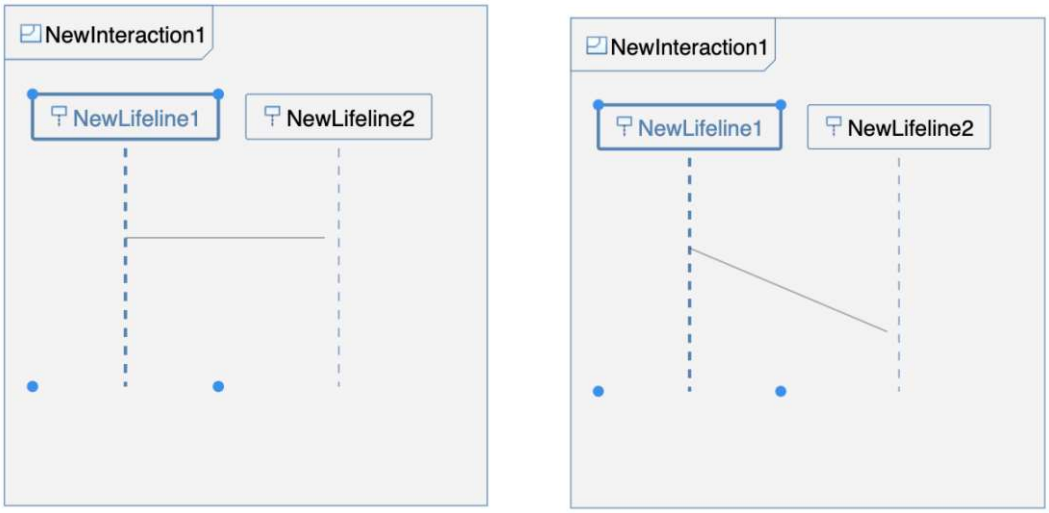
In the case of found and lost messages, these are messages which, according to the UML specification [Gro17], have no source or target. These, however, are both required to visualize an edge. To overcome this challenge, we extended the diagram model by a MessageAnchor node, depicted as a filled circle in the diagram but not instantiated in the UML model.

The final differentiation in messages is their duration. In SDs, most messages are modeled to be instantaneous by drawing them as a horizontal line. To depict messages with a duration, their target position is lower than their starting position, resulting in a downward diagonal. In our implementation, the created messages are drawn horizontally by default, with the target height matching the source height. This is to ensure the creation of perfectly horizontal lines without manual alignment. Creating diagonal lines is possible by holding "alt" as a modifier key, allowing the creation of messages with duration. The behavior is also appropriately reflected in the feature's preview feedback visualized in Figure 7.10b.

7.2.4 Execution Specifications

Execution specifications are modeled as thin nodes along the lifeline. They are automatically created with new synchronous messages. Their dimension is defined by the position of their incoming and outgoing messages as seen in Figure 7.11.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.



(a) Instant Message (b) Message with duration

Figure 7.10: Message Creation Preview Feedback

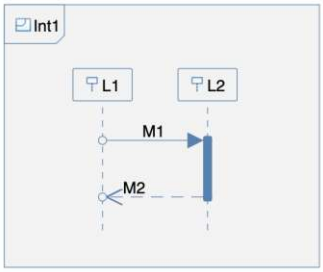


Figure 7.11: Execution Specification

7.2.5 Combined Fragment

Combined Fragments are contained within the interaction or combined fragments as shown in Figure 7.13. Following the specification in Figure 7.12, combined fragments contain interaction fragments such as interaction uses, occurrence specifications, execution specifications, and combined fragments.

The implemented structure of combined fragments reflects the UML specification [Gro17] and is shown in Figure 7.13. The node features a header defining the combined fragments operator. Depending on the defined operator, the combined fragments contain one or multiple interaction operands that stack within the node. The user can scale the combined fragment or the individual operands to contain messages visually.

Combined fragments provided a syntactical challenge regarding how to handle their contained components. The graphical model contains message occurrence specifications along lifelines and within combined fragments. Syntactically, message occurrence speci-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

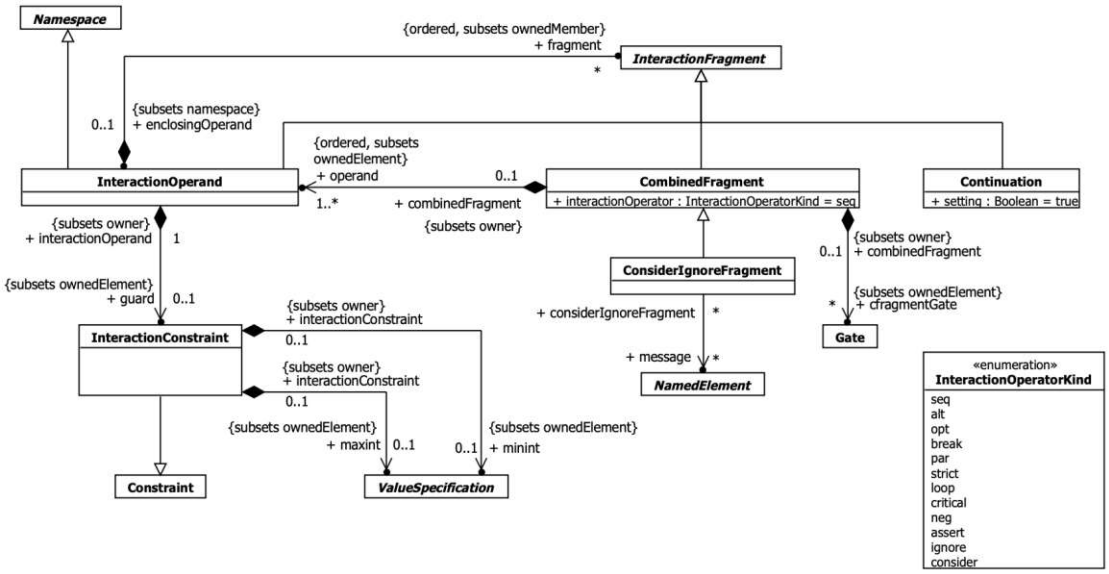


Figure 7.12: Combined Fragment Abstract Syntax [Gro17]

