Institute of Software Engineering
Software Quality and Architecture

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelor's Thesis

IDE Support of Issue Management for Component-based Architectures

Tim Neumann

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr.-Ing. Steffen Becker
Supervisor:	Dr. rer. nat. Uwe Breitenbücher, Sandro Speth, M.Sc.
Commenced:	March 23, 2020
Completed:	November 11, 2020

Abstract

Context. Today software is often developed in architectures with many independent components, multiple projects, and even multiple independent teams. One of the key challenges in such an environment is managing software issues such as bugs or enhancement requests. One possible approach to deal with this problem was recently introduced by Sandro Speth in the form of multi-project coding issues and traceability links as well as his graphical notation for them.
Problem. The tool developed by Sandro Speth is web-based and, therefore, sub-optimal for developers, who do the majority of their work in an Integrated Development Environment (IDE). With it, a developer needs to switch between two different environments whenever he is working on an issue. Furthermore, advanced features such as opening the relevant software artifact in the editor of the IDE are not possible.
Objective. The objective of this thesis is to develop a concept for an issue management IDE extension specialized for this kind of environment and to evaluate if, how much, and how such an extension helps developers.
Method. To achieve this, a prototype extension is designed and implemented, which is then reviewed by experts. Finally, an expert survey is conducted to determine how useful the concept is.
Result. A solid concept for such an issue management extension has been developed. Most experts appreciate the idea. Additionally, many experts state that it solves various problems they face when working with issues in a multi-project, multi-team, component-based environment.
Conclusion. The developed IDE extension can practically help developers manage software issues in the challenging environments of multi-project, multi-team architectures. Additionally, the introduced concept can be used as a basis for extensions for other IDEs.

Kurzfassung

Kontext. Software wird heute oft in Architekturen mit vielen unabhängigen Komponenten, in mehreren Projekten und sogar mehreren unabhängigen Teams entwickelt. Eine der größten Herausforderungen in einer solchen Umgebung ist die Verwaltung von sogenannten Software Problemen (*Software Issues*) wie Bugs oder Erweiterungsanforderungen. Ein möglicher Ansatz zur Bewältigung dieses Problems wurde kürzlich von Sandro Speth in Form von *multi-project coding issues* (Multiprojekt-Kodierungsprobleme) und *traceability links* (Nachverfolgungsverküpfungen), zusammen mit seiner grafischen Modellierungssprache für diese, vorgestellt.
Problem. Das von Sandro Speth entwickelte Tool ist webbasiert und daher für Entwickler suboptimal, die den Großteil ihrer Arbeit in einer integrierten Entwicklungsumgebung (IDE) erledigen. Mit seinem Tool muss ein Entwickler immer zwischen zwei verschiedenen Umgebungen hin- und herwechseln, wenn er an einem Problem arbeitet. Darüber hinaus sind erweiterte Funktionen wie das Öffnen der relevanten Softwareartefakte im Editor der IDE nicht möglich.
Ziel. Das Ziel dieser Arbeit ist es, ein Konzept für eine IDE Erweiterung zur Problemverwaltung zu entwickeln, welche für diese Art von Umgebung spezialisiert ist, und zu evaluieren, ob, wie und in welchem Umfang eine solche Erweiterung den Entwicklern hilft.
Methode. Um dies zu erreichen, wird eine prototypische Erweiterung entworfen und implementiert, die dann von Experten untersucht wird. Schließlich wird eine Expertenbefragung durchgeführt, um festzustellen, wie nützlich das Konzept ist.
Ergebnis. Ein solides Konzept für eine solche Problemverwaltungserweiterung wurde entwickelt. Die meisten Experten schätzen die Idee. Zusätzlich geben viele Experten an, dass diese verschiedene Probleme löst, mit denen sie bei der täglichen Arbeit mit Software Problemen in einer komponentenbasierten Umgebung mit mehreren Projekten und Teams konfrontiert sind.
Fazit. Die entwickelte IDE Erweiterung kann Entwicklern praktisch helfen, Software Probleme in den herausfordernden Umgebungen von Multi-Projekt-Multi-Team-Architekturen zu bewältigen. Zusätzlich kann das vorgestellte Konzept als Grundlage für Erweiterungen für andere IDEs verwendet werden.

Contents

List of Figures

Glossary

Eclipse refers to a well known IDE but is also the name of the software foundation mainatining it. *see* IDE, 2

EMF Parsley is an EMF based UI framework for Eclipse plugins. *see* EMF, UI & Eclipse, 6

GraphQL is a query language devleoped by Facebook, used for querying data from servers. 3

Gropius is a tool for managing cross-component issues introduced by [SBB20]. ix, 1

Gropius EI is short for Gropius Eclipse Integration and the name of the tool developed for this thesis. *see* Eclipse & Gropius, 20

Java refers to the programming language originally devloped by Sun Microsystems. 6

Acronyms

API	Application Programming Interface. 4
DSL	Domain-specific Language. 6
EMF	Eclipse Modeling Framework. *see* Eclipse, 6
GQM	Goal Question Metric. vii, 29
IDE	Integrated Development Environment. iii, 1
IMS	Issue Management System. 3
MDSD	Model-Driven Software Development. 6
PDF	Portable Document Format. 32
PNG	Portable Network Graphics. 21
UI	User Interface. 6
UML	Unified Modeling Language. 6

1 Introduction

New software today is often based on architectures with many independently developed but interacting components. One popular example is the so-called microservice architecture. Such software consists of many small independent services, all fulfilling a small portion of the complete task. Often these services are implemented in many independent software projects by multiple teams. As with any software development process, it is advisable to have some kind of issue management solution as well as a way to document any major design decisions. Furthermore, documenting the interfaces between the services may even be more important then documenting the interfaces between components of a typical application.

However, as the documentation and issues are spread over multiple projects and each may affect more than one project there are several challenges to managing these documents and issues effectively. For example, changes to documentation may not be propagated to all teams. Another example is that issues found by one team may have the same root cause as issues found by another team. However, the root cause may even be caused by a piece of code of a third team.

One possible approach to deal with this problem is introduced by Sandro Speth [Spe19]. The proposed solution are so-called multi-project coding issues. These are issues concerning multiple projects and, therefore, are stored in each relevant project's issue management system. Additionally, traceability links between multiple issues and artifacts are proposed. As a concrete syntax, a graphical notation for these issues and links is introduced and implemented in a prototype framework. The frontend of this framework is a web-based application with an architecture graph at its core. This graph is used to view and edit the system architecture and issues displayed with the introduced notation. The development on this has since been continued by Speth et al. [SBB20]. The framework is now called Gropius. Therefore, the name *Gropius approach* is used to refer to that concept as a whole. The system is a supportive tool for software architects and project managers to keep an overview over the services and dependencies between those as well as the current issues and their influence on all components.

Even though developers can use this tool, it is rarely necessary for a developer to look at the whole system. It is more important that developers understand what parts of the system depend on the piece of source code they are working on and which other components may influence the behavior of it. Similarly, when working on an issue, a developer does not need to look at all issues of the complete system, but only at those, which may be a cause or an effect of the one being worked on. Finally, it should be as easy as possible for developers to create a new issue, think about and discuss how it may relate to other issues, and write down their findings. A web-app, which needs to be opened, the correct service selected, and the relevant file searched are all obstacles, which may impede a developer during this process.

This leads to the following research question: How can developers working in a multi-project environment with an Integrated Development Environment (IDE) be supported in managing issues that affect multiple of those projects.

The objective of this thesis is to develop, implement, and evaluate a concept for integrating the Gropius approach into the developer's IDE. This way, a developer's workflow is disrupted as little as possible, and most of these obstacles are removed. Developers can be shown all issues attached to a file or service as well as issues in components depending on it. Furthermore, they can create issues for the code they are currently looking at and can be aided in referencing any other relevant issues.

First, a concept for such an IDE extension is developed based on the results of a requirements engineering process. Based on this concept, an Eclipse plugin is designed and implemented. The resulting IDE extension is then subjected to a review by industry experts. Through that, valuable feedback is gained and the experts are introduced to the concept. Finally, an expert survey is conducted to evaluate if the concept is useful as well as in what way and by how much the extension can help developers in the field cope with the challenges when working with issues.

Most experts appreciate the developed concept and extension. Many state that it solves problems they are actually facing when working with issues in the mentioned environment. Most give valuable feedback on how to improve the plugin. Some even say they would use the result once a few more features are added.

The main contributions of this thesis are the concept for integrating the Gropius approach into an IDE, the implemented version of it in the form of an Eclipse extension, and the evaluation of both through the expert review and survey.

Thesis Structure

The thesis is structured as follows:

Chapter 2 – Foundations and Related Work: In this chapter, the work this thesis' is based on as well as other research in this area is presented.

Chapter 3 – Concept: This chapter introduces the developed concept.

Chapter 4 – Plugin Design and Implementation: Here the design and implementation of the IDE extension is described.

Chapter 5 – Evaluation: The realization and results of the expert review and survey are described and discussed.

Chapter 6 – Conclusion: In this chapter, the thesis is summarized as well as possible future work highlighted.

2 Foundations and Related Work

This chapter outlines other work relevant to this thesis. First, the foundations are described in Section 2.1. Then Section 2.2 introduces and discusses other work in this research area, discovered using the process described in Section 2.2.1.

2.1 Foundations

This section describes the foundations this work is based on, which may also help understanding this thesis. First, in Section 2.1.1 the query language GraphQL is introduced. After that, Section 2.1.2 covers the idea of software issues and issue management systems in general. Then, in Section 2.1.3, a concept for issue management in environments where multiple software components, which are developed by independent teams, is described. Section 2.1.4 introduces the concept of IDEs and how they can be extended with a focus on Eclipse as the prototype developed for this thesis is an Eclipse plugin. Finally, Section 2.1.5 covers two frameworks used in the implementation of that plugin.

2.1.1 GraphQL

GraphQL is a query language developed by Facebook, where the client needs to specify the parts of data to return [Fac18]. Furthermore, changed data cannot just be sent to the server to be updated, but the changes must be recorded and split into so-called mutations, which can then be sent to the server.

2.1.2 Software Issues and Issue Management Systems

Software issues are a way to communicate a need for action in the software engineering process. Traditionally, a distinction was made between a bug, which needs to be debugged or fixed, a task someone needs to do, and an enhancement being requested. Issues can represent any of these things as well as anything a developer needs them to represent [Atl20; Git20].

Issue Management Systems (IMS), also called Issue Tracking Systems, are programs, which maintain these issues and “help organizations manage issue reporting, assignment, tracking, resolution, and archiving” [BVGW10]. Other common names include bug tracker, defect tracker, and ticket system. They can bring various benefits like improved software quality, request accountability, and an increase in productivity [Jan09].

Software issues typically consist of at least a title, a detailed description or body text, and a state, which typically has at least the possible values open and closed. Additionally, each IMS supports various additional fields like the type (bug, enhancement, task, and others), labels to further classify the issue, relevant parts of the source code, the creation date, a due date, the issue creator, the developer(s) assigned to work on it, or many more. However, the amount of supported fields varies significantly between different programs.

Common IMS include Atlassian's Jira[1], Bugzilla[2] as well as the issue management support integrated into Redmine[3], GitHub[4] and GitLab[5]. Most provide various features in addition to just storing these issues. Those features include linking related issues, grouping issues into so-called Milestones, organizing issues on project boards, similar to the Kanban Boards described by Thomas Epping [Epp11], advanced time tracking, and more. An in-depth comparison of a few well known IMS is provided by Janák, Jiří [Jan09].

2.1.3 Issue Management in Multi-Team, Multi-Component Environments (Gropius)

Managing documentation and issues for multi-project, multi-team scenarios efficiently is a major problem for the software development process [MNH15]. This assertion is supported by some posts in forums of Jira and Redmine asking for solutions to this problem[6][7][8]. Another strong argument supporting it are the results of an expert survey conducted by Sandro Speth [Spe19], which indicate that industry experts also see this as a problem.

One possible approach to deal with this problem is introduced in [Spe19]. The proposed solution are so-called multi-project coding issues. These are issues concerning multiple projects and, therefore, are stored in each relevant project's issue management system. Additionally, traceability links between multiple issues and artifacts are proposed. For example, one issue about a crash could link to another issue describing the Exception causing the crash, which in turn links to the piece of code causing it and the part of the architecture specification involved. As a concrete syntax, a graphical notation for these issues and links is introduced and implemented in a prototype framework. The front-end of this framework is a web-based application with an architecture graph at its core. This graph is used to view and edit the system architecture and issues displayed with the introduced notation.

Speth et al. [SBB20] further refine this concept and introduces a tool named Gropius, which is based on the prototype mentioned above. Among other things, the paper presents the architecture of the Gropius system. One relevant fact for this thesis is that the Gropius back-end provides a GraphQL Application Programming Interface (API) for the front-end, which can also be used by

[1] https://www.atlassian.com/software/jira
[2] https://www.bugzilla.org/
[3] https://www.redmine.org/
[4] https://github.com/
[5] https://gitlab.com/
[6] https://www.redmine.org/boards/1/topics/21939
[7] https://community.atlassian.com/t5/Jira-questions/How-do-others-work-with-issues-affecting-multiple-projects/qaq-p/399950
[8] https://community.atlassian.com/t5/Jira-Software-questions/Share-one-issue-quot-ticket-quot-across-multiple-projects-and/qaq-p/407534

the tool of this thesis. The Gropius system does not just store the issues in its own database but works with one or more traditional IMS to retrieve and store the issues. Therefore, some developers of a team can use the system without causing problems for other colleagues, not using it.

