Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik

AWAD KARIM
BACHELOR THESIS

AN ACCOUNTING TOOL FOR INNER SOURCE CONTRIBUTIONS

Submitted on 18 December 2018

Supervisor: Prof. Dr. Dirk Riehle, M.B.A.; Maximilian Capraro M.Sc.
Professur für Open-Source-Software
Department Informatik, Technische Fakultät
Friedrich-Alexander-Universität Erlangen-Nürnberg

Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 18 December 2018

License

Erlangen, 18 December 2018

Abstract

"Inner source (IS) is the use of open source software development (SD) practices and the establishment of an open source-like culture within an organization." (Capraro & Riehle, 2016)

However, it is currently impossible for an individual manager of an organization to account for the IS contributions of his or her subordinates, as IS collaboration has never been modeled from an economic perspective before.

To support organizations in utilizing IS, prior work has developed a Collaboration Management Suite (CMSuite). One component of CMSuite is the accountancy tool, which allows the user to view IS contributions and to apply filters on the data to facilitate proper accounting.

Within this thesis, various shortcomings of the current accountancy tool are identified and sensible requirements are elicited. Furthermore, the resulting design and implementation decisions are being discussed. Finally, the fulfillment of the requirements is evaluated and possible further improvements are suggested.

As a result of this thesis, managers in an IS context can use CMSuite to understand how their resources are spend on IS collaboration.

Contents

1 Introduction

"Inner Source (IS) is the use of open source software development practices and the establishment of an open source-like culture within organizations." (Capraro & Riehle, 2016)

The open source approach is a programming concept that became famous due to the success of showcase projects like the Linux Operating System, the Apache web server or the Emacs Text Editor (Dinkelacker, Garg, Miller & Nelson, 2002). As the name indicates, the central part of the open source approach is universal access to all development artifacts. Developers can consequently inspect the artifact and submit contributions (Stol, Avgeriou, Bahar, Lucas & Fitzgerald, 2014). Discussions about the code are held in public forums like newsgroups or mailing lists and are thus thoroughly documented and transparent to all participants (Dinkelacker et al., 2002).

In contrast to the traditional software development process, which can be compared to building a cathedral, where individuals or a small group of people work on a project in isolation and eventually publish a finished product, the open source approach is more like a bazaar, less structured, more transparent and with shorter release cycles. (Raymond, 2000)

The inner source approach can be described as open source, behind the firewall of a company (Dinkelacker et al., 2002). Moreover, to stick with the analogy from before, it can be expressed as a bazaar within a corporate cathedral (Wesselius, 2008). Using inner source projects enables corporations to embrace multiple benefits of the open source approach, while still maintaining the code proprietary.

Inner source is expected to bring a variety of benefits to companies using it: First, due to the openly accessible code base, inner source encourages the reuse of existing components, hence advancing the modularity of the project (Dinkelacker et al., 2002). Also, the overall quality is being raised, because "[g]iven a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone" (Raymond, 2000) or simpler: "Given enough eyeballs, all bugs are shallow" (Raymond, 2000). Extending the programmer base also means profiting from the accumulated know-how, because the whole organization can then contribute their expertise instead of merely relying on one team or department (Riehle et al., 2009). The bigger community also leads to quicker development, considering the increased amount of active developers (Wesselius, 2008). Last but not least developers can familiarize themselves with various projects and tools used in the company, therefore the mobility of the employees increases, as they do not have to face learning curves as steep as before when being reallocated (Dinkelacker et al., 2002).

The advantages of inner source are diverse, however, having a bazaar within a corporate cathedral, leads to new difficulties. Many structures of classical corporations are not adapted to the new concept. Currently, individual managers cannot account for IS contributions of their subordinates, as IS has never been modeled from an economical point of view before. Therefore, managers often fear a loss of resources and also of their influence over subordinates (Capraro & Riehle, 2016). One tool that supports organizations to overcome these difficulties is the Accountancy Tool contained in the Collaboration Management Suite, currently being developed at the Open Source Research Group of the Friedrich-Alexander-University Erlangen-Nuremberg. This thesis' purpose is to improve said accountancy tool, in order to grant prospective users a more intuitive experience with new features to facilitate their work.