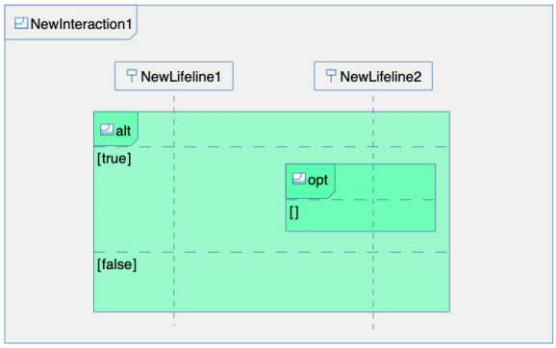


Figure 7.13: Combined Fragment

fications are contained within the interaction or combined fragments and reference as *covered* a lifeline as seen in Figure 7.7.

We implemented combined fragments as freely movable nodes whose movement applies to their contained combined fragments and interaction uses. As stated, Occurrence Specification nodes are contained within the lifeline nodes to ensure alignment rather than being contained within the combined fragments and their interaction operands.

This implementation ensures a consistent anchoring of messages and lifelines. Lifelines dynamically scale to the movement of messages, and message occurrence specifications reliably follow the movement of their referenced lifelines, providing the user with consistent visual feedback and interactions.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

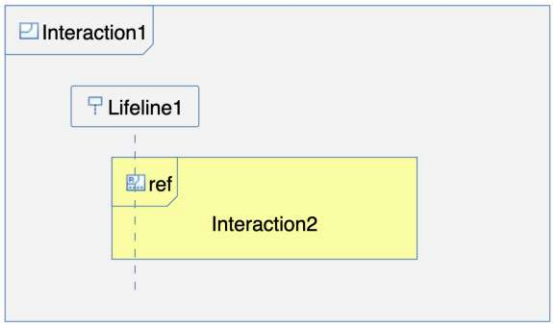


Figure 7.14: Interaction Use

7.2.6 Interaction Use

Interaction Uses are implemented as freely movable nodes within interactions or combined fragments as shown in Figure 7.14. The referenced interaction is implemented textually. At the current stage of the project, linking across models is not supported. In our implementation, the user can freely move Interaction Uses.

7.3 Behavior Realization

In this second realization section, we focus on user interaction, which was implemented to aid the user in modeling. These implementations follow the interactions already conceptualized in the chapter 6.

7.3.1 Semantic Selection

As defined in the conceptualization in chapter 6, the semantic selection feature automatically adds dependent components to the user selection. In the case of the SD, this would mean that upon the user selecting a message, the following messages are added to the selection.

In our implementation, the semantic selection consists of a recursive evaluation and selection. The dependent components of the initially selected component are added to the selection, and the added components are checked for additional dependencies. This evaluation process is repeated until no additional components can be added (line 28) in the Listing 7.4.

The dependency of each component is defined as follows:

- *Message* -> its Message Ends (*Message Occurrence Specifications*)
- *Message Occurrence Specifications* -> all the following Occurrence Specifications on their lifeline, their opposite Message Occurrence Specification if it is at the same time or later, and the newly created lifeline if it is part of a Create Message

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

- *Lifeline* -> its covering fragments

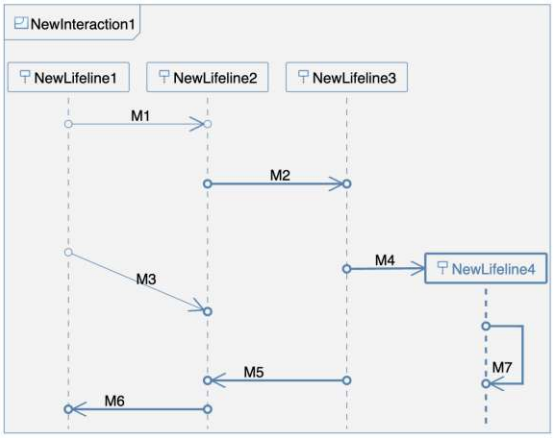


Figure 7.15: Semantic Selection Example

The resulting semantic selection is visualized in Figure 7.15. The message *M2* was manually selected by the user, based on which its dependent components *M4*, *NewLifeline4*, the *messageEnd of M3*, *M7*, *M5*, and *M6* were automatically appended to the selection. The movement of *M2* moves all semantically selected components.

The semantic selection is implemented as part of the server framework due to the requirement for its semantic awareness. One minor downside of this implementation is a delay upon selection due to the query for the additional components to be selected.

Listing 7.4: Semantic Selection

```
1  private List<String> collectElements(final int i, final List<String>
       selectedElementsIds,
2       final List<String> checkedElementIDs,
3       final EMFModelIndex currentModelIndex) {
4       List<String> addToSelectionIDs = new ArrayList<>();
5       selectedElementsIds.removeIf(t -> t == null);
6       for (String elementId : selectedElementsIds) {
7         if (checkedElementIDs.contains(elementId)) {
8           continue;
9         }
10        checkedElementIDs.add(elementId);
11        var semanticElement =
              currentModelIndex.getEObject(elementId).get();
12        if (semanticElement == null) {
13          continue;}
14        if (semanticElement instanceof message) {
15          checkElement(addToSelectionIDs, (Message) semanticElement);
16          continue;}
17        if (semanticElement instanceof lifeline) {
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
18            checkElement(addToSelectionIDs, (Lifeline) semanticElement);
19            continue;}
20          if (semanticElement instanceof OccurrenceSpecification) {
21            checkElement(addToSelectionIDs, (OccurrenceSpecification)
                  semanticElement);
22            continue;}
23        }
24        selectedElementsIds.addAll(addToSelectionIDs);
25        removeDuplicates(selectedElementsIds);
26        removeDuplicates(checkedElementIDs);
27        if (checkedElementIDs.containsAll(selectedElementsIds)) {
28          return new ArrayList<>();}
29        selectedElementsIds.addAll(collectElements(i + 1,
              selectedElementsIds, checkedElementIDs, currentModelIndex));
30        removeDuplicates(selectedElementsIds);
31        return selectedElementsIds;
32    }
```