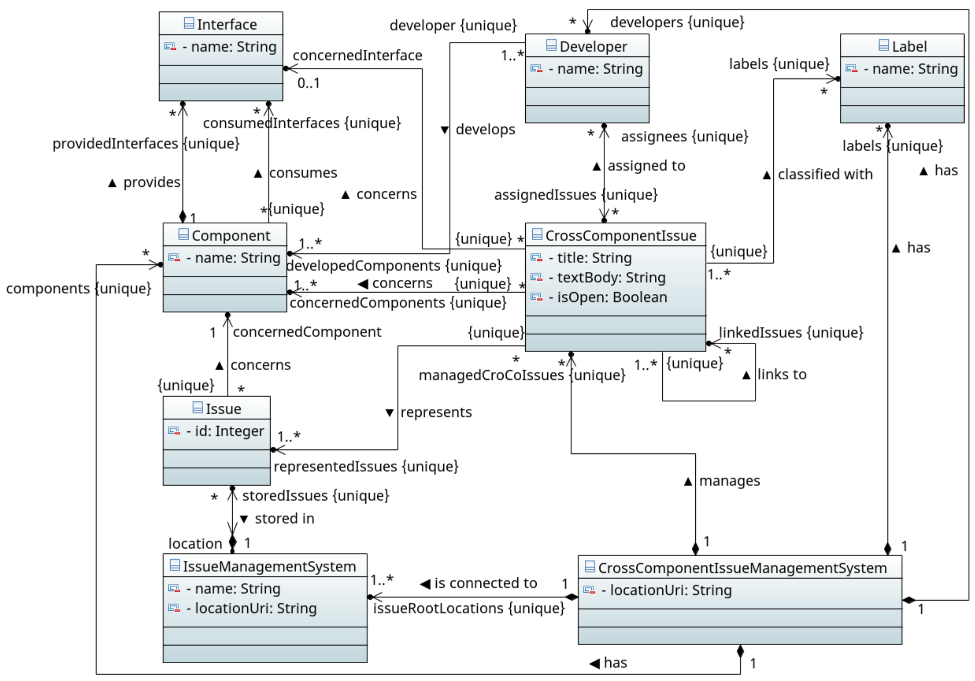


Figure 2.1: Domain meta Model of the Gropius system

The same figure can be found enlarged in Appendix A

It also introduces a domain meta-model for the Gropius system, as depicted in Figure 2.1 [SBB20]. At the center is the Cross-Component-Issue, which represents multiple normal issues. These Cross-Component-Issues are stored in the Cross-Component-Issue-Management-System, for example, an instance of the Gropius system. That is connected to multiple traditional IMS, which hold the regular issues. In addition to some simple attributes like the title, text body, and the flag for whether the issue is open, a Cross-Component-Issue also has references to the assigned developers, its labels, and the components or interfaces it occurs in.

2.1.4 Extending Integrated Development Environments

An IDE is a software tool for programming, which typically has a rich text editor at its core and contains various other tools for use during software development. The concept of a dedicated software IDE can be traced back to the Maestro I system, first called “Programm-Entwicklungs-

Terminal-System" (program development terminal system), short PET, developed by Softlab Munich reported about in [Com75]. Well known IDEs today include Visual Studio[9], Eclipse[10], NetBeans[11], IntelliJ IDEA[12], and Visual Studio Code[13] .

Various IDEs follow the general trend to allow extending a piece of software with the help of plugins [CCPP08]. This enables developers to add custom features or integrated support for previously unsupported external tools. The concept of integrating many tools into one environment was already investigated in 1990 by Anthony I. Wasserman [Was90].

One of those is Eclipse, originally being developed by IBM using Java as proprietary software. It was open-sourced in 2001 and is maintained by the Eclipse Foundation[14] today [Bur05]. Originally, it was intended as a Java IDE but with the help of plugins support for many programming languages can be added today. According to various online rankings[15][16], Eclipse is one of the most popular IDEs at the time of writing.

Eclipse is designed to be extended by plugins. This is regularly done by the industry as well as by researchers because it is a simple way to create a large and powerful tool containing custom features without requiring the effort to build a complete tool from scratch. Recent papers using Eclipse as the base of their tool are, for example, [SL19] or [HQM19]. The process of creating an Eclipse plugin is explained in great detail by Clayberg et al. [CR06].

2.1.5 The Eclipse Modeling Framework and EMF Parsley

One alternative for manually implementing every part of a program is Model-Driven Software Development (MDSD). With this approach, most of the software is modeled using some modeling language or tool and then generated [VSB+13].

The Eclipse Modeling Framework (EMF) is a framework for MDSD optimized for use with Eclipse. At its core is the Ecore meta-model, with which models are represented [SBMP08]. Such an Ecore model can, for example, be obtained by converting an existing Unified Modeling Language (UML) model. From that, the complete source code for the data structure represented by the model can be generated. The generation of other components, such as editors, is also supported but not used in this thesis.

EMF Parsley is a framework on top of EMF for creating User Interfaces (UIs). It allows developers to build an Eclipse UI based on a data model generated by EMF with very little setup [Bet14]. Developers can use existing UI elements provided by EMF Parsley and customize them to their needs using injected aspects [Bet14]. One core feature is the integrated Domain-specific Language (DSL) allowing developers to specify the desired UI in a simple language [Bet14].

[9] https://visualstudio.microsoft.com
[10] https://www.eclipse.org/ide/
[11] https://netbeans.org
[12] https://www.jetbrains.com/idea/
[13] https://code.visualstudio.com/
[14] https://www.eclipse.org/org
[15] https://pypl.github.io/IDE.html
[16] https://www.positronx.io/top-10-best-ide-for-software-development/

2.2 Related Work

First, the methodology used for finding the related work is presented in Section 2.2.1. Then the related academic work found is introduced in Section 2.2.2. Finally, Section 2.2.3 covers existing issue management plugins for eclipse.

2.2.1 Literature Research Methodology

The following procedure was used for finding related work: First, an initial search was performed using the search terms below and Google Scholar[17] as the search engine. Of each query, the top ten results were taken and examined. First, results were rejected based on their title and the short excerpt Google Scholar includes in the search results page, then based on the abstract. All remaining results were examined in depth to check whether they can be considered related work. Finally, the same process was repeated with all references to English papers of any work found, which was classified as related work until no more such papers were identified (Snowballing).

The build the search terms, all possible combinations of the following two parts were used. For part number one the following terms were used:

- IDE
- Integrated development environment
- Eclipse

For the second part these terms were used:

- issue management
- issues
- issue trackers
- bug tracker
- defect tracker

This makes a total of 15 search terms resulting in 150 possible papers and books, only considering the first ten results of each term. After sorting by title and excerpt, a total of nine unique results remained, three after reading the abstract, and two after further analysis. Of these two papers, the first had 12 references, which were all rejected based on the title and the second had 18 references in the relevant section, of which also none could be identified as relevant for this work.

Additionally, the eclipse marketplace[18] was searched using the terms “issue management”, “defect”, and “defects”. All results were searched for appropriate plugins. The term “bug” could not be used to search the marketplace because over 200 plugins were found, and on the first few pages, most results were not relevant.

[17] `https://scholar.google.com`

[18] `https://marketplace.eclipse.org`

2.2.2 IDE Support for Issue Management in Academia

Only two papers about the integration of issue management into an IDE could be found. This indicates that not a lot of research has been done in this area.

The first of the two papers is [IUH09], which includes a short section on integrating their custom framework called “Linked Data Driven Software Development (LD2SD)” into the Eclipse IDE. LD2SD allows searching in various software development artifacts like documentation or issues. However, the only integration into Eclipse presented in the work is the ability to perform a query based on parts of the source code (like a class or method) within Eclipse and the result would be shown in the integrated web browser provided by Eclipse [IUH09]. They show that such a query can be started by a custom element in the context menu of Java class files in the Eclipse package explorer.

Similarly to theirs, this thesis aims to integrate an existing framework into the Eclipse IDE with the help of a plugin. However, in contrast to them, this work only focuses on issues but aims to provide much more extensive support to work with those, like adding custom UI elements to view and edit them.

The second related work found is [Jan09]. It first gives an overview of existing issue management plugins for IDEs and then describes the process for developing an “Atlassian NetBeans Connector”.

In the overview, the author differentiates between two kinds of plugins realizing the integration of issue management into IDEs. According to the author, the first kind is a “Universal plugin with bridge for connecting the issue tracking system”, which allows the easy integration of multiple IMS by providing an interface, which can be implemented by so-called bridges, each attaching a single IMS to the universal plugin. The core of this universal plugin is then responsible for working with the IDE to show the retrieved issues to the user. The author states that this kind of issue management plugin generally does not support advanced features of the individual IMS. The second kind is the “Single issue tracking system plugin” [Jan09], which typically supports most features of a single IMS.

For the first kind, he describes Mylyn for Eclipse, which is discussed in detail in Section 2.2.3 as well as Cube°n for Netbeans. Cube°n[19] is a task-focused UI for NetBeans similar to Mylyn, which was originally started at Google’s “Summer of Code“ in 2008 [Jan09]. It has since been renamed to Task Focused NetBeans[20] and is being maintained by the NetBeans team and community.

For the second kind, the paper introduces two plugins for the IMS CodeBeamer ALM by Intland Software[21], one for Eclipse and one for NetBeans, as well as the “Atlassian IDE Connector” for IntelliJ IDEA and Eclipse. According to the author, the CodeBeamer ALM is a good issue management plugin supporting all CodeBeamer ALM features with a simple and intuitive UI. Both Atlassian Connectors allow the use of various Atlassian products, including Jira, in the IDE [Jan09]. The author states that the connector for IntelliJ IDEA is a lot more advanced than the one for Eclipse, which is based on Mylyn. The Atlassian Connector for Eclipse is also covered in Section 2.2.3.

[19] https://code.google.com/archive/p/cubeon/

[20] http://wiki.netbeans.org/TaskFocusedNetBeans

[21] https://codebeamer.com/

Then the work describes the concept, design, and implementation of a new Atlassian connector for NetBeans, with the Jira connector being the primary goal. According to the author, it is primarily inspired by the above mentioned Atlassian IntelliJ IDEA connector. The created Jira integration allows users to view a filtered list of issues, a specific issue as well as their comments, create a new issue or comment, log work on an issue, view stack traces from an issue, and click through to the relevant source file, assign an issue to someone and perform workflow actions (similar to changing the state) on a selected issue.

That work provides a much more advanced integration than the previous one, with a similar amount of features to the one presented in this thesis. On the other hand, the plugin presented in it only works with the Jira IMS in contrast to the one of this thesis, which integrates into the Gropius framework, which theoretically can work with any IMS. Additionally, theirs is for NetBeans, while ours is for Eclipse.

However, the essential difference between this thesis and the two covered related works is the fact that, in contrast to the other two, the tool presented in this thesis has support for the features of Gropius, which are intended to help with the issue management in component-based, multi-team environments.

2.2.3 Issue Management Plugins for Eclipse

Compared to academia, the industry has put a little more work into this area. There exist a few issue management plugins for various IDEs [Jan09], but in this section, the focus is on issue management plugins for Eclipse. In total, five issue-management plugins for Eclipse could be found. Four, which are specific to one IMS and one universal plugin using bridges to connect to many IMS.

The first is “Teamscale Integration for Eclipse”[22], which allows users to browse the defects found by a Teamscale software-quality analysis server. This server is not a typical IMS, but rather a software analyzing the source code for potential issues.

The second is called “CollabNet Desktop”[23] and integrates with the application lifecycle management platforms of CollabNet, which include issue management and can therefore be called IMS in the context of this thesis.

The next is “JiraBuddy - Eclipse Plugin for JIRA”[24] is a plugin adding various enhancements when working with Atlassian Jira and Eclipse, but does not provide the capability to create or update issues from within Eclipse. According to their website[25], the only feature is to display the issue when hovering over the corresponding issue identifier in the Eclipse editor.

The last plugin of this kind is “Atlassian Connector for Eclipse”[26]. It not only provides integration into Atlassian Jira, which is relevant for this thesis but also other Atlassian products like the Bamboo continuous integration and deployment server. It is actually built on top of Mylyn, which is explained below.

[22]https://marketplace.eclipse.org/content/teamscale-integration-eclipse

[23]https://marketplace.eclipse.org/content/collabnet-desktop-eclipse-edition

[24]https://marketplace.eclipse.org/content/jirabuddy-eclipse-plugin-jira

[25]http://home.jirabuddy.com/

[26]https://marketplace.eclipse.org/content/atlassian-connector-eclipse

All of the above only work with one specific IMS, whereas the tool proposed by this thesis can work with any IMS supported by Gropius. Additionally, the last two are not maintained anymore and, therefore, have no support for current Eclipse versions.

The fifth and most important plugin is Mylyn[27] together with its many connectors for various IMS. It is one of the oldest issue management plugins for any IDE [Jan09]. At its core is a list of tasks, which can contain local tasks stored on the computer and tasks provided by one of the many connectors. One important concept is the task-focused interface, which means Mylyn tries to highlight the parts of the Eclipse UI relevant to the current task and makes the less relevant parts less prominent. That mainly applies to the Eclipse package explorer, where different resources and files will be drawn with different shades of gray based on their relevance for the current task. Data about the relevancy of parts for the current task is extracted from the user's behavior. However, in contrast to the solution provided in this thesis, Mylyn can only associate issues or tasks (as they are called in Mylyn) to resources as specific as source code files and not to lines within them.

Similar to the academic work, all these plugins lack support for all or some features necessary in component-based, multi-team environments, like one issue being saved in multiple different IMS of multiple teams or the ability to quickly navigate between related issues.

[27] `https://marketplace.eclipse.org/content/mylyn`

3 Concept

The objective of this work is to evaluate the usefulness of an IDE extension for the Gropius system [SBB20]. This chapter describes the approach taken in this thesis for archiving that goal. At the heart of the concept is a plan to develop an issue management extension for one IDE, which integrates into the Gropius framework. This tool is then used in the evaluation by presenting to experts in the field for a review.

The first step was to gather requirements for that prototype. This requirements engineering process, as well as its results, are described in Section 3.1. Visualizing details about an issue in the corresponding icon is a crucial part of multiple requirements. Therefore, the next step was to determine which properties to visualize and how. The resulting concept for the icons is explained in Section 3.2. Finally, as described in Section 3.3, a rough concept for the extension was made before continuing with the design and implementation.

3.1 Analysis and Requirements Engineering

The goal of requirements engineering is to gather, document, validate and manage requirements for a computer-based system [SS97]. To gather requirements for the extension, a process with six steps, as shown in Figure 3.1, was used. This process is split into two stages, each with the three activities of requirements elicitation, specification, and validation.

As the first step of the first stage, a preliminary requirements elicitation was performed. For this, internal brainstorming and discussion with the thesis' supervisors were used. Next, the resulting requirements were incorporated into the proposal for this thesis, which serves as the first requirements specification. After this, the primary requirements validation was performed. The positive feedback for that proposal paper was one of the signs that the requirements are valid. Additionally, the discussion and feedback after the thesis proposal presentation was also used to validate the requirements.

Then, the second stage of requirements elicitation was performed using the requirements from the first elicitation, the results from the validation as well as the results of a stakeholder interview. For this interview, the general concepts of Gropius and the idea of this thesis was sent to representatives of each group of stakeholders. They were then asked for their requirements with the help of two questions. The first question asked for features expected from an issue management plugin in general, the second for those which the representatives would expect from such a plugin, which integrates into Gropius.