In detail, this thesis contributes the following:

1. A detailed discussion of the shortcomings and requirements in the prior implementation of CMSuite's accountancy tool.
2. A report and discussion of the design and implementation details of our elicited improvements.
3. An evaluation of the performed implementations and a suggestion for future work.

Chapter 2 introduces the conceptual basics of the tool. In chapter 3 we will describe what we planned to achieve with this work, how we approached the requirement elicitation process and what requirements we eventually worked out. We will also determine a suitable evaluation scheme for said requirements. In chapter 4, we describe the architecture and design decisions we made when modeling our work. Afterward, in chapter 5, we will discuss selected implementation details, before we go on to evaluate our work regarding our elicited requirements, using our defined evaluation scheme in chapter 6. Finally, we conclude the thesis with a prospect on ideas of further improvements of the accountancy tool in chapter 7.

2 Conceptual Basics

2.1 Collaboration Management Suite

The Collaboration Management Suite (CMSuite) is a set of software tools and components, that is currently under development by the Open Source Research Group of the Friedrich-Alexander-University Erlangen-Nuremberg. CMSuite aims at supporting organizations in successfully utilizing IS. On that account it features the following components:

- Patch Flow Crawler:

  A tool that crawls through inner source repositories to measure the patch flow and to create a data set of the repository that can then be used by other components of CMSuite. The Patch Flow Crawler can be extended with plugins to make it compatible with multiple repository hosting services.

- Dashboard:

  The Dashboard uses the data collected by the Patch Flow Crawler to display custom metrics. Users can define transformations that are persisted in CMSuite's database, and that can subsequently be applied to the collected data.

- Taxation Tool:

  The Taxation Tool is still in an early state of development and is planned to enable users to account for transfer pricing.

- Accountancy Tool:

  The Accountancy Tool accounts for and visualizes IS contributions. It permits managers to gain an overview over the patch flow within their organization. The Accountancy Tool will be explained in more detail in the following section.

2.2 Accountancy Tool

2.2.1 Resource Events Agents Model

The Resources Events Agents Model (REA) is an approach to model accountancy created by McCarthy in 1982. It aims to provide a more generalized model, that can be used by accountants, as well as non-accountants. With this, it is possible to have one central data model for a company. It eliminates the need for maintaining multiple redundant data sets for various use cases, as conventional methods had used. REA achieves this centrality by keeping the model granularity fine. Traditional approaches often had an aggregation level that was set too high, which made it impossible to use for some purposes. A lower level allows users to aggregate by themselves, to adapt the data output to their own needs. (McCarthy, 1982)

McCarthy used Chen's Entity-Relationship model for databases. To model the entities and relationships, it uses the following components:

- Economic resources

  Economic resources are "objects that (1) are scarce and have utility and (2) are under the control of an enterprise" (Ijiri, 1975).

- Economic agents

  Economic agents are persons or agencies who participate in economic events or who are responsible for their subordinates' participation (McCarthy, 1982).

- Economic events

  Economic events are "a class of phenomena which reflect changes in scarce means [economic resources] resulting from production, exchange, consumption and distribution" (Yu, 1976). Economic events follow the concept of duality. Every resource increment event should have a corresponding resource decrement event (Ijiri, 1975). The two events must be from two different event entity sets. One must be transferring out, while the other event must be transferring in (McCarthy, 1982).