7.3.2 Shifting Feature

As described in the concept in chapter 6, the shifting feature allows one to select multiple components based on their position and move them vertically or horizontally. The primary use case of this feature is to organize the diagram by allowing the user to easily create or close gaps in the diagram with one click and drag.

The feature is fully implemented within the GLSP client, meaning it can be efficiently utilized for any other diagram type as exemplarily visualized in Figure 7.16 on a class diagram.

Upon releasing the mouse, the horizontal or vertical movement distance is collected, and a movement operation is issued for all selected components by the defined distance.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

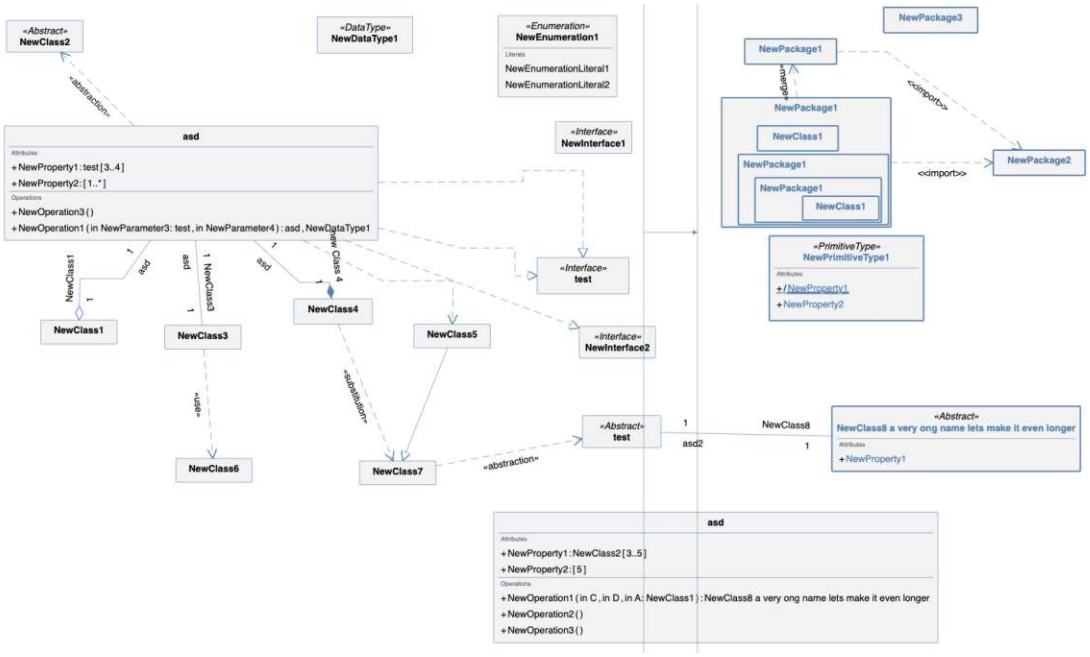


Figure 7.16: Shifting Feature Example on Class Diagram

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 8

Artifact Evaluation

In this chapter, we evaluate the developed artifact. The evaluation is broken down into two steps. As an initial step, we evaluate the essential functionalities of the implemented tool. We utilized the same evaluation methodology as the researched existing tools to ensure a consistent evaluation and obtain comparable results.

The second part of our evaluation focuses on the defined concepts in chapter 6. We check our success in realizing them within the final artifact. We conclude the evaluation by discussing our findings and reflecting on possible future improvements.

8.1 Modeling Process-based evaluation

Following the modeling process defined in Figure 5.2, we modeled the reference SD Table 5.1. The outcome of the modeling process closely resembles the expected outcome defined in Figure 5.2. It contains all required components except the duration constraint, which is not yet supported. Going through the process step by step ensured the completeness of our artifact and highlighted possible missing functionalities, which were collectively reflected upon following the defined questions defined in Table 5.3. Following the modeling steps and the associated questions, we came to the following answers:

Steps 3 and 9: Is the role: class naming scheme supported? Are multiple types, such as actors, of lifelines supported? Is the stereotype notation supported?

> Our modeling tool does support the role: class naming scheme. The definition of a stereotype for lifelines is not supported in the tool. The representation of specific stereotypes iconography is not featured.

Step 4: Are labels editable on the diagram or in the properties?

> The labels can be edited directly within the graphical diagram and in the components properties tab.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Steps 5 and 6: Is it possible to move lifelines horizontally and vertically? Is the movement fluent or in steps (grid)?

Lifelines can be freely moved horizontally and organized by the user. Following the SD layout, lifelines cannot be moved vertically. If attempted, they snap back to their defined height in alignment with all other lifelines. By default, the tool has no grid; all components can be moved fluently.

Step 10: Do the lifelines overlap, is the overlap automatically resolved (by moving L3 to the right)?

The editor automatically handles the overlap of lifelines. The modeling tool ensures an organized layout and spacing between lifelines by enforcing consistent margins. The user can still easily reorder lifelines by dragging and dropping them in the desired location. The required space in the desired location is adjusted automatically by shifting overlapping lifelines.