Since the prototype is a new tool for a system under development, only three groups of stakeholders could be identified: First, the potential users, in this case, developers, working with component-based systems, such as microservices. Various members of the department for software quality

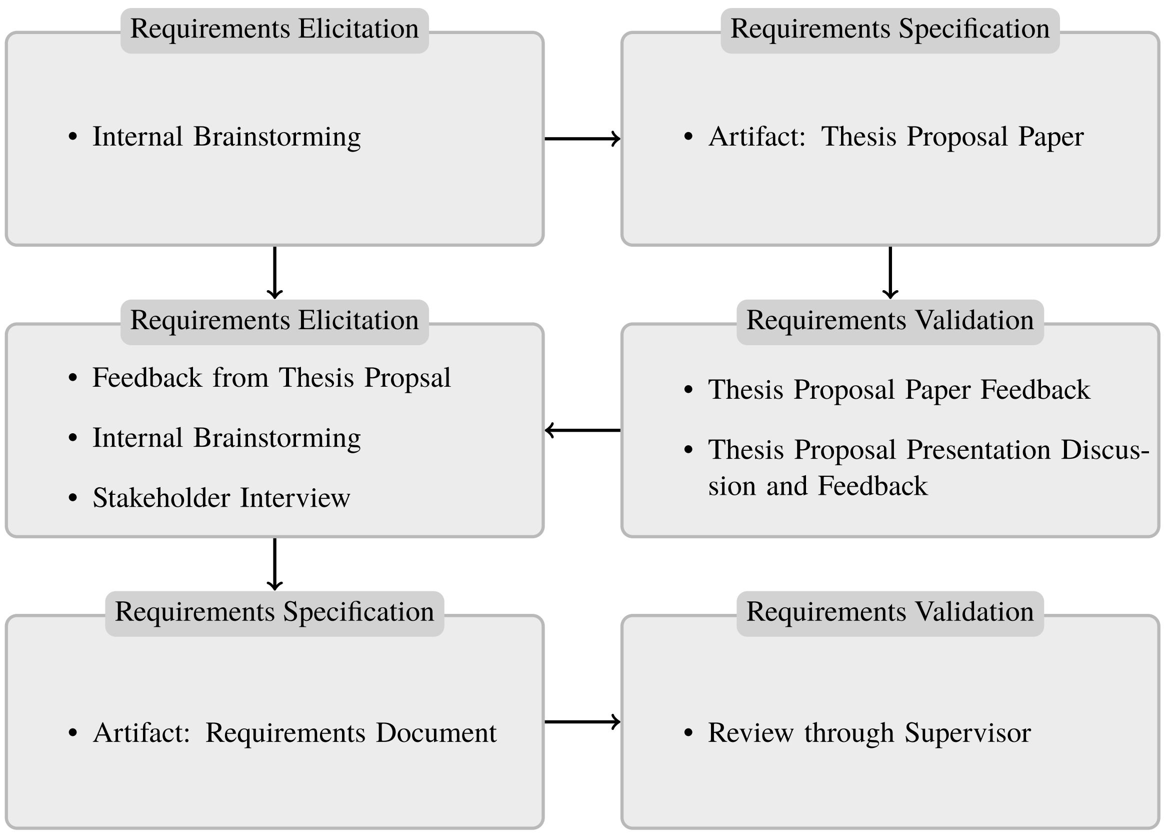

Requirements Elicitation
• Internal Brainstorming
Requirements Specification
• Artifact: Thesis Proposal Paper
Requirements Elicitation
• Feedback from Thesis Propsal
• Internal Brainstorming
• Stakeholder Interview
Requirements Validation
• Thesis Proposal Paper Feedback
• Thesis Proposal Presentation Discussion and Feedback
Requirements Specification
• Artifact: Requirements Document
Requirements Validation
• Review through Supervisor

Figure 3.1: Requirements Engineering Process

and architecture were contacted as representatives of this group. Second, the thesis' supervisors to ensure the scientific scope of the work. Third, Sandro Speth, who is also one of the supervisors, as the author of the Gropius system.

Next, the gathered requirements were formally specified as user stories in the requirements document. Finally, this document was validated by a review from the thesis' supervisors.

Gathered Requirements

The result of this requirements engineering process is a list of functional requirements in the form of user stories. These requirements are listed below, split into those expected from any issue management plugin and those expected from a plugin integrated into Gropius.

First, the requirements expected from any issue management plugin:
As a developer, I want to ...

1. ... have a list of all open issues.
2. ... have a list of all issues assigned to me.
3. ... have a searchable list of issues.
4. ... have a list of issues that can be filtered by tags or labels.

5. ... have a list of issues that can be filtered to only show issues relevant to my currently opened project.
6. ... have a list of issues that can be filtered to only show issues relevant to my currently opened source file.
7. ... quickly see the type of an issue in this list.
8. ... be able to see all details for a specific issue.
9. ... be able to jump to a related issue from my currently viewed issue.
10. ... be able to quickly open a relevant source file at the relevant line for an issue, even if it is in another eclipse project (as long as that project is in my workspace).
11. ... be shown which lines of a source file are relevant for an issue. (For example, through markers at the side of the editor)
12. ... be able to edit every property of an issue that I have write access to.
13. ... be able to create a new issue.
14. ... be able to create a new issue for some lines of code without manually entering the source file and lines in a dialog.

These are the requirements expected from such a plugin integrated into Gropius:
As a developer, I want ...

15. ... the system architecture graph in my web browser to focus on the component I've selected in eclipse.
16. ... the system architecture graph in my web browser to highlight the issue I've selected in eclipse.
17. ... to be able to open details about an issue in the eclipse view through the architecture graph in my web browser.
18. ... to be able to open a relevant source code line of an issue in eclipse from the architecture graph in my web browser.
19. ... to be able to have the possibility to set the focus of the architecture graph and eclipse editor as well as the issue view of my colleagues, for example, in meetings.

3.2 Concept for the Issue Icons

For requirements 7 and 11, it is useful to have icons for each issue. Especially for requirement 7 the icons can not all be the same but need to carry some information. Therefore, a concept was made about which properties should be visualized in these icons.

One possible property is the type of issue. As described in Section 2.1.2 an issue can represent various concepts. The type of issue can be used to roughly specify which of these concepts is represented by an issue. Common values are bug and enhancement.

Another interesting concept that could be visualized in the icons is the idea that one issue can be the cause of another. For example, a Null-Pointer in one component could cause an illegal response in an API, causing a crash in another component. Then the Null-Pointer is the root cause. The illegal response is a symptom of that but also the cause of the crash. The crash is just a symptom of the illegal response and can probably not be fixed entirely without fixing the other two.

Other interesting properties include to whom the issue is assigned and the state of the issue. It could be interesting to recognize the issues assigned to yourself immediately. With the state of the issue, a developer can directly see if the issue still exists and if work is required.

As it was not clear how much information could be visualized in one small icon, the properties that should be visualized were ordered by their priority. That way it was easier to make decisions about what information to include in the icon and how prominent to visualize it later during the creation of these icons. The following properties were chosen to be visualized with decreasing priority.

- Type of Issue

  Enhancement vs. Bug
- Does this issue cause problems in other projects/components?
- Is this issue just a symptom of another issue elsewhere or is this the root cause?
- Is the issue assigned to the viewing developer?
- Is the issue new or work in progress or done?

Finally, another aspect that could be visualized is the visibility of the issue as some IMS can have issues that are only visible internally. However, it was decided against it for now, as it did not seem that important of a property, and the Gropius system currently does not support it.

3.3 Concept for the IDE Extension

Before starting with the detailed design and implementation of the prototype tool, a rough concept needed to be created. Most parts of this were already done during the thesis' proposal process, and a version of this concept was included in the proposal paper. Therefore, the paper's feedback and the presentation in front of the department confirmed that the general concept is reasonable.

According to the concept, the plugin consists of two main UI elements: The issue list and the issue details element. These are complemented by various dialogs and other features described below. The plugin uses the API of the Gropius system to retrieve the required data and perform any modifications on it. Furthermore, the plugin is split into multiple components, of which only one is specific to the supported IDE, while the others are reusable for other IDEs with minimal changes. That way, the amount of work to port it for another IDE is minimized, as long as it supports Java-based plugins.

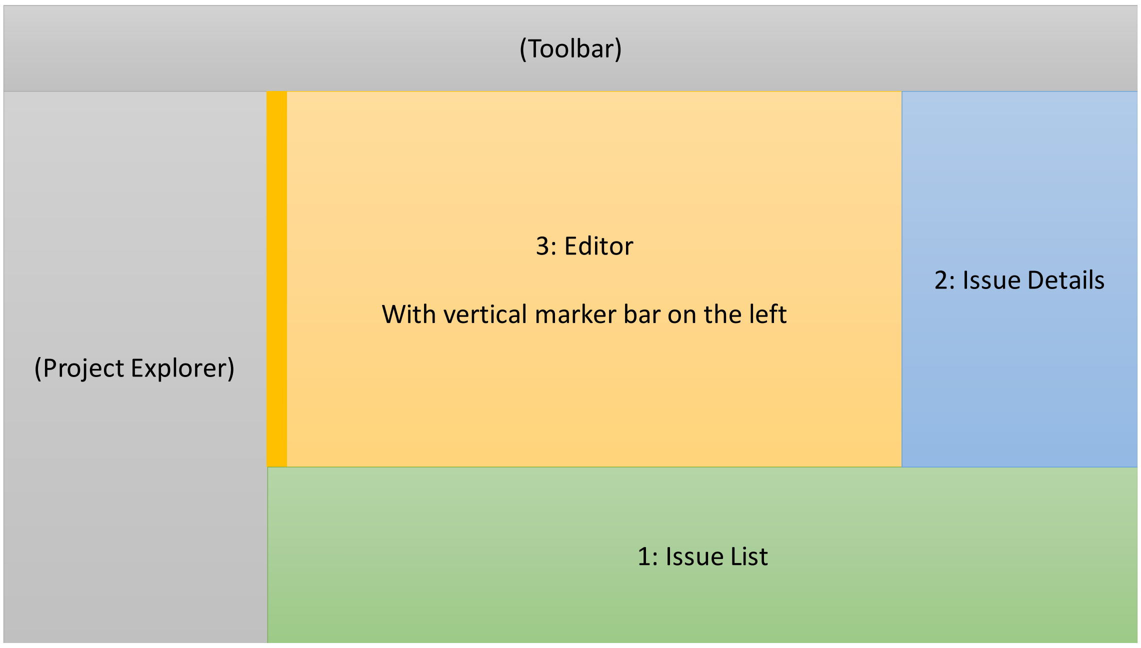
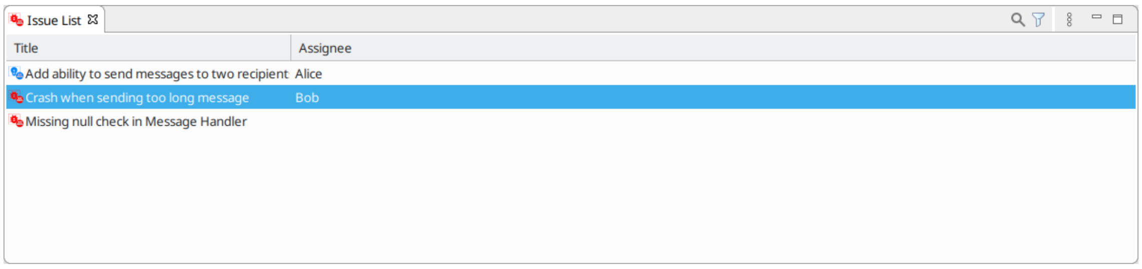


Figure 3.2: Mock-up of a possible IDE window

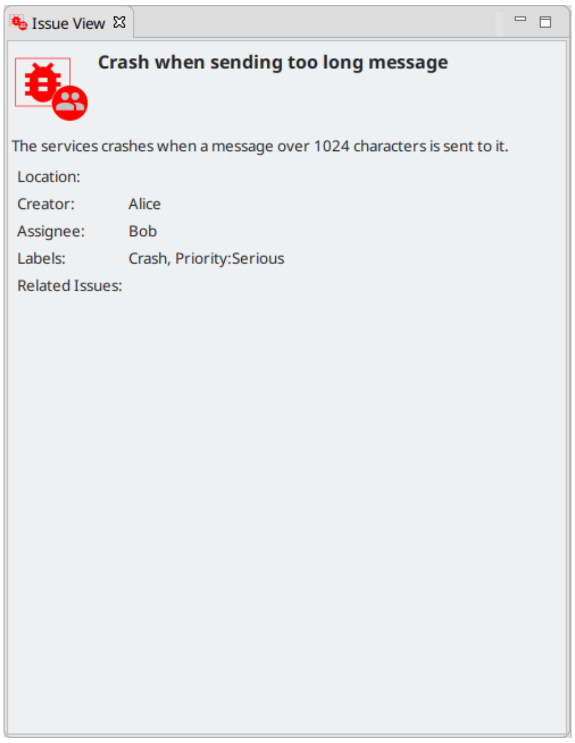
Figure 3.2 shows one possible layout for the main IDE window while the extension is being used. The issue list can be seen in the lower part of the image, while the issue details UI element is on the right. In the center is the IDE's editor with the vertical bar for markers on the left edge of it. However, the aim is, to allow the user to move all of these parts to wherever in the IDE is best for his personal process.


Issue List

Title	Assignee
Add ability to send messages to two recipient	Alice
Crash when sending too long message	Bob
Missing null check in Message Handler	

Figure 3.3: Mock-up for Issue List (1)

The issue list is a table of issues with columns for the various properties of those, similar to the one shown in the mock-up in Figure 3.3. In the first column, the icon for the issue is shown followed by the title. This table allows filtering by various attributes of the issues. To satisfy requirements 1 to 6 it supports at least filtering by issue state, assignee, text contained in the title or description, labels as well as whether the issue is relevant to the open IDE project or the open file. Furthermore, it is possible to sort the table based on any column. Finally, the visible columns can also be configured by the user.



Crash when sending too long message

The services crashes when a message over 1024 characters is sent to it.

Location:	
Creator:	Alice
Assignee:	Bob
Labels:	Crash, Priority:Serious
Related Issues:	

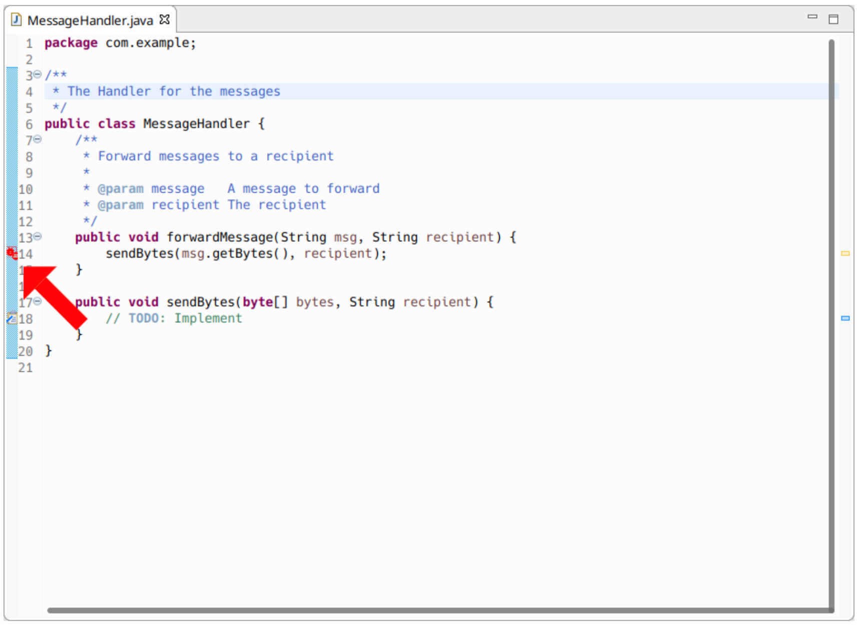
Figure 3.4: Mock-up for Issue Details (2)

The issue details UI element shows all attributes for the issue selected in the issue list. A mock-up of it can be seen in Figure 3.4. At the top of the form, the icon of the issue is displayed. The related issues are links, which, when clicked, change the issue list selection to the corresponding issue. Furthermore, the displayed locations are also links, which open the correct resource in the IDE's editor. Additionally, this element supports editing the issue, either by opening a dialog or by allowing to edit the values in the element itself like a form. In both cases, additional dialogs are used to allow editing of the more complex attributes.

Moreover, the issues are displayed as markers in the IDE's editor, whenever a file with issues is open in the editor. This can be seen in the mock-up in Figure 3.5. For these the icons from Section 3.2 are used, if the IDE supports it.

Additionally, there are two ways to create a new issue. First, there is a button somewhere, which creates a new empty issue, optionally opening a dialog to enter details for the new issue. Second, it is possible to mark some lines in the editor and create a new issue for them. In this case, the new issue already has the correct location selected.

Finally, the tool is connected to the existing web UI with the help of a messaging component. That way it is possible to enable synchronization between the plugin and the web user interface, such that selecting an issue in one will cause the other to also jump to the same issue and select it accordingly. Each user has a separate topic in the messaging component, preventing undesirable interaction between multiple users. In addition to this, special topics, for example, for meetings are a possibility, such that one developer can select an issue and the same is selected for all other participants of the



```
package com.example;

/**
 * The Handler for the messages
 */
public class MessageHandler {
    /**
     * Forward messages to a recipient
     *
     * @param message   A message to forward
     * @param recipient The recipient
     */
    public void forwardMessage(String msg, String recipient) {
        sendBytes(msg.getBytes(), recipient);
    }

    public void sendBytes(byte[] bytes, String recipient) {
        // TODO: Implement
    }
}
```

Figure 3.5: Mock-up of Issue in the Marker Column (3)

meeting. Figure 3.6 shows the basic architecture planned for this feature. The plugin, as well as the web user interface, communicate with the back-end through the GraphQL API and with each other using a messaging component.

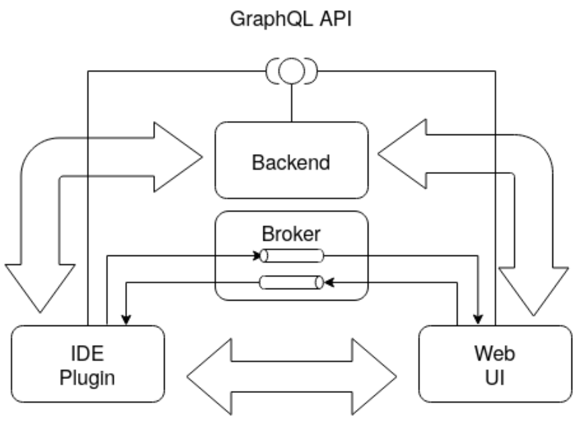


Figure 3.6: Messaging Concept for Gropius Frontend Integration

4 Plugin Design and Implementation

This chapter describes how the concept was implemented and presents the design of the plugin. First, a decision had to be made, for what IDE the extension would be implemented. This decision, as well as some reasoning for it, can be found in Section 4.1. Then, the icons conceived in Section 3.2 needed to be created, which is described in Section 4.2. Section 4.3 presents the architecture for the prototype. Finally, some implementation details of the plugin, as well as interesting parts of the implementation process, are described and the final prototype presented in Section 4.4.

4.1 Choosing the IDE

The concept presented in Chapter 3 is independent of any IDE. However, the extension can only be implemented for one IDE during the course of this thesis. Therefore, a decision had to be made, which IDE to choose. To support this decision, the popularity, the support for creating plugins, and my prior experience were considered.

To determine the popularity, a current online ranking [1], which is based on Google statistics, was consulted. According to it, at the time of writing, Visual Studio was the most popular with a 25% share. The second best was Eclipse with 16%, followed by Android Studio and Visual Studio Code (completely separate from Visual Studio) with 11% and 9%. The last two considered were pyCharm on rank 5 with 8% and IntelliJ IDEA on rank 6 with 6%. We decided against Android Studio and pyCharm, as they only support the development of Android Apps and Python programs, respectively. All the other IDEs support multiple languages allowing the extension to be used in a more diverse set of scenarios.

Of the remaining four candidates, the ability and complexity of creating extensions were researched. The IDE with the easiest process to create plugins is Visual Studio Code[2]. The second best is Eclipse, also having a lot of material helping with creating plugins for it, such as the book by Clayberg et al. [CR06]. Plugins are a core concept of both and are used to provide almost all features available in these IDEs. Next, IntelliJ IDEA also has a fairly simple process for creating plugins[3]. Finally, Visual Studio also supports creating plugins, but the process is more complex than with the previous three[4]. Additionally, Visual Studio Extensions can only be developed on Windows.

[1]https://pypl.github.io/IDE.html
[2]https://code.visualstudio.com/api/get-started/your-first-extension
[3]https://jetbrains.org/intellij/sdk/docs/basics/getting_started.html
[4]https://docs.microsoft.com/en-us/visualstudio/extensibility/starting-to-develop-visual-studio-extensions?view=vs-2019

Eclipse scored second in both of the above criteria and the winners of both categories are worse in the other. Additionally, I already had experience with developing plugins for it. Therefore, the Eclipse IDE was chosen as the one for which the extension will be implemented.

As the prototype plugin integrates the Eclipse system into the Eclipse IDE, it has been named *Gropius Eclipse Integration* (Gropius EI).

4.2 Icon Creation

To create the icons described in Section 3.2 the plan was made to combine multiple individual icons, one for each property, into many finished icons. The center of each icon indicates the type of issue, as that is the most relevant property. The main symbol's color shows the state of the issue because it is the least essential property. However, to not discriminate people with partial or full colorblindness, a small black icon is also overlaid over the main icon. The remaining three properties are visualized as annotations, small symbols displayed in the corners of the icon. All of these parts are drawn as vector graphics so that the final icons can easily be scaled to any size. One challenge while drawing was that eclipse uses 16x16 pixel icons. Therefore, all finished icons need to convey their information even if they are just 16x16 pixels. This significantly reduces the possible detail in the parts.

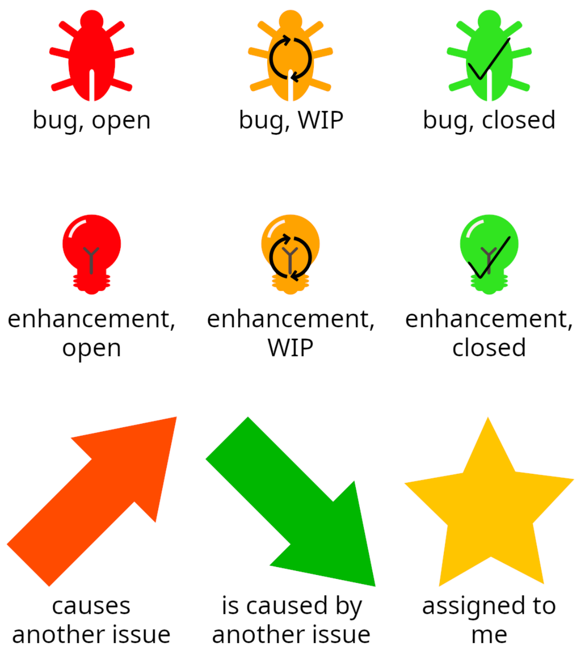


Figure 4.1: Issue Icon Parts

Figure 4.1 shows all the parts described above. The first row contains the bugs in all three possible states, open, work-in-progress, and closed. In the second row are the enhancements with the same three states. The third row contains the annotations for the other three properties. The first annotation is used when this issue is the cause of another issue in some other component. In the finished icon, it is located in the top right corner. The second symbol is the annotation for when the issue is caused by another issue and is, therefore, just a symptom of that issue. On the final icons, this is shown in the top left. The third symbol represents the fact that the issue is assigned to the current developer. It is added in the bottom left of the final icon.

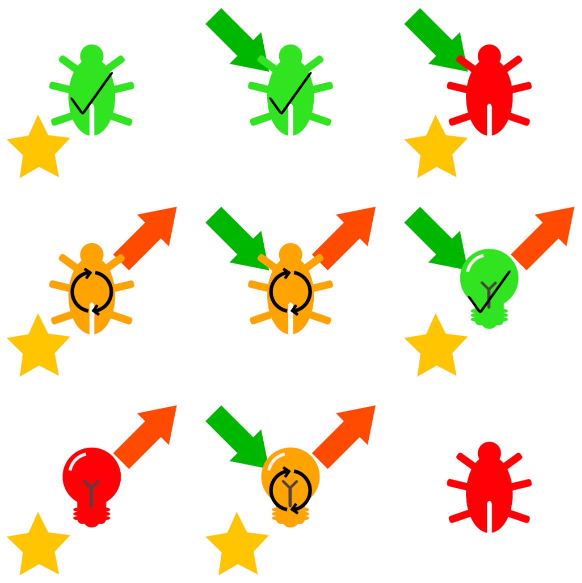


Figure 4.2: Issue Icon Examples

In Figure 4.2 some examples of the final icons can be seen. In total, there are 48 different combinations of the above parts. All of those need to be created and then converted into a pixel-based image format in multiple sizes so they can be used. This would be a very tedious manual task for one person. Especially, as the icons were changed multiple times during the development process. Therefore, a small script was created, which does all that automatically and results in a folder with all combinations as Portable Network Graphics (PNG) files in the specified sizes.

4.3 Eclipse Plugin Architecture

One possible approach that was investigated is to implement the plugin as an extension to Mylyn. As described in Section 2.2.3, Mylyn is an Eclipse plugin for managing tasks and can, therefore, also be used to manage plugins. However, many of the required features from Section 3.1 could, to our knowledge, not be implemented in a Mylyn extension without contributing some significant changes to Mylyn itself. Any changes proposed for Mylyn would need to be discussed with the community and justified to the maintainers causing, in the best case, a major delay for the implementation of the extension. In the worst case, some changes may be rejected entirely, preventing the implementation of some of the required features. Therefore, it was decided to create a new Eclipse extension.

As described in Section 3.3, the plugin is split into several components for portability reasons. Therefore, the prototype actually consists of a total of four eclipse plugins, as can be seen in Figure 4.3. Moreover, it was decided to use a model-driven software development approach similar to the one described by Beydeda et al. [BBG+05]. As a big part of the tool is a list and form for viewing and manipulating data, which can be described formally, large parts of these two UI elements can be automatically generated that way.

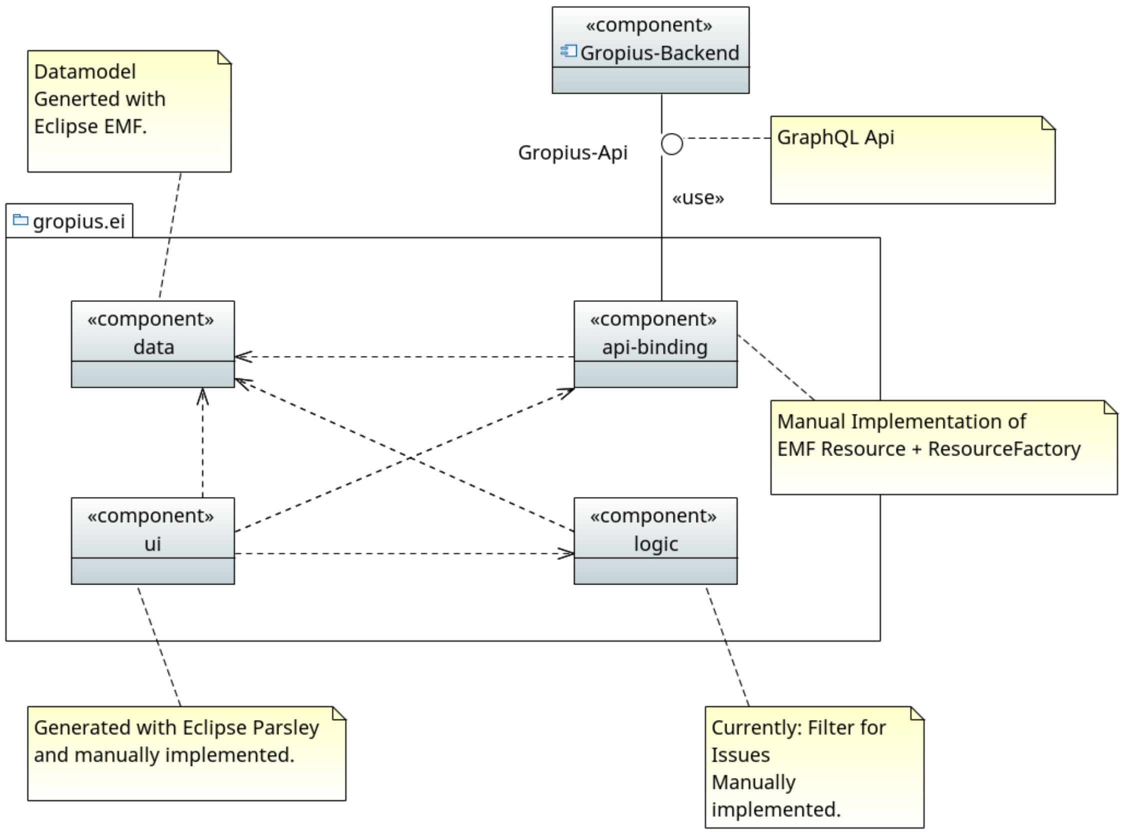

«component»
Gropius-Backend
Datamodel Generted with Eclipse EMF.
Gropius-Api
GraphQL Api
«use»
gropius.ei
«component»
data
«component»
api-binding
Manual Implementation of EMF Resource + ResourceFactory
«component»
ui
«component»
logic
Generated with Eclipse Parsley and manually implemented.
Currently: Filter for Issues Manually implemented.