Step 14: Are response messages (M18) automatically created? Are execution specifications created and visualized? Can execution specifications be added manually? Are the function brackets "()" added automatically or manually? Where can messages be created? Is the creation location constrained? Does the position follow a grid/row layout?"

The response message, as well as the execution specification, is automatically created on the creation of a synchronous message. Function brackets are not added automatically to the message label.

Step 16: Does the created lifeline align to the message height?

Lifelines created within the interaction by a *create message* automatically align to the message and can be adjusted in height by the user. If the *create message* is deleted, the lifeline jumps back to its original height.

Step 18: Does the creation of the message cause the following elements to shift down? Does the tool create additional space for the new element between existing elements? Is there a margin constraint/enforcement between the existing and the newly created message?

By adding new messages or components, the components below are not affected. No margin or constraints are applied to the components. The editor features the shifting feature to create additional space for new components or reduce excessive spacing between components. With it, the user can quickly adjust the diagram as required.

Step 19: Are messages with duration supported by the tool?

Messages with duration, which are represented as diagonal messages, are supported by the editor. By default, messages are instantaneous. Holding the alt-key allows the user to draw messages with duration freely.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Steps 23 and 26: Does the combined fragment interact with other diagram elements? Is the covering of the lifelines recognized? Is the change in lifeline covering recognized?

Combined fragments interact with their contained combined fragments and interaction uses, which follow their movement. Combined fragments do not interact with the lifelines or cover them.

Steps 24 and 25: Are there constraints in the movement? Does the movement move other elements? Does the CF1 movement minimize the overlay of objects?

Combined fragments are freely movable by the user. The modeling tool does not resolve overlaps of combined fragments.

Step 30: Do combined fragments drawn over messages also contain them?

Combined fragments do not contain message or message occurrence specifications.

Step 33: Does the target lifeline scale back to the height of the destruction message? Does L3 automatically receive a destruction symbol at its end? How does the tool handle M17 below M18?

Following the SD specifications, the destruction event is required to be the last component of a lifeline, and the user is, therefore, unable to create a destruction message if the above components of the target lifeline. Upon targeting a given lifeline by a destruction message, a destruction event is created, and the lifeline is automatically cut off to that height as shown in Figure 8.1 for lifeline *L3*.

Step 34: Does the tool support messages to self?

The modeling tool supports messages to self, and the routing of the message is handled appropriately, as recognizable in Figure 8.1 for message *M19*. Moving a message to self dynamically adapts its routing points.

Step 35: Does the deletion of the messages cause a vertical movement of the other elements? Is the freed-up space reduced?

The removal of a message does not affect other components. The exception is removing create messages, after which their created lifeline jumps back to the default position.

Summarizing the reflection on the provided questions and the overall modeling process, we can confidently state that the tool supports all required essential modeling components except duration constraints as recognizable from the tools palette in Figure 8.1. The editor reliably provides interaction behaviors such as creating, deleting, modifying, and moving components. The editor offers SD-specific behaviors and constraints such as the constraining messages between lifelines, the routing of messages to self, the anchoring of lifelines at the top, the shortening of lifelines at their destruction event, and the automatic creation of execution specifications and return messages for synchronous messages. The evaluation shows a partial implementation of automatically handling the overlapping of components. The tool automatically handles lifelines' positioning and layout, ensuring