Figure 4.3: Component Diagram of the Eclipse Extension

The same figure can be found enlarged in Appendix A

The plugin on the top left of Figure 4.3, called `data` contains the data model for the plugin. It is first modeled as an UML model, then converted to an EMF ecore model, and then the model code is generated based on that. Ecore is the metamodel to represent models in the EMF world [SBMP08].

The plugin `api-binding` is responsible for all communication with the Gropius back-end through the Gropius API. Additionally, it converts between the data format defined in the `data` plugin and the format used by the API. As mentioned in Section 2.1.3, the API provided by the Gropius system uses GraphQL. Therefore, the client needs to specify exactly what data is required and create mutations for any changes to be sent to the server. To be able to create these mutations, the difference between the old and the new version of the change needs to be recorded. Both the specification of the required data and the creation of mutations is also done by this component.

The third plugin, just called `ui`, implements all the UI elements. It is the only plugin of the four that directly interacts with the Eclipse IDE. This way, it is the only plugin, which needs major changes when the tool is ported to another IDE, which supports Java-based plugins. It uses the EMF Parsley framework introduced by Lorenzo Bettini [Bet14] to generate the issue list and issue details view based on the data model from the `data` component.

The `logic` plugin holds all classes containing logic needed by the `ui`, which is not Eclipse specific. One example would be the classes for filtering the issues based on some properties. They are used by the filters for the issue list but do not require any IDE specific functions.

A separate Eclipse plugin for the messaging component would likely make sense. However, it was not modeled, as the idea of the messaging component was dropped from this thesis for time constraint reasons.

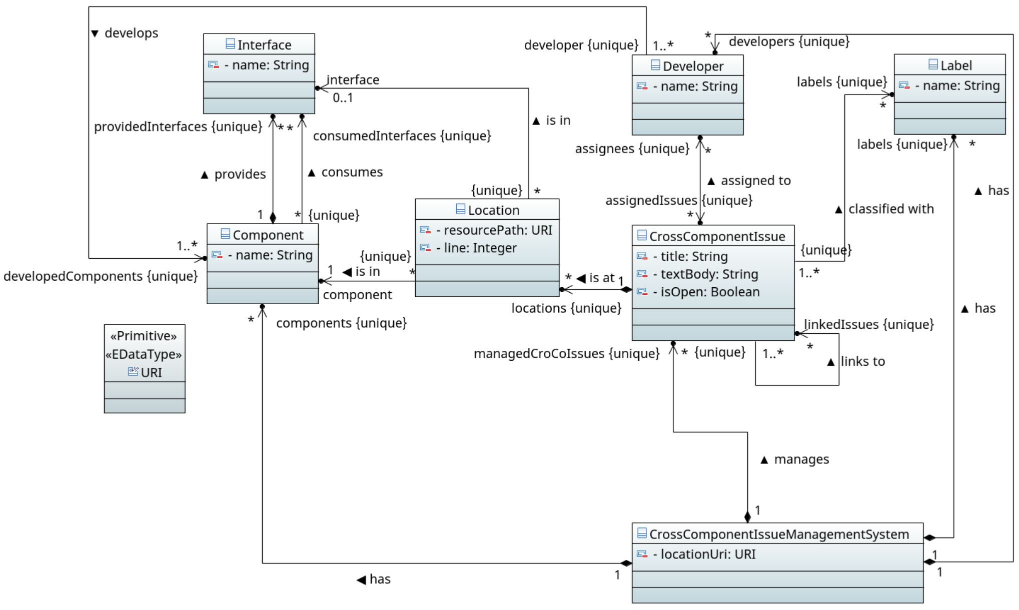


Figure 4.4: Class Diagram of the Data Model

The same figure can be found enlarged in Appendix A

The class diagram of the data model for Gropius EI used to generate the `data` component can be seen in Figure 4.4. It is based on the Domain Meta Model shown in Section 2.1.3. At the core of the data model is the `CrossComponentIssue`, with a title, text body, and a flag indicating whether it is open. Additionally, it has various references to other objects, such as the assigned developers.

All `CrossComponentIssues` are contained within a `CrossComponentIssueManagementSystem`. That also has a list of all Labels, Developers, and Components relevant for the associated issues. The tool always works with the issues contained in exactly one `CrossComponentIssueManagementSystem`.

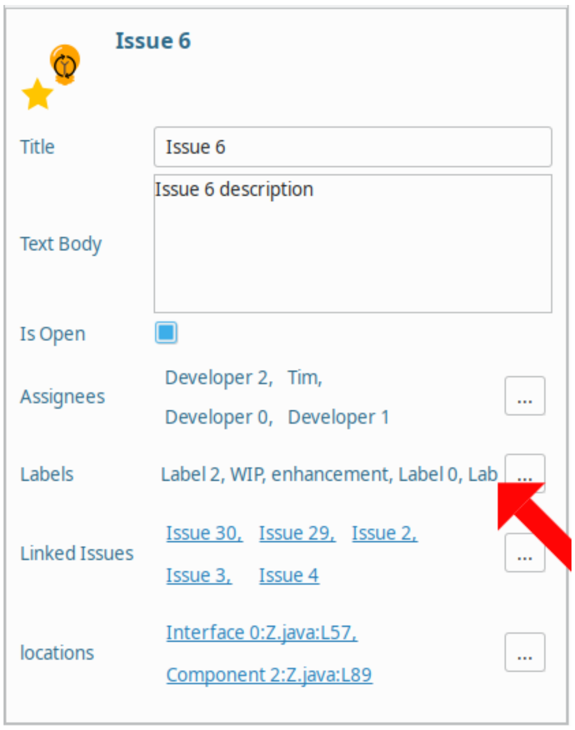
One important reference for the functionality of the tool is the `linkedIssues` list of a `CrossComponentIssue`. It allows Gropius EI to show users linked issues as well as letting users navigate between them.

Another core concept of this model are the `Locations`. One issue can be at zero or more `Locations`, which specify the resource and line the `Location` is at. Additionally, a `Location` can be in a `Component` or an `Interface`. `Interfaces` can be provided by exactly one `Component` and consumed by any number of them.

4.4 Eclipse Plugin Implementation

During the implementation phase, one of the biggest challenges was to understand and to correctly use EMF Parsley as well as EMF. Not a lot of documentation and tutorials could be found for either of them. However, as both are open source, answers for specific questions could be searched and found in the code itself. Yet, this process is rather tedious and slow, so a lot of time was spent just looking through these frameworks' source code.

EMF Parsley is well suited for generating default UIs for data models. It has basic support to customize them, but very little existing customization already available. Therefore, a lot of the implementation work during this thesis was improving classes, which already exist in parsley and adding new variants of them, which allow further customization. Most of them could be contributed to EMF Parsley after a little cleanup.



Issue 6

Title	Issue 6
Text Body	Issue 6 description
Is Open	☑
Assignees	Developer 2, Tim, Developer 0, Developer 1
Labels	Label 2, WIP, enhancement, Label 0, Lab
Linked Issues	Issue 30, Issue 29, Issue 2, Issue 3, Issue 4
locations	Interface 0:Z.java:L57, Component 2:Z.java:L89

Figure 4.5: Difference between original and new Control for Lists

One example is the UI element displayed inside a form for the value of properties, which are lists. Parsley only supports to display the string representation of the complete list in one label. The new implementation of the same class allows child classes to customize this by overwriting some methods. Based on this, some other classes were implemented using a grid of controls instead of a single label. This allows for nicer formatting when there are too many elements for one line. Additionally, this allows the use of links as the control for each element, which is used, for example, by the `linkedIssues` property. The difference between the original and the improved version can be seen in Figure 4.5. The control for the list `Labels` is created by the original, and you can see the last label is not entirely readable. The other controls are the new ones, with `Developers` using a grid of labels and the other a grid of links.

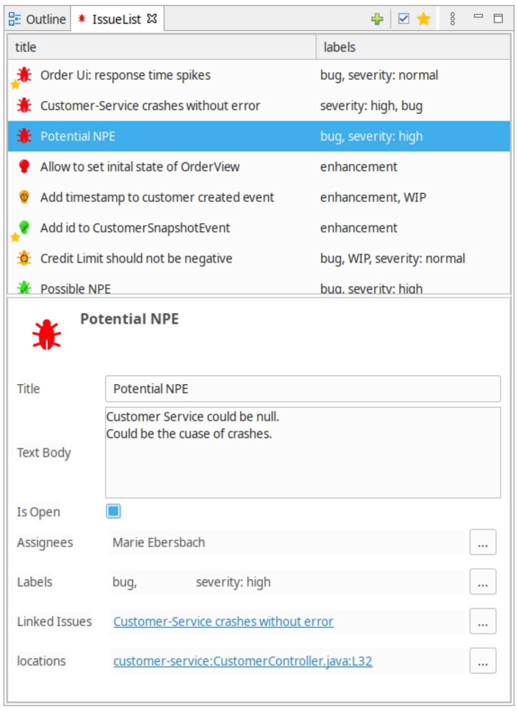
Another significant amount of time was taken by working on the `api-binding` plugin. As the Gropius back-end was not ready in time, the component was implemented to work with a simple mock-up of the back-end. Because of the fact, that the data from the API is transformed into the data model generated with EMF, the GraphQL API cannot be used as intended. Instead, all relevant data needs to be queried at once every time the data is reloaded from the server. This problem was detected too late to change the overall architecture of the plugin, mainly because the decision to switch to GraphQL for the Gropius API was made after the concept for Gropius EI was already done. However, to achieve this, a complex query needs to be sent to the server. Additionally, no appropriate Java library could be found for using GraphQL APIs without manually generating the queries and manually parsing the responses. In the end, some open-source ruby scripts[5] were used to generate helper classes from the Gropius GraphQL schema. The next problem was, that the mock-up of the back-end used to implement the `api-binding` component is not sophisticated enough. The returned data is not consistent in itself, preventing a correct transformation into the Gropius EI data model. As the real back-end would not be ready in time, the decision was made to halt implementing the `api-binding` plugin for now. For the remainder of the thesis, the data is generated by a mock-data generator and stored in a file instead.

4.4.1 The Prototype

Due to the lack of time, many of the features described in Section 3.3 were not implemented. Everything that was implemented is presented in the next few paragraphs.

As stated in Section 3.3 the main elements of Gropius EI are the issue list and the issue detail view. Deviating from the concept, those two elements are implemented in one big view element, as shown in Figure 4.6. This is the case because EMF Parsley provides such a view, and implementing interactions between the two parts is easier that way. The issue list is in the upper half of the picture. It consists of multiple columns, each responsible for one property of the issues. By default, only the title and label columns are shown, but the user can change this through the menu on the top right. In theory, all properties that can be seen in the form in the lower half of the image can also be displayed as columns in the list. However, especially the text body does not make much sense to be displayed in the list, as it can take a lot of space. In the first column of the list, the icons from 4.2 are displayed in addition to the property of that column. Currently, the annotations for this issue being caused by another or causing another issue are not used because that feature has not yet been implemented. The entries of columns with a list of values are not sorted in any way but display the values in the order they are contained in the respective list for this property. This list is initially ordered by when the value was added but can be reordered by the user in the dialog for editing that property.

[5] https://github.com/Shopify/graphql_java_gen



Outline | IssueList

title	labels
Order Ui: response time spikes	bug, severity: normal
Customer-Service crashes without error	severity: high, bug
Potential NPE	bug, severity: high
Allow to set inital state of OrderView	enhancement
Add timestamp to customer created event	enhancement, WIP
Add id to CustomerSnapshotEvent	enhancement
Credit Limit should not be negative	bug, WIP, severity: normal
Possible NPE	bug, severity: high

Potential NPE

Title	Potential NPE
Text Body	Customer Service could be null. Could be the cuase of crashes.
Is Open	☑
Assignees	Marie Ebersbach
Labels	bug, severity: high
Linked Issues	Customer-Service crashes without error
locations	customer-service:CustomerController.java:L32

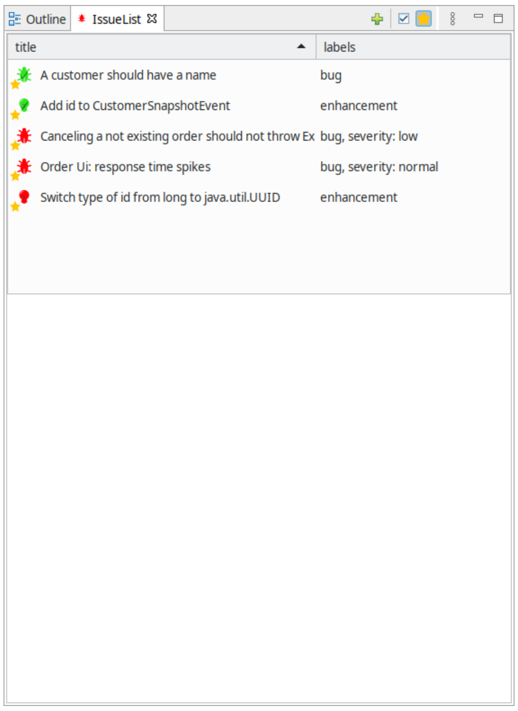
Figure 4.6: Prototype Issue List

As you can see in Figure 4.7, the issue list also supports sorting by a column. This is done by clicking on the column header. Furthermore, filtering is supported using the buttons in the bar on the top edge of the view. Currently, only the filters `Only show open issues` and `Only show issues assigned to me` are implemented. However, all the other filters specified in Section 3.3 should be fairly straight forward to include.

Figure 4.6 also shows the details view, which additionally is the form for editing a selected issue. On the top of that form, the icon and the title of the issue can be seen. Below that, all properties are displayed. Like the issue list, the properties with multiple values are ordered according to the order in the data instead of being sorted somehow. For the `Linked Issues` and the `Locations` each value is a link, selecting the correct issue in the list and jumping to that location in the code, respectively. To change a textual property, the user can simply type in the corresponding text area. The properties, which contain a list of values, can be modified using the button to the right.

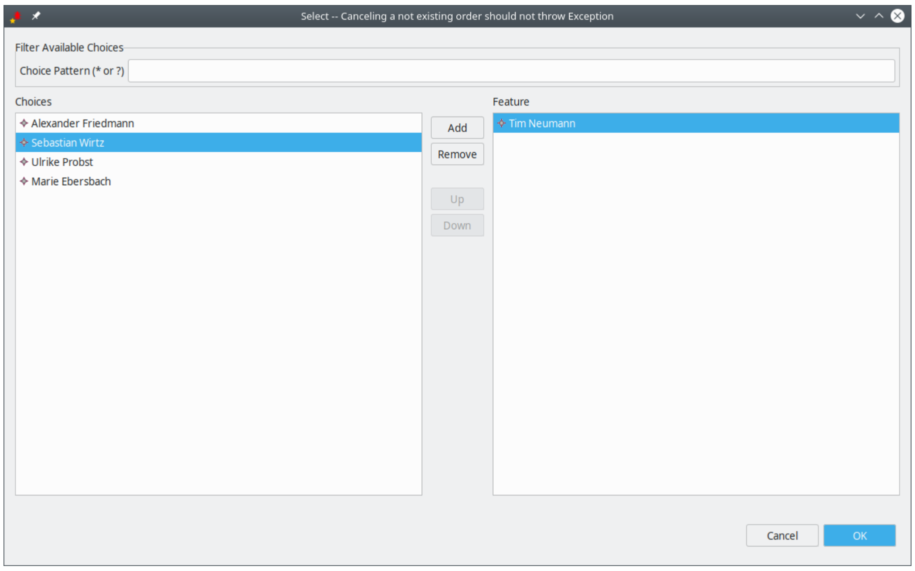
When that button is pressed a dialog similar to the one in Figure 4.8 is opened. With it, the user can select the values for that issue from a list of all possible values for the property. This dialog also allows the reordering of the values mentioned above. Changes done in the dialog are applied when the `Ok` button is pressed.

The only exception for the editing dialogs is the `locations` property. For those a dialog (shown in Figure 4.9), which allows creating and editing locations is opened. Using the `New` button, a new location is added to the list of locations for that issue. Then it, as well as any existing location, can be edited with the left half of the dialog.


Outline | IssueList

title	labels
A customer should have a name	bug
Add id to CustomerSnapshotEvent	enhancement
Canceling a not existing order should not throw Ex	bug, severity: low
Order Ui: response time spikes	bug, severity: normal
Switch type of id from long to java.util.UUID	enhancement

Figure 4.7: Prototype Issue List with Filter and Sorting


Select -- Canceling a not existing order should not throw Exception

Filter Available Choices

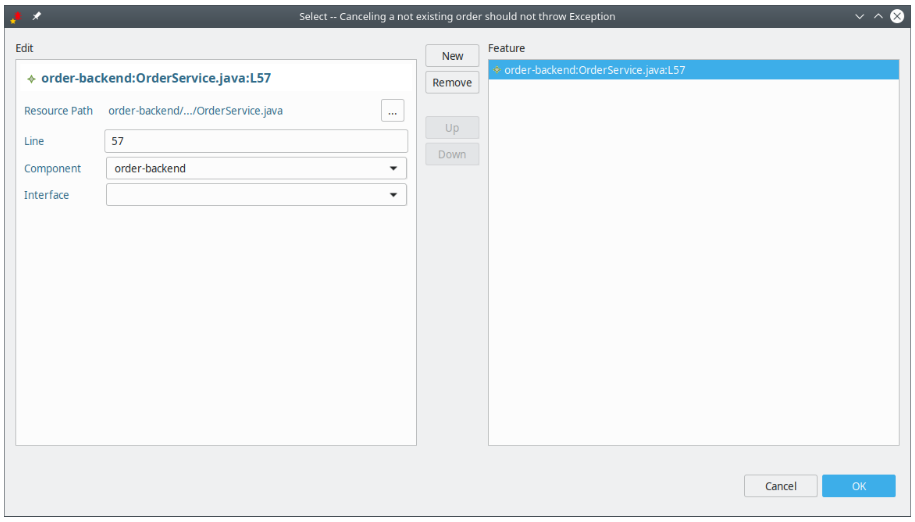
Choice Pattern (* or ?)

Choices
- Alexander Friedmann
- Sebastian Wirtz
- Ulrike Probst
- Marie Ebersbach

Feature
- Tim Neumann

Figure 4.8: Prototype Edit Developers Dialog

After all changes, the issue list is in an unsaved state, indicated by a star in the view name in the tab of the view. To save the changes, all usual ways to save a resource in Eclipse can be used, such as the save button in the toolbar.


Select -- Canceling a not existing order should not throw Exception

Edit

order-backend:OrderService.java:L57

Resource Path order-backend/.../OrderService.java

Line 57

Component order-backend

Interface

Feature

order-backend:OrderService.java:L57

Figure 4.9: Prototype Edit Locations Dialog

The green plus button seen above the issue list in Figure 4.6 can be used to create new issues. It adds a new issue with empty properties to the list, which can then be edited using the form below. The feature for creating issues directly from some lines of code has not yet been implemented.

5 Evaluation

For evaluating the usefulness of the extension, an expert review and survey are performed. To acquire the questions for this survey, a process based on the Goal Question Metric (GQM) approach introduced by Caldiera et al. [CR94] is used. Section 5.1 covers that process and presents its results. Section 5.2 describes the design and realization as well as the results of the expert review and survey. In Section 5.3, the results of the survey, as well as the validity of the thesis, are discussed. Finally, the identified threats to the validity of this work are described in Section 5.4.

5.1 Applied Goal Question Metric (GQM) approach

The process described in this section does not exactly follow the original Goal Question Metric (GQM) approach, but is based on it. The objective of the process is to find good questions, which can be used to evaluate the success of this thesis. Based on these questions, an expert survey can then be performed.

The first step is to identify the goals of the work. For this thesis, the following could be identified as the primary and only goal **(G1)**: Improve the efficiency, effectiveness, and convenience for developers when working with issues in environments with many components and multiple teams during their software development process.

Next, questions have to be found, which can be used to evaluate whether that goal is reached. For that purpose, the following three questions could be identified:

Q1 What problems do developers face when working with issues affecting multiple independent projects during the software development/engineering process?

Q2 Which of these problems does this thesis's tool solve and to which degree?

Q3 Can the industry imagine using such a tool in production?

As it was clear from the start that an expert review and survey would be conducted, the metrics for these questions did not need to be determined. However, it must be investigated if an expert review is a reasonable metric for these questions. For **Q1**, an expert survey is a reasonable metric if the experts are developers who currently work or have worked in a field with multiple independent projects or have enough experience to put themselves into that situation. It is an adequate metric for **Q2** if the experts fulfill the conditions above and have used the thesis' tool or at least were given an introduction about the features of the tool. An expert survey is a reasonable metric for **Q3** if the experts are or have been working in the industry and fulfill the above conditions.

Therefore, the evaluation metric (**M1**) for all questions is an expert review and survey with the correct experts, who are introduced to Gropius EI.

5.2 Expert Survey

The main element of this evaluation is the expert survey, where selected experts answer a set of prepared questions. But first, an expert review of Gropius EI is performed. This makes sure the experts know the tool and its features and has the additional benefit of direct feedback about the prototype. The process of the review and survey is explained in Section 5.2.1. The results from both are presented in Section 5.2.2.

5.2.1 Design and Realization

To perform an expert review and survey, first, a group of experts has to be found. To achieve this, various developers from academia and industry, including a number companies in different countries were contacted and asked to participate in this review. If they were generally interested, the thesis' concept and the review and survey procedure were explained to them. In total, 24 experts were contacted, of which 14 agreed to take part.

As preparation for the expert review, a detailed guide on installing Gropius EI was produced to ensure experts could install it on their own device if they wanted. It can be found in Appendix B.1 and the exact release used for the review is attached as Appendix B.2.

Additionally, a scenario was created in which experts could test the tool in as real as possible conditions. This scenario consists of an Eclipse workspace with a Java-based demo software using multiple components[1] and a Gropius EI data file. This data file contains the correct components and interfaces for the workspace as well as some developers and labels in addition to several invented issues. These issues are as realistic as possible with a title that could be found in a live IMS somewhere, a feasible text body, typical labels, sometimes linked issues, and most with at least one location in the workspace. Furthermore, a few tasks were prepared, which were used to introduce each feature to the expert. These tasks were typical things a developer would need to do during his workday, like finding an issue in the code or creating a new issue. For each task, a detailed explanation, how the task can be done using Gropius EI is also included. The scenario can be found in Appendix B.4 and a short explanation as well as the mentioned tasks in Appendix B.3.

After an expert agreed to participate, they were provided with three options for executing the expert review and asked which they preferred. The first option was to install the tool and the scenario on their device with the help of the mentioned installation guide. Then, an online meeting would be scheduled, such that the tasks from the scenario could be completed while I watch with the help of a screen sharing solution. That way, I could help the experts with any problems and directly answer questions as well as guide them through the prepared tasks instead of sending the tasks. The second option was that the expert installs both on their device and goes through the scenario doing the tasks independently. In this case, they were sent the tasks, including the explanations in a written document. The last option was that I would present the tool in an online meeting using screen sharing tools. That way the expert did not need to install anything on their device. For that reason, this was also the fastest option. On the other hand, the experts could not use the tool themselves but just watch me perform the tasks. In every case, the expert was asked for general feedback.

[1]https://github.com/eventuate-tram/eventuate-tram-examples-customers-and-orders

After an expert completed the review he was sent the following questions and asked to answer them:

1. What do you see as the most significant problems when working with issues affecting multiple independent projects during your software development/engineering process?
2. Do you think the following points represent problems or challenges of existing issue management systems you face as a developer?
    - The lack of insufficient support of linking issues to multiple specific software development artifacts accurate to the line, especially if the artifacts are part of multiple independent projects.
    - The lack of insufficient support of linking issues to each other
    - The lack of inadequate support of navigating from one to the next issue through the chain of related or dependent issues.
3. For which of these problems (from points 1 and 2) would you say Gropius EI completely solves them?
4. For which of these problems (from points 1 and 2) would you say Gropius EI reduces them?
    - How far would you say are each of these problems reduced?
5. Could you imagine to use Gropius EI in your day-to-day business?

These questions are based on the evaluation questions **Q1** to **Q3** presented in Section 5.1. Question 2 was added in order to get good answers to questions 3 and 4 even if the expert did not find a lot of answers to question 1. The points for question 2 were conceived in an internal brainstorming session.

5.2.2 Results

Of the twelve experts, who agreed to participate, eight actually took part in the review and survey. However, only seven could send me their answers in time. All of these experts are software developers. One works in academia and the others in the industry. The industry experts work in various different companies from small businesses with just a few employees to well known global corporations, from IT consultancies to companies, where software is just a small part of the final product. Furthermore, the experts have very different levels of experience. One is working as an intern during his masters' studies, others have been developers for a few years, and some are senior software engineers who have been working in the industry for decades.

Six of the seven experts appreciate the general concept. They also see potential in the extension but would like a few more features. In the following paragraphs, the general and technical feedback for the extension is summarized, followed by the results of the expert survey.

While reviewing the implemented extension, many experts missed features that were planned but could not be implemented because of the time. Two experts also noticed some minor bugs with the issue list, like it freezing under some circumstances when changing the displayed columns. The most requested feature was the ability to filter the issue list by more criteria, especially the ability to search by text contained in the title or description. Another feature that was requested multiple times was the ability to create locations and issues from the source code editor. Additionally, multiple experts stated, they would like to be able to navigate from a marker in the code to the corresponding issue in the issue list. Furthermore, the ability to group issues in the issue list by various properties was requested by some experts. One expert also requested the ability to specify the kind of a relation between two issues. Moreover, another expert would like some kind of key for the icons and would prefer if the state of the issue was only visualized by one aspect, preferably the color and not the small black overlay. He also stated that an arrow pointing up might be easier to identify as an enhancement than the light bulb. One expert would like the ability to link issues to other kinds of software development artifacts such as Portable Document Format (PDF) documents. He also stated that Gropius EI is missing any kind of anonymization, which is typically required for cross-company collaboration. Additionally, he would expect drag-and-drop capabilities. Moreover, he would prefer if the interface for adding assignees and labels would be more similar to the one of existing IMS like Github. Another expert suggested also showing the markers in the Eclipse problems view and as annotations on the files, folders, and projects in the Eclipse package explorer. Furthermore, he would like the issue list and issue details UI element to be separate views, so they can separately be moved within the Eclipse window, making it easier to find an optimal layout. He would also prefer the default filter for the issue list to only show open issues. Moreover, two experts stated that they would like to be able to sort the list of labels for each issue in the issue list, such that, for example, the work-in-progress label is always the first one displayed. Finally, one expert thinks that the issue locations would rarely be used in practice as it is too much extra work and most developers dislike managing such additional data.

The existing features the experts liked the most were the ability to quickly navigate between issues and to the code as well as the markers in the editor. One also stated he likes how the existing save button of Eclipse is used to save the data of the issue list. Another especially liked the icons because they allow him to quickly get an overview of the issues and some of their properties without being overloaded. He also liked that there are not too many types of issues.

The experts' answers to question 1 were rather diverse. One expert does not see any problems with existing tools when it comes to working with issues affecting multiple independent projects. The next expert stated that the most significant problem is tracking issues that depend on each other across multiple projects. In the mind of another expert, the accurate description of the issues and a good specification of the issue location are the core problems. Similar to that, the fourth expert thinks that the lack of support for specifying the issue location other than as part of the description or a comment is a major problem. He also stated that he often misses the ability to specify the kind of a relation between two issues or not enough different types of relations are provided. On the other hand, he thinks that most IMS have too many different types of issues, which causes confusion and allows too much room for interpretation. Another expert stated that the most significant problem is that some, especially closed-source, projects do not have a public IMS, and some open-source projects are managing their projects in a custom IMS instead of a well-known one like the Github issue tracker. One expert also sees the problem, that developers need to keep the overview of the components to know in which IMS they need to create or search an issue. He also thinks that it is a problem when one issue affects multiple components from the customers' perspective. Another expert stated, the biggest problem for him is managing dependencies when a fix for an issue might break compatibility and the new version of multiple components, therefore, need to be deployed simultaneously.

Three experts completely agree with first point of question 2. Two experts think it would be a nice feature, but do not see its lack as a problem. The expert, who does not see any problem with existing IMS when working with issues affecting multiple independent projects, states that the real problem is visualizing this link in his IDE. One expert stated that existing IMS mostly support this well enough for his needs. He and one of the experts only considering this a nice feature also stated that a concrete location cannot be determined for most issues, reducing the need for this feature.

Four experts stated, that the feature mentioned in the second point is very important, but that most IMS have adequate support of it. Two experts agree that this is a problem with existing IMS. One of them also stated, it should be as easy as possible to create such a link. Another expert states that sometimes the support of this feature is inadequate, especially when working with independent projects.