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

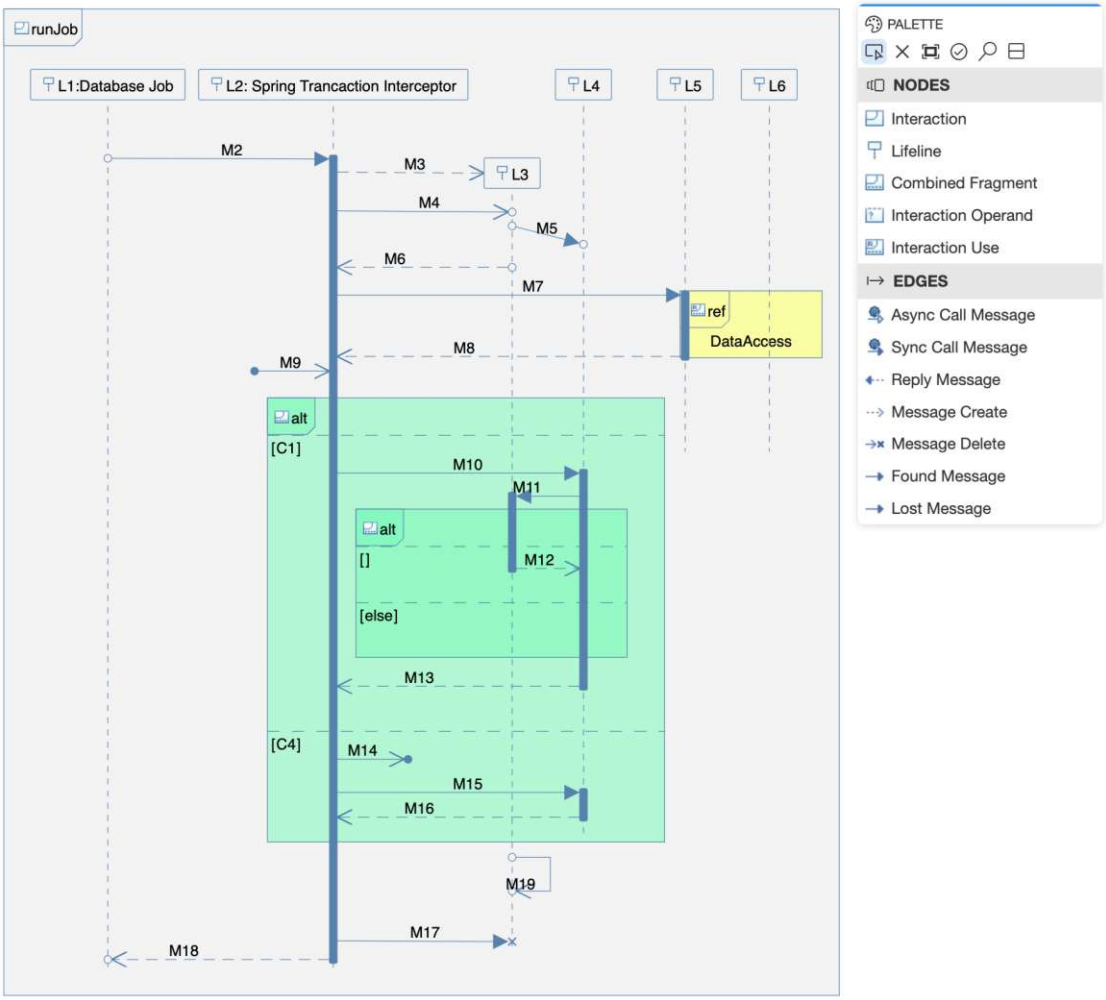


Figure 8.1: bigUML Sequence Diagram

consistent spacing between them in the desired order. The overlap and other layout functionalities of other components, such as combined fragments or messages, are not supported at the current stage of development. The tool provides features to aid the user in maintaining a well-structured layout through the shifting and semantic selection features.

8.2 Requirements Evaluation

In the second part of our evaluation, we verify to what extent the concepts for the individual diagram components defined in chapter 6 are supported in the final implementation. These concepts go beyond the essential functionalities, such as creating individual components and focusing on more advanced modeling behaviors and layout interaction

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

between different components. In Table 8.1 we collect all requirements from the Tables 6.1, 6.2, 6.5, and 6.4. For each requirement, we evaluate whether our implementation supports (●), partially supports (◐), or does not support it (○), as indicated with the respective iconography.

Table 8.1: Artifact Requirements Support

#	Requirement	Support
Interaction Requirements:		
1	The SD contains a single interaction	●
2	Interaction scales automatically to include all SD components	●
3	Empty interaction is created with a new model	○
Lifeline Requirements:		
1	Lifelines are locked at a fixed height unless they are created	●
2	The length of a lifeline should scale based on its content	●
3	Destructed lifelines have to terminate at the location of their destruction message	●
4	The overlap of lifelines is resolved by enforcing a horizontal margin between them	●
5	Lifelines should be horizontally ordered according to their moment of interaction	○
6	The label of cut-off lifelines is shown in a toolbar over the modeling view	○
Messages Requirements:		
1	By default, messages should be created as horizontal lines with no duration	●
2	The user should be able to reorder messages	●
3	Moving a message should maintain its being horizontal	●
4	Support for messages with duration	●
5	Messages should have vertical margins between them to ensure good readability	○
6	When adding a message, the components below should move down to provide space and sufficient margin between them	○
7	Message overlap should be handled by the editor	○
8	Message crossing should be minimized	○
Exectuiron Specification Requirements:		
1	Execution Specifications are automatically created	●
2	Execution Specifications adapt their length to their respective message's position	●
3	Execution Specification stack when overlapping as in Figure 6.4	○
4	Execution Specifications can be hidden	○
Combined Fragments Requirements:		
1	Components can be created within combined fragments	●

●= *supported*, ◐= *partially supported*, ○= *not supported*

Continue on the next page

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Table 8.2: Artifact Requirements Support (cont)

#	Requirement	Support
2	Moving a combined fragment vertically also moves its content	◐
3	Moving a combined fragment can alternatively be moved freely without their content	◐
4	Combined fragments scale automatically to include their content	◐
5	Combined fragments scale with the horizontal movement of covered lifelines	○
6	Combined fragments can be drawn over existing components and will contain them	○
7	Messages cannot cross the combined fragments frame	○

●= *supported*, ◐= *partially supported*, ○= *not supported*

The table shows that our artifact supports all essential conceptualized functionalities for components. The tool incorporates sophisticated layout behaviors, including automated overlap resolution and automated layout for lifelines. For handling the layout of other elements, such as messages, the tool offers layout functionalities through the *Shifting* and *Semantic Selection* features, facilitating manual layout adjustments.

8.3 Discussion

Upon reviewing the evaluation, our tool demonstrates a notable blend of flexibility and power. Unlike the heavyweight and complex modeling tools within the third category of Table 5.8, our tool is web-based, obviating the need for local workstation installations and allowing the deployment over the web as a theia-based editor or as the currently available extension to VS Code.

In contrast to the web-based modeling tools typically associated with categories one and two from the Table 5.8, our tool addresses the requirements outlined in category three and partially in category four. Our tool provides a combination of robust modeling functionalities and the convenience of web-based features, including seamless instant deployment and ease of use.

Moreover, the web-based nature of our tool paves the way for future expansion, enabling the incorporation of features like real-time collaborative diagram editing. Such collaborative functionality is only found in web-based modeling tools within categories one and two of Table 5.8. The more extensive workstation-based modeling tools within categories three to five of Table 5.8 have no support for such functionalities.

Based on the evaluation, we can classify our implementation following the established tool classification in Table 5.8. This guides comparable modeling tools and outlines commonalities and differences between our solution and existing ones.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Our implementation features predefined diagram components with their specific properties and behavior as required by category one.

Following the criteria for the second category, Lifelines are aligned and anchored along the top side of the diagram. Execution Specifications are centered along the timeline of the lifeline. Messages are appropriately anchored along their source and target lifeline, depending on the message's sort and kind as defined by the UML specification [Gro17].

As required for modeling tools of the third category, when modeling execution specifications and response messages are appropriately created with synchronous messages. The user is supported in aligning messages by ensuring that messages are horizontal upon creation. Additionally, the user can rely on the semantic selection and shifting features to move and arrange messages accurately.