Similar to the previous point, all experts agree that third feature is important, but four of them think the support of it is good enough in existing IMS. One thinks that very few IMS have adequate support for this. Two experts also stated that a graphical overview of the linked issues would be nice. The expert, who wants the ability to specify the kind of relation, also mentioned that only having one type of relation is bad for the navigability between issues. It makes the list of related issues long and confusing. One expert also states navigating between issues is not a problem when they are properly linked but initially finding the IMS of another project can be a problem when only the name or identifier of the related issue is given.

Most experts answered question 3 and question 4 together, therefore the answers are also summarized in this paragraph. As the first expert didn't see any problems in response to questions 1 and 2, Gropius EI does not solve or reduce any problems in his eyes. Another expert stated, that the problems he sees, which are the missing public IMS, custom IMS are not solved by Gropius EI but all the points of question 2 are reduced by Gropius EI, even though other IMS also do that. One expert stated, that he thinks Gropius EI solves all of the problems from question 2. According to another expert, the concept reduces these problems and the need to keep an overview of the components and their respective IMS, but the implementation still needs some work. The other three experts think Gropius EI solves the problem of linking issues to artifacts of multiple projects. One of them also stated that the problem of linking issues is solved by Gropius EI in his eyes. Another expert also states that navigating between issues is easy with Gropius EI, but the links should have more than one type for an even better overview.

Only one expert can not imagine using Gropius EI at all. The other experts would not use it right now, as it lacks many features, especially the ability to synchronize data with the Gropius back-end, but could imagine using it in the future, once these issues have been resolved. One expert would only use it once it is available for IntelliJ IDEA or Visual Studio Code. Another point is also the support of Gropius for various IMS. One expert would, for example, only use Gropius EI if it has support for Jira.

Finally, the expert who did not see any problems with existing IMS when working with issues affecting multiple independent projects stated he has other problems with them, which are solved by Gropius EI. As all major IMS are web-based, source code is also opened in the browser whenever they link to it. However, in the browser, various IDE features for exploring or editing the code are not available. Furthermore, he thinks that with existing IMS, the effort for creating a new issue while working in the code is larger than required. In his eyes, both of these points are solved by Gropius EI.

5.3 Discussion

In this section the validity of the concept and the success of this thesis is discussed based on the GQM plan introduced in Section 5.1 and the results of the expert survey presented in Section 5.2.2. For that, three hypotheses based on the research questions **Q1** to **Q3** presented in Section 5.1 are established. Then, their validity is discussed based on the results of the expert review. Finally, if none of the hypotheses is rejected, the goal **G1** has been reached.

The following three hypotheses, one for each research question, were created:

H1 Developers do face problems when working with issues affecting multiple independent projects during the software development/engineering process.

H2 These problems are solved by this thesis' tool at least to some degree.

H3 The industry can imagine using such a tool in production.

As shown by their answers to 1, the experts found various problems, which occur when working with issues affecting multiple independent projects. Additionally, almost half of the experts see the first point of question 2 as a problem, and two more think it would be a nice feature. Another does not exactly have that problem, but a related one. Furthermore, the other two points provided for question 2 are seen as essential features by the experts, even though most think their support of existing IMS is good enough. However, these points are also seen as problems by three and two experts, respectively. In total, it can be said that there are at least some problems for developers when working with such issues. Therefore, *hypothesis **H1** is accepted*.

According to most experts, Gropius EI solves at least some of these problems. Additionally, all but one expert think it at least reduces these problems. Furthermore, this one expert stated that Gropius EI solves other problems he has with existing IMS. Therefore, it can be said it also improves the efficiency, effectiveness, or convenience for him when working with issues, even though the problems are not specific to issues affecting independent projects. Consequently, all experts think there is some benefit of using Gropius EI. Even though not all of the problems are solved equally well, it can be said, the problems all together are solved to a significant degree. Therefore, *hypothesis **H2** is accepted*, too.

Finally, all but one expert could see themselves using Gropius EI in their daily business once its implementation has advanced enough. This clearly indicates that the industry can imagine using it in production. Therefore, *hypothesis **H3** is accepted*, too.

All three hypotheses have been accepted. Therefore it can be said that this thesis's goal, as defined by **G1** in Section 5.1, has been reached.

5.4 Threats to Validity

This section discusses which threats to the validity of the results discussed above could be identified. An overview of these threats can be seen in Figure 5.1. The threats can be grouped into four categories [RH09].

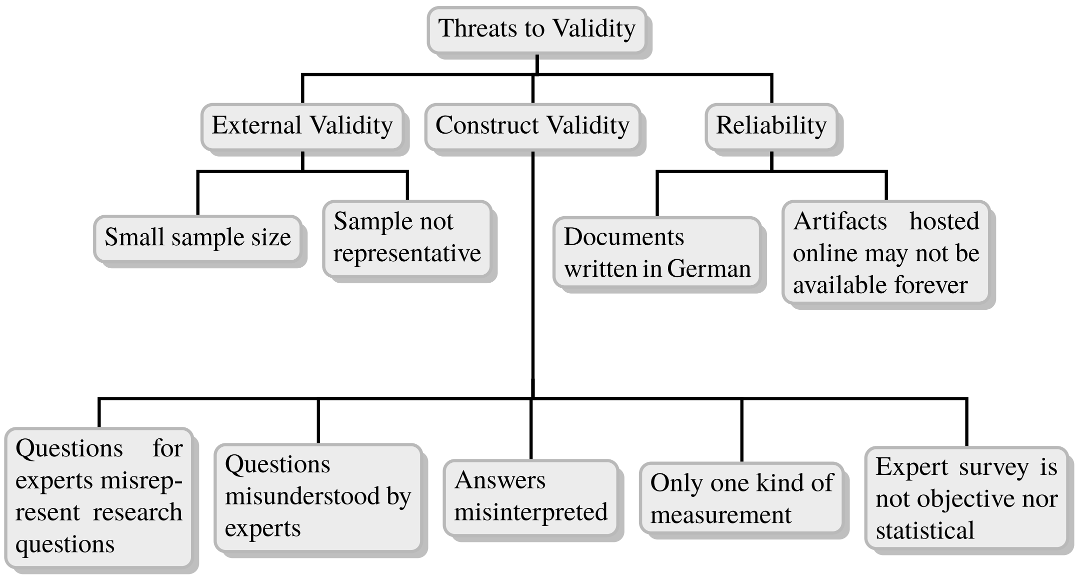

Threats to Validity
External Validity
Construct Validity
Reliability
Small sample size
Sample not representative
Documents written in German
Artifacts hosted online may not be available forever
Questions for experts misrepresent research questions
Questions misunderstood by experts
Answers misinterpreted
Only one kind of measurement
Expert survey is not objective nor statistical

Figure 5.1: Overview over the Threats to Validity

The first category investigated is internal validity. This aspect of validity is concerned with whether the observed results are actually caused by the changes introduced by this work, or if the results could be the effects of any unknown influence [RH09]. However, as the experts were directly asked if Gropius EI solves or reduces the stated problems, no threat to this aspect of validity could be identified.

The next kind is construct validity. It represents the correctness of the assumption that the collected results actually correspond to the research questions [RH09]. For this aspect, a few threats could be identified. First, the questions sent to the experts, which are presented in Section 5.2.1 could misrepresent the research questions introduced in Section 5.1. Next, these questions could have been misunderstood by one or more experts. Furthermore, the answers of the experts could have been misinterpreted. Moreover, just a single kind of measurement was performed. Finally, an expert survey does not produce objective or statistical results. The results are subjective answers from the experts given in free text form, such that statistical evaluation is difficult.

Another aspect is the external validity. That is concerned with whether the results can be generalized beyond the small studied sample [RH09]. The following threats were identified for this category: First, the sample size was small compared to all developers working with issues in a relevant field. Second, the experts were not picked randomly, but all had prior contact with me or my supervisors. Therefore, the sample was not very representative of the entire population.

The last aspect of validity is reproducibility or reliability. It deals with any reasons a separate evaluation by other researchers reproducing this work would not have the same conditions [RH09]. For this aspect, no major threats could be identified, as the whole process is described in detail in Section 5.2.1, and all artifacts used can be found in Appendix B. The first minor threat found is that these documents are written in German and would need to be translated for researchers or experts who do not speak German. The other minor threat is that the artifacts hosted on the internet referenced in the appendix might not be available forever.

6 Conclusion

This chapter concludes this thesis by first giving a short summary in Section 6.1, highlighting the most important lessons learned during the process of this thesis in Section 6.2, and finally discussing what work could be based on this thesis in the future in Section 6.3.

6.1 Summary

The goal of this thesis was to develop, implement, and evaluate a concept for integrating Gropius into the developer's IDE and therefore improve the efficiency, effectiveness, and convenience for developers when working with issues in environments with many components and multiple teams during their software development process.

First, a concept for such an IDE extension was developed based on a requirements engineering process. This concept is independent of the IDE and can therefore be used to develop an extension for any IDE. At the core of the concept is the issue list, and the issue detail UI element.

Then, an Eclipse extension has been implemented based on this concept. Large parts of it are created using a MDSD approach. Therefore a lot of the code is generated. Even though various features could not be implemented in time, the result helps demonstrate the concept. It can easily be developed into a full issue management extension using Gropius.

Using this extension, an expert review has been performed, followed by an expert survey. The former was used to gain feedback about the extension while the latter served to evaluate the concept. While the experts generally like the concept, not all see the problems it is trying to solve. However, most experts do agree with at least some of the problems and think Gropius EI does solve them. Finally, all but one expert can imagine using it in the future. Based on the discussion of these results, it can be said that the thesis' goal is reached.

In the end, this thesis provides a detailed concept for issue management plugins working with the Gropius framework as well as a basic implementation of it together with a lot of feedback for future improvements.

6.2 Lessons Learned

The most important thing I learned during the process of this thesis is using a MDSD approach, especially working with EMF and particularly EMF Parsley. Both do not have a lot of documentation, but by working with them and looking into their source, I learned a lot about their use but also their internals.

Additionally, I gained practical experience with planning and performing an expert review and survey. Especially creating the research questions using the GQM approach and evaluating the threats to validity gave me a lot of insight into those areas.

Moreover, I also practiced and improved my literature research as well as scientific writing skills. I especially learned searching in an area with very little scientific work and finding hidden papers.

6.3 Future Work

One possible first step for any future work is obviously improving Gropius EI. The remaining features proposed in the concept, as well as the ones suggested by the experts, could be implemented. Once the extension is in a more usable form and the Gropius back-end also has support for some IMS, a study with many more developers could be performed in one or more companies. An instance of the back-end could be set up within each company and linked to their existing IMS and other IMS of projects they are using. Then, the developers of the company could try using Gropius EI in their daily business. This study could reveal the performance increase gained by using the extension.

Alternatively, extensions for other IDEs could be developed based on the concept of this thesis. This helps to evaluate how easily it can be applied to those other IDEs and how much work needs to be done to port the existing IDE-independent parts of the extension to another extension. It would need to be evaluated if it is easier to get the code generated by EMF working with another IDE or if it is easier to generate new code from the existing models with the help of another MDSD framework.

Bibliography

[Atl20] Atlassian. *What is an issue?* 2020. URL: https://support.atlassian.com/jira-software-cloud/docs/what-is-an-issue/ (cit. on p. 3).

[BBG+05] S. Beydeda, M. Book, V. Gruhn, et al. *Model-driven software development*. Vol. 15. Springer, 2005 (cit. on p. 22).

[Bet14] L. Bettini. “Developing user interfaces with EMF parsley”. In: *2014 9th International Conference on Software Paradigm Trends (ICSOFT-PT)*. IEEE. 2014, pp. 58–66 (cit. on pp. 6, 23).

[Bur05] E. Burnette. *Eclipse IDE Pocket Guide: Using the Full-Featured IDE*. Ö’Reilly Media, Inc.”, 2005 (cit. on p. 6).

[BVGW10] D. Bertram, A. Voida, S. Greenberg, R. Walker. “Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams”. In: *Proceedings of the 2010 ACM conference on Computer supported cooperative work*. 2010, pp. 291–300. DOI: 10.1145/1718918.1718972 (cit. on p. 3).

[CCPP08] S. H.-H. Chang, X. Chen, R. A. Priest, B. Plimmer. “Issues of extending the user interface of integrated development environments”. In: *Proceedings of the 9th ACM SIGCHI New Zealand Chapter’s International Conference on Human-Computer Interaction: Design Centered HCI*. 2008, pp. 23–30. DOI: 10.1145/1496976.1496980 (cit. on p. 6).

[Com75] Computerwoche. “Interaktives Programmieren als Systems-Schlager”. German. In: *Computerwoche* (Nov. 21, 1975). URL: https://www.computerwoche.de/a/interaktives-programmieren-als-systems-schlager (cit. on p. 6).

[CR06] E. Clayberg, D. Rubel. *Eclipse: building commercial-quality plug-ins*. Pearson Education, 2006 (cit. on pp. 6, 19).

[CR94] V. R. B. G. Caldiera, H. D. Rombach. “The goal question metric approach”. In: *Encyclopedia of software engineering* (1994), pp. 528–532 (cit. on p. 29).

[Epp11] T. Epping. *Kanban für die Softwareentwicklung*. Springer-Verlag, 2011 (cit. on p. 4).

[Fac18] Facebook, Inc. *GraphQL*. 2018. URL: http://spec.graphql.org/June2018/ (cit. on p. 3).

[Git20] GitHub, Inc. *About issues*. 2020. URL: https://help.github.com/en/github/managing-your-work-on-github/about-issues (cit. on p. 3).

[HQM19] M. Hajji, M. Qbadou, K. Mansouri. “Onto2DB: towards an eclipse plugin for automated database design from an ontology.” In: *International Journal of Electrical & Computer Engineering (2088-8708)* 9 (2019). DOI: 10.11591/ijece.v9i4.pp3298-3306 (cit. on p. 6).

[IUH09] A. Iqbal, O. Ureche, M. Hausenblas. "Integrating linked data driven software development interaction into an IDE". In: *SWESE 2009* (2009), p. 73 (cit. on p. 8).

[Jan09] J. Janák. "Issue tracking systems". PhD thesis. Masarykova univerzita, Fakulta informatiky, 2009 (cit. on pp. 3, 4, 8–10).

[MNH15] S. Mahmood, M. Niazi, A. Hussain. "Identifying the challenges for managing component-based development in global software development: Preliminary results". In: *2015 Science and Information Conference (SAI)*. IEEE. 2015, pp. 933–938. DOI: 10.1109/sai.2015.7237254 (cit. on p. 4).

[RH09] P. Runeson, M. Höst. "Guidelines for conducting and reporting case study research in software engineering". In: *Empirical software engineering* 14.2 (2009), p. 131 (cit. on pp. 34, 35).