Our implementation only partially meets the requirements for a category four tool. Our tool implements the integration of combined fragments and their behavior with other components only partially. Combined fragments can contain other combined fragments as well as interaction uses. Message occurrence specifications and execution specifications cannot be contained within combined fragments, as they are contained within lifelines. Moving a combined fragment also moves its content accordingly. The tool resolves overlaps and supports automatic layouts of lifelines, ensuring an organized diagram. This feature is not yet supported for combined fragments or messages. Currently, this functionality is only covered by more restrictive modeling tools.

Categorizing our modeling tool according to its supporting modeling features and constraints as defined in Table 5.8 would best place it into the third category as it meets all requirements of categories one through three.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 9

Conclusion

In section 4, we conducted an exhaustive exploration and compilation of the prevailing SD modeling tools accessible in the market. We systematically categorized these tools to guide the selection of the most suitable option.

To methodically gather modeling behaviors that are particularly pertinent to SD, we formulated an idealized modeling process in Table 5.1, which encompasses the most pertinent modeling behaviors. This comprehensive evaluation offered valuable insights into the underlying modeling principles inherent in each tool. These insights, in turn, served as the basis for the imagination of a generalized modeling concept, which places a central emphasis on the essential modeling behaviors and practices.

The established concept guided the implementation of the established modeling behaviors. Our build artifact adds support for modeling SDs to the BIGUML modeling editor [bor23], deployed as a VS Code extension with the support for SDs. Our implementation consists of support for the SD and all its essential components as well as addition to BIGUML core functionalities such as the addition of the shifting feature.

Our evaluation in chapter 8 showed our artifact can compete in features and functionalities with existing modeling solutions. The added value our modeling tool provides is the combination of tool completeness, functionality, and flexibility.

Our research and implementation ultimately allow us to answer the four initially posed researched questions in chapter 1.

RQ1: What editing behaviors and interactions do current SD modeling tools offer?

During the evaluation chapter, we uncovered a broad spectrum of editor support and functionalities, based on which we categorized the evaluated modeling tools in

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Table 5.8. This categorization clusters the tools by their supported interactions and behaviors specific to SDs.

General graphical modeling tools, while not offering SD-specific modeling behaviors, offer supporting functionalities such as real-time collaboration and ease of access over the browser. This makes modeling accessible to multiple users from different devices.

SD-specific modeling tools provide responsive SD-specific behaviors, including the reactive movement of all component-dependent components and reactive movements to avoid overlapping messages or lifelines.

RQ2: Which behaviors significantly increase manual interactions by the user and encumber the SD modeling process?

The answer to this question hinges on the nature of the tool employed by the user. When dealing with unconstrained editors, the meticulous management of shapes and their precise alignment constitutes a substantial portion of the user's workload. This is especially true if the modeling tool does not provide predefined diagram-specific shapes, such as a lifeline, as the elementary building blocks of the diagram. The primary focus of the modeler; instead, it should be delegated to the editor.

For editors featuring preassembled SD components that automatically align and connect, the primary challenge shifts to the vertical alignment and positioning of messages. Resolving overlapping elements, such as messages and combined fragments, can swiftly become a source of frustration and a time-consuming endeavor. Moreover, modifying the diagram at its inception may necessitate significant efforts from the user to adjust all subsequent components.

In the context of a highly responsive editor that reacts dynamically to component movements and adjusts the diagram in response to user interactions, the assistance provided can potentially become overwhelming for the user if its functionality lacks transparency. In the most unfavorable scenarios, this may culminate in disputes over component positioning and snapping between the user and the editor, reaching a point where the user might prefer no tool support.

RQ3: Which concepts for tool behavior can reduce the necessary modeling user interactions and, in that respect, improve the efficiency of the SD modeling tool?

Throughout the evaluation process of the existing tools and the following conceptualization, we found a selection of modeling interactions that benefit the modeling of SD. The interactions most beneficial to the user are the ones overcoming the most encumbering behaviors identified in *RQ2* while providing the best resolution.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

A foundational requirement for a modeling tool to be efficient is to provide the user with prebuilt diagram components as defined by the UML specification [Gro17]. These components must be ready to use as fixed building elements.

RQ4: What advantages and disadvantages, concerning the prospect of new functionalities and efforts of implementation, does the implementation of GLSP-based modeling bring in comparison to existing tools?

The modular design of GLSP presents a distinct advantage in delineating interactions specific to diagram types versus global interactions. This is well-illustrated by the functionality of the horizontal and vertical shifting feature. While this feature holds particular relevance for SD, its applicability extends to various diagram types, benefitting effective component layouts. GLSP, as a flexible platform accommodating various modeling languages, facilitates the seamless integration of new features with manageable supplementary efforts. The cumulative integration of these enhancements fortifies GLSP as a versatile platform, enhancing its utility across all implementations.

The modular nature of GLSP gives ample possibilities for its deployment as it is ready-to-use integrations with VS Code, Eclipse Theia, and Eclipse desktop. While BIGUML is currently being developed as a VS Code extension, adding other deployment means involves only minor adaptions. This positively impacts the tool's accessibility, effectively making it a cross-platform application and opening it up to many more users and devices.

In summary, this thesis provides a comprehensive exploration of the extensive array of existing SD modeling tools and their inherent diversity. We offer a methodically structured overview and assessment of these tools, effectively mapping the current landscape of available solutions. By introducing a set of criteria for functionalities, we categorize the most widely used and generally accessible tools (refer to Table 5.8). The in-depth evaluation of these tools in chapter 5 yields valuable insights into their individual functionalities, empowering users to make informed choices of tools.

Throughout the evaluation, we identify and analyze the prevalent modeling behaviors employed across these tools. These behaviors serve as a foundational element, subsequently informing our concept. This concept, in turn, refines and extends these behaviors into a set of component-specific requirements, aimed to enhance the overall modeling experience and efficiency.

Based on our findings and concept we successfully developed and evaluated a new SD modeling tool employing all essential SD modeling behavior with the addition of some specialized behaviors. Our artifact is being deployed as part of the complete UML modeling editor BIGUML. The editor is released as a VS Code extension as an open-source project [bor23].

Building upon our conceptual foundation and research findings, we successfully developed and evaluated a novel SD modeling tool. This tool not only incorporates all essential SD

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

modeling behaviors but also introduces specialized functionalities. Our artifact seamlessly integrates into the complete UML modeling editor, bigUML, which is released as an open-source project [bor23] as a VS Code extension [BIG].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

List of Tables

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Listings

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

[AL17] Luciane Agner and Timothy Lethbridge. A survey of tool use in modeling education. MODELS '17, pages 303–311. IEEE Press, 2017. Book Title: 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS).

[Ala07] Alan R Hevner. A Three Cycle View of Design Science Research. *Scandinavian journal of information systems*, 19(2):4–, 2007. Place: Aalborg Publisher: Association for Information Systems.

[ALS19] Luciane T. W. Agner, Timothy C. Lethbridge, and Inali W. Soares. Student experience with software modeling tools. *Software and systems modeling*, 18(5):3025–3047, 2019. Place: Berlin/Heidelberg Publisher: Springer Berlin Heidelberg.

[ATB03] Auer, Tschurtschenthaler, and Biffl. A flyweight UML modelling tool for software development in heterogeneous environments. In *2003 Proceedings 29th Euromicro Conference*, pages 267–272, September 2003. ISSN: 1089-6503.

[Bel04] Donald Bell. Explore the UML sequence diagram, 2004. https://developer.ibm.com/articles/the-sequence-diagram/. [Accessed: 9/26/2023].

[BIG] BIGUML. bigUML Modeling Tool - Visual Studio Marketplace. https://marketplace.visualstudio.com/items?itemName=BIGModelingTools.umldiagram. [Accessed: 11/10/2023].

[BL23] Dominik Bork and Philip Langer. Catchword: Language Server Protocol An Introduction to the Protocol, its Use, and Adoption for Web Modeling Tools. *Enterprise Modelling & Information Systems Architectures*, 18(ISSN: 1866-3621), 2023.

[BLO23] Dominik Bork, Philip Langer, and Tobias Ortmayr. A Vision for Flexibile GLSP-based Web Modeling Tools. In *16th IFIP WG 8.1 Working Conference on the Practice of Enterprise Modelling*. Springer International Publishing, July 2023. arXiv:2307.01352 [cs].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[bor23] borkdominik. bigUML Modeling Tool, August 2023. https://github.com/borkdominik/bigUML. [Accessed: 8/31/2023].

[Buc] Paul Buck. Theia Adopter Story: logi.CLOUD, a modern engineering platform for industrial automation | Eclipse Foundation Staff Blogs. https://blogs.eclipse.org/post/paul-buck/theia-adopter-story-logicloud-modern-engineering-platform-industrial-automation. [Accessed: 8/30/2023].

[Cab22] Jordi Cabot. UML tools : Our curated selection of free, online, OSS, for MAC,... tools, February 2022.

[DCLB22] Giuliano De Carlo, Philip Langer, and Dominik Bork. Advanced visualization and interaction in GLSP-based web modeling: realizing semantic zoom and off-screen elements. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*, MODELS '22, pages 221–231, New York, NY, USA, October 2022. Association for Computing Machinery.

[DI16] Quang-Vinh Dang and Claudia-Lavinia Ignat. Performance of real-time collaborative editors at large scale: User perspective. In *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 548–553. IEEE, 2016.

[Dos23] Nina Doschek. *Managing Concurrent Heterogeneous Editing in Web-Based Modeling Tools*. Wien, 2023.

[Ecla] EclipseFoundation. GLSP. https://www.eclipse.dev/glsp/. [Accessed: 8/29/2023].

[Eclb] EclipseFoundation. Sprotty - Diagram Visualization Tools for Your Web Applications. https://sprotty.org/. [Accessed: 10/9/2023].

[Ecl22] EclipseSource. UML Editor based on GLSP, May 2022. https://github.com/eclipsesource/uml-glsp. [Accessed: 6/7/2022].

[EDDT11] Mohammed El Dammagh and Olga De Troyer. Feature Modeling Tools: Evaluation and Lessons Learned. In Olga De Troyer, Claudia Bauzer Medeiros, Roland Billen, Pierre Hallot, Alkis Simitsis, and Hans Van Mingroot, editors, *Advances in Conceptual Modeling. Recent Developments and New Directions*, Lecture Notes in Computer Science, pages 120–129, Berlin, Heidelberg, 2011. Springer.

[EES09] Holger Eichelberger, Yilmaz Eldogan, and Klaus Schmid. A Comprehensive Survey of UML Compliance in Current Modelling Tools. 2009.

[EES11] Holger Eichelberger, Yilmaz Eldogan, and Klaus Schmid. A comprehensive analysis of UML tools, their capabilities and their compliance (2011).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

2011. Publisher: Institute for Computer Science, University of Hildesheim, Germany.

[Faka] Kirill Fakhroutdinov. Spring and Hibernate transaction UML sequence diagram example. https://www.uml-diagrams.org/examples/spring-hibernate-transaction-sequence-diagram-example.html?context=seq-examples. [Accessed: 9/21/2023].

[Fakb] Kirill Fakhroutdinov. UML sequence diagrams overview of graphical notation - lifeline, message, execution specification, interaction use, etc. https://www.uml-diagrams.org/sequence-diagrams.html. [Accessed: 8/25/2023].

[FDSP05] A. Funes, A. Dasso, C. Salgado, and M. Peralta. UML Tool Evaluation Requirements. 2005.