[SBB20] S. Speth, U. Breitenbücher, S. Becker. "Gropius—A Tool for Managing Cross-component Issues". In: *European Conference on Software Architecture*. Springer. 2020, pp. 82–94. DOI: 10.1007/978-3-030-59155-7_7 (cit. on pp. xi, 1, 4, 5, 11).

[SBMP08] D. Steinberg, F. Budinsky, E. Merks, M. Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008 (cit. on pp. 6, 22).

[SL19] Á. M. Segura, J. de Lara. "Extremo: An Eclipse plugin for modelling and meta-modelling assistance". In: *Science of Computer Programming* 180 (2019), pp. 71–80. DOI: 10.31219/osf.io/hpwu3 (cit. on p. 6).

[Spe19] S. Speth. "Issue Management for Multi-Project, Multi-Team Microservice Architectures". MA thesis. University of Stuttgart, 2019 (cit. on pp. 1, 4).

[SS97] I. Sommerville, P. Sawyer. *Requirements engineering: a good practice guide*. John Wiley & Sons, Inc., 1997 (cit. on p. 11).

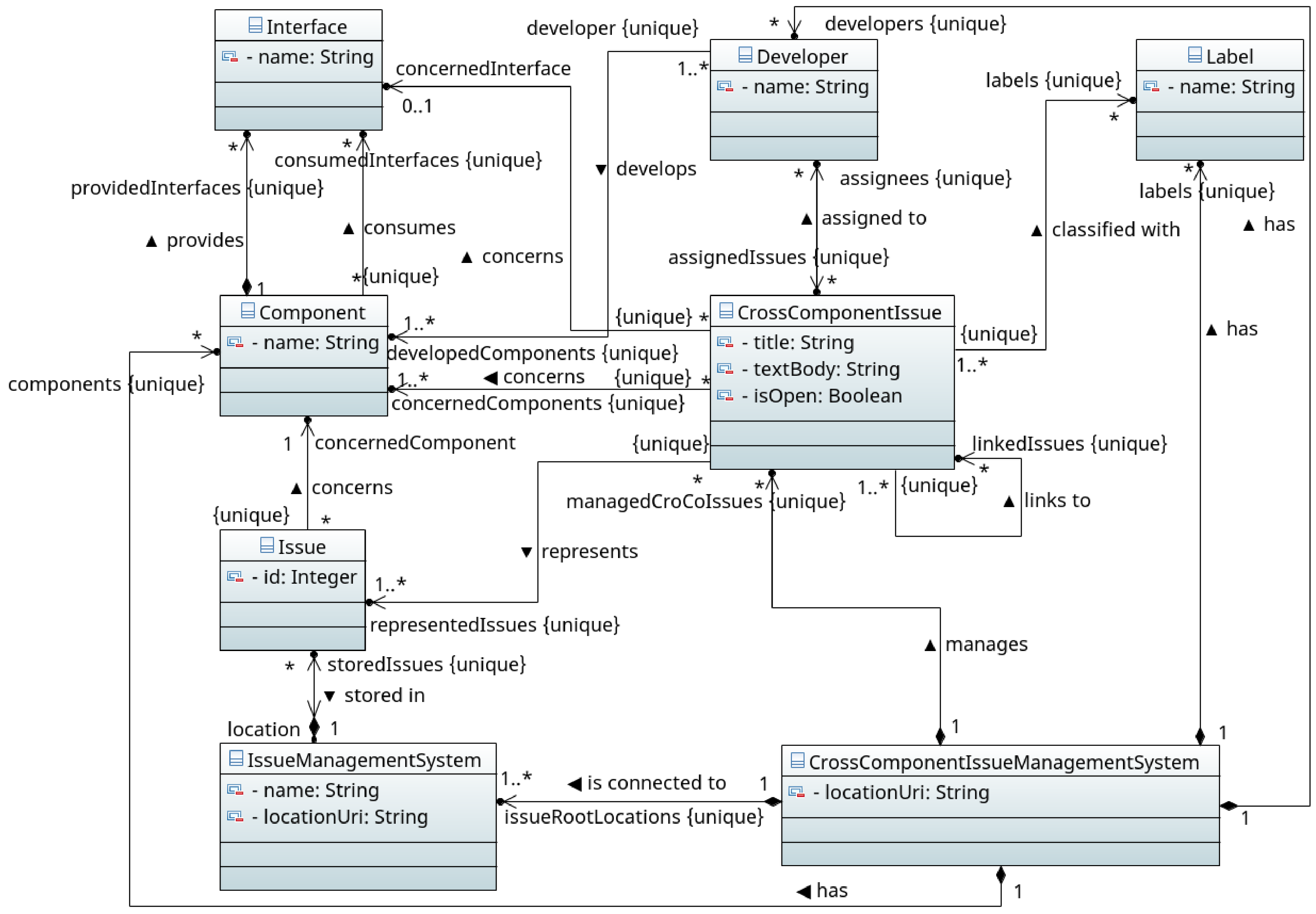
[VSB+13] M. Völter, T. Stahl, J. Bettin, A. Haase, S. Helsen. *Model-driven software development: technology, engineering, management*. John Wiley & Sons, 2013 (cit. on p. 6).

[Was90] A. I. Wasserman. "Tool integration in software engineering environments". In: *Software Engineering Environments*. Springer. 1990, pp. 137–149. DOI: 10.1007/3-540-53452-0_38 (cit. on p. 6).

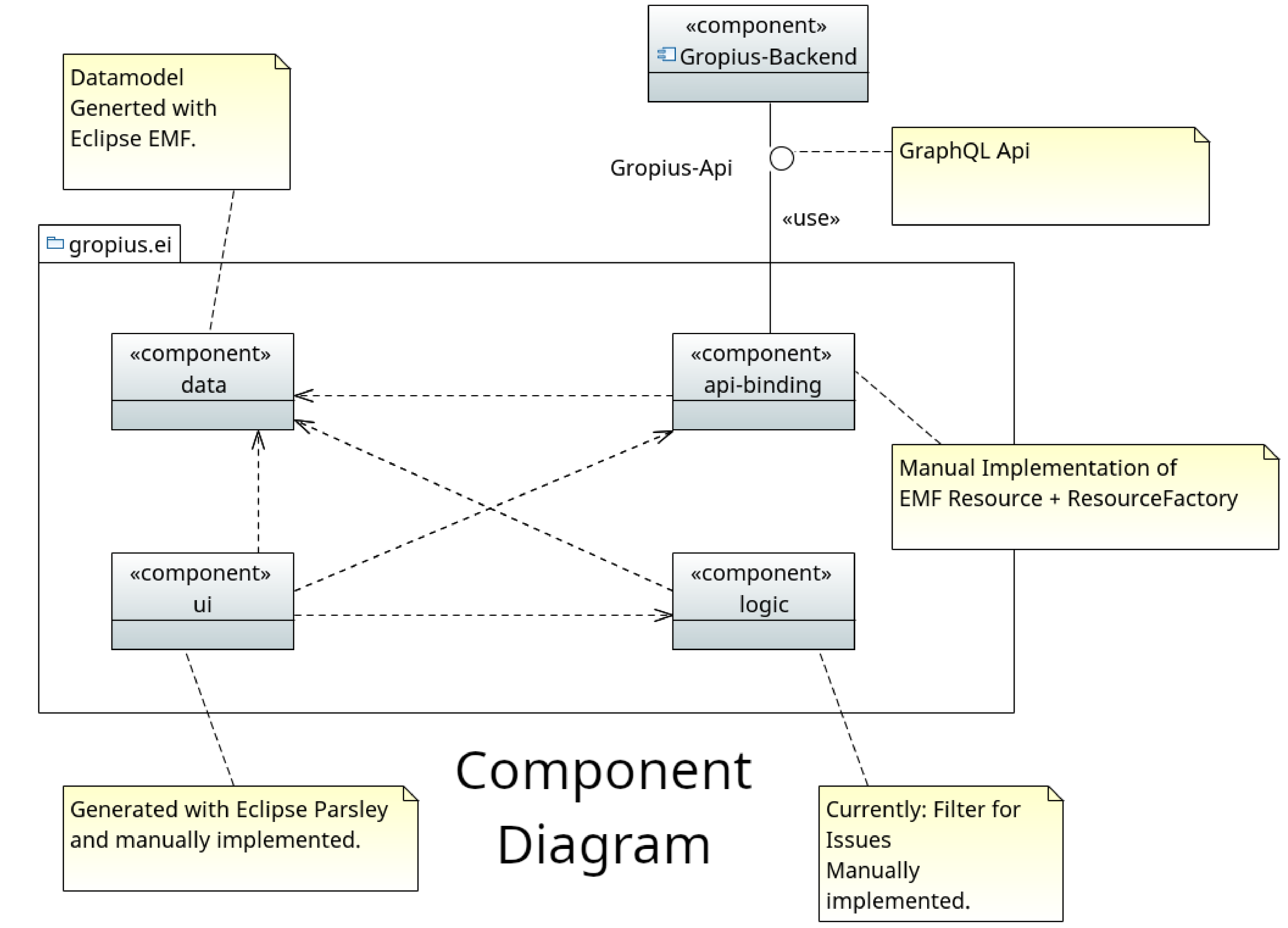
All links were last followed on November 11, 2020.

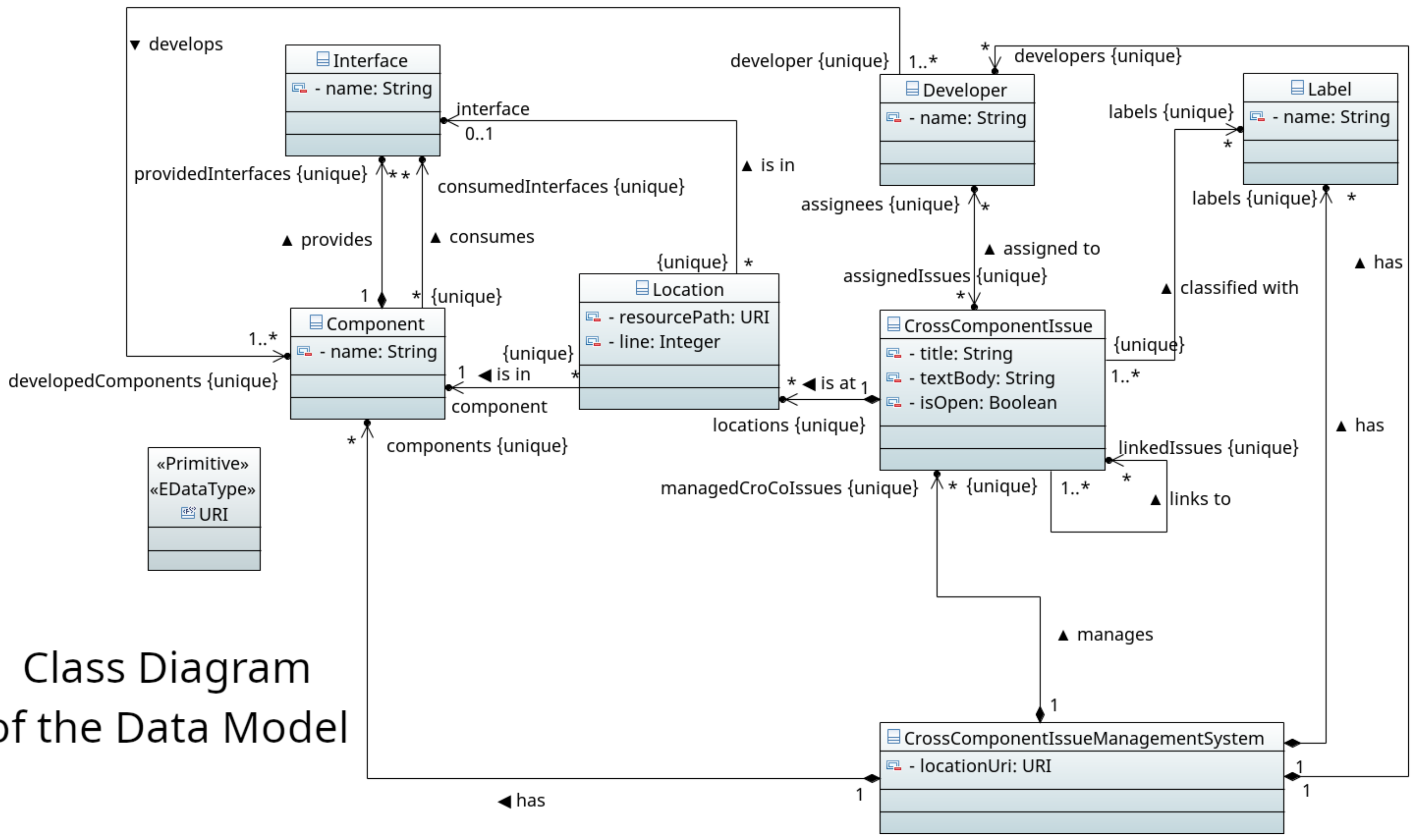
A UML Documents

As the UML documents mentioned in Section 2.1.3 and Section 4.3 can be hard to read there because they are scaled to fit into the text, they are included here again each on a full page.



Domain Meta Model


«component»
Gropius-Backend
Gropius-Api
«use»
GraphQL Api
Datamodel
Generted with
Eclipse EMF.
gropius.ei
«component»
data
«component»
api-binding
«component»
ui
«component»
logic
Manual Implementation of
EMF Resource + ResourceFactory
Generated with Eclipse Parsley
and manually implemented.
Component
Diagram
Currently: Filter for
Issues
Manually
implemented.



Class Diagram of the Data Model

B Artifacts for the expert review

For the expert review various artifacts were used. Those are attached here.

B.1 Gropius EI Installation Guide

The installation guide used for the expert review has been uploaded to `https://github.com/ccims/gropius.ei/releases/download/v0.1.0/installation_guide_for_expert_review.pdf`

B.2 Gropius EI Release

The installation guide uses an eclipse update site to install Gropius EI from. The version of this update site, that was used during the expert review has been preserved in the following archive: `https://github.com/ccims/gropius.ei/releases/download/v0.1.0/gropius.ei_release_for_expert_review.tar.gz`

B.3 Gropius EI Scenario Explanation and Tasks

The explanation of the scenario also including the tasks for the developers has been uploaded to `https://github.com/ccims/gropius.ei/releases/download/v0.1.0/explanation_of_scenario_for_expert_review.pdf`

B.4 Gropius EI Scenario

The eclipse workspace for the scenario is contained in the following archive: `https://github.com/ccims/gropius.ei/releases/download/v0.1.0/scenario_for_expert_review.tar.gz`