[Fow04] Martin Fowler. *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley object technology series. Addison-Wesley, Addison Wesley Professional, Boston, Mass., [Boston, Mass.], 3rd ed. edition, 2004.

[Gee17] GeeksforGeeks. Unified Modeling Language (UML) | Sequence Diagrams, October 2017. Section: Design Pattern.

[Gro17] Object Management Group. UML2.5.1 - About the Unified Modeling Language Specification Version 2.5.1, December 2017. https://www.omg.org/spec/UML. [Accessed: 6/3/2022].

[HMPR04] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, 2004. Publisher: Management Information Systems Research Center, University of Minnesota.

[Hoo13] Gregor Hoops. *Automatic layout of UML sequence diagrams*. Christian-Albrechts-Universität zu Kiel, 2013.

[Hop18] SJBA Hoppenbrouwers. From Expert Discipline to Common Practice: A Vision and Research Agenda for Extending the Reach of Enterprise Modeling. *Business & information systems engineering*, 60(1):69–80, 2018. Place: Wiesbaden Publisher: Springer Fachmedien Wiesbaden.

[Imia] Imixs. Open-BPMN - Documentation. https://www.open-bpmn.org/. [Accessed: 8/30/2023].

[Imib] Imixs. Open BPMN - Documentation – Anchors & Ports. https://www.open-bpmn.org/glsp-client/SPROTTY_ANCHORS_AND_PORTS.html. [Accessed: 10/9/2023].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[Ing17] Yanic Inghelbrecht. TraceModeler - UML Sequence Diagram Editor for Professionals - Easy-To-Use and Smart, 2017. http://www.tracemodeler.com/. [Accessed: 10/2/2023].

[Jec] Mario Jeckle. Unified Modeling Language (UML) Tools. http://www.mario-jeckle.de/umltools.html. [Accessed: 8/25/2023].

[KL22] Reyhaneh Kalantari and Timothy C. Lethbridge. Characterizing UX Evaluation in Software Modeling Tools: A Literature Review. *IEEE Access*, 10:131509–131527, 2022.

[Lim05] Andreas Limyr. Graphical editor for UML 2.0 sequence diagrams. 2005. Accepted: 2013-03-12T08:04:43Z.

[MB23] Haydar Metin and Dominik Bork. On Developing and Operating GLSP-based Web Modeling Tools: Lessons Learned from bigUML. In *Proceedings of the 26th International Conference on Model Driven Engineering Languages and Systems, MODELS 2023*. IEEE, 2023.

[Mica] Microsoft. Language server protocol implementations. https://microsoft.github.io/language-server-protocol/implementors/servers/. [Accessed: 8/7/2023].

[Micb] Microsoft. Language server protocol specifications. https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/. [Accessed: 8/4/2023].

[Micc] Microsoft. Official page for language server protocol. https://microsoft.github.io/language-server-protocol/. [Accessed: 8/4/2023].

[MW08] Zoltan Micskei and Hélène Waeselynck. *UML 2.0 Sequence Diagrams' Semantics*. August 2008.

[NPA16] Oksana Nikiforova, Sergii Putintsev, and Dace Ahiļčenoka. Analysis of Sequence Diagram Layout in Advanced UML Modelling Tools. *Applied Computer Systems (Online)*, 19(1):37–43, 2016. Publisher: De Gruyter Open.

[OMG17] OMG. Unified Modeling Language, v2.5.1. *Unified Modeling Language*, page 796, December 2017.

[Ozk19] Mert Ozkaya. Are the UML modelling tools powerful enough for practitioners? A literature review. *IET software*, 13(5):338–354, 2019. Place: HERTFORD Publisher: The Institution of Engineering and Technology.

[Par] Visual Paradigm. What is Sequence Diagram? https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/. [Accessed: 9/25/2023].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[PC20] Elena Planas and Jordi Cabot. How are UML class diagrams built in practice? A usability study of two UML tools: Magicdraw and Papyrus. *Computer Standards & Interfaces*, 67:103363, January 2020.

[REIWC18] Roberto Rodriguez-Echeverria, Javier Luis Cánovas Izquierdo, Manuel Wimmer, and Jordi Cabot. Towards a Language Server Protocol Infrastructure for Graphical Modeling. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, MODELS '18, pages 370–380, New York, NY, USA, October 2018. Association for Computing Machinery.

[SHvH18] Christoph Schulze, Gregor Hoops, and Reinhard von Hanxleden. Automatic Layout and Label Management for UML Sequence Diagrams. report, Uni Kiel, August 2018.

[Sim01] Herbert Alexander Simon. *The sciences of the artificial*. MIT Press, Cambridge, Mass. [u.a.], 3. ed., 4. print.. edition, 2001.

[SMB23] Aylin Sarioğlu, Haydar Metin, and Dominik Bork. How Inclusive is Conceptual Modeling? A Systematic Review of Literature and Tools for Disability-aware Conceptual Modeling. In *Proceedings of the 42nd International Conference on Conceptual Modeling*. Springer, 2023.

[SSHK15] Martina Seidl, Marion Scholz, Christian Huemer, and Gerti Kappel. *UML @ Classroom: An Introduction to Object-Oriented Modeling*. Undergraduate Topics in Computer Science. Springer International Publishing, Cham, 2015 edition, 2015. ISSN: 1863-7310.

[Sta] StarUML. Sequence Diagram. https://docs.staruml.io/working-with-uml-diagrams/sequence-diagram. [Accessed: 9/26/2023].

[Vpj22] Vpjick. Top 30 Best UML Modeling Software [2022], February 2022. https://www.cybermedian.com/top-30-best-uml-modeling-software-2022/. [Accessed: 8/13/2023].

[Wik23] Wikipedia. List of Unified Modeling Language tools, July 2023. Page Version ID: 1163112205.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.