- Economic units

  Economic units are a subset of economic agents. They aggregate multiple agents while acting as an economic agent themselves. (McCarthy, 1982)

Stock-Flow relationships connect the individual components. Economic resources are "stocks of goods, services, and claims at a particular time" (McCarthy, 1982), while economic events model the flow of said items over a period of time. McCarthy also emphasizes that participating parties can be grouped into parties that operate inside or outside the company. (McCarthy, 1982)

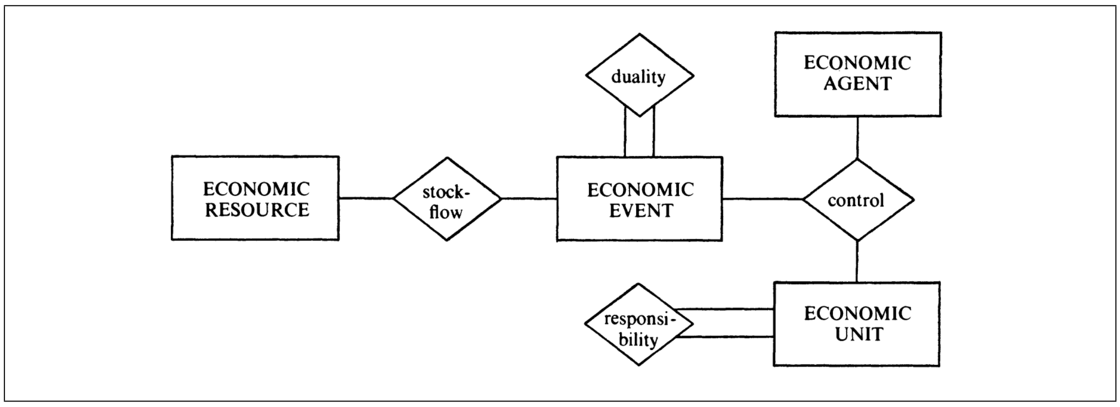
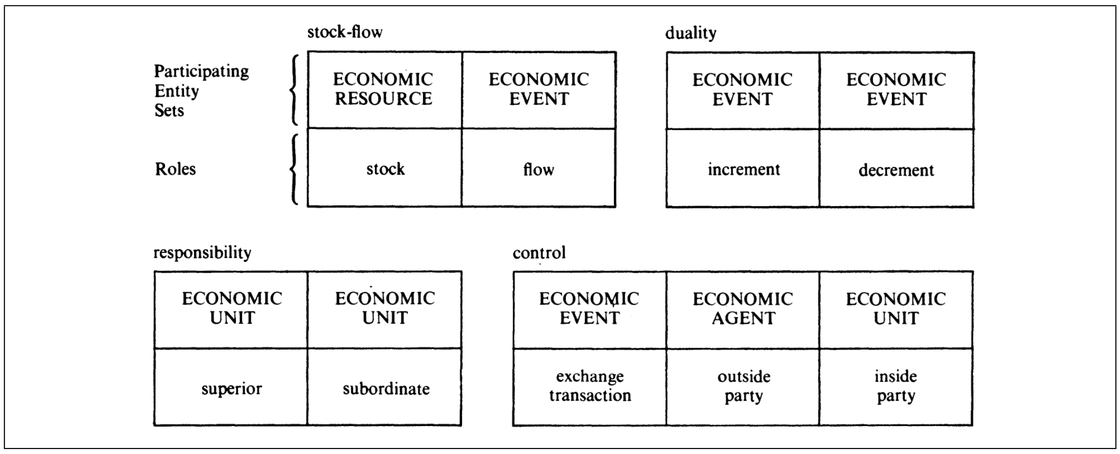


Figure 2.1: Entities and relationships (McCarthy, 1982)



stock-flow

Participating Entity Sets	ECONOMIC RESOURCE	ECONOMIC EVENT
Roles	stock	flow

duality

ECONOMIC EVENT	ECONOMIC EVENT
increment	decrement

responsibility

ECONOMIC UNIT	ECONOMIC UNIT
superior	subordinate

control

ECONOMIC EVENT	ECONOMIC AGENT	ECONOMIC UNIT
exchange transaction	outside party	inside party

Figure 2.2: Role declarations (McCarthy, 1982)

2.2.2 Domain Model

The domain model of the accountancy tool was created using the REA approach.

Economic resources

In our domain, economic resources are code contributions, feature requests, bug reports, code reviews, and discussions. However, before this thesis, the accountancy tool only focused on one type of code contributions, namely patches. Capraro et al. define code contributions as “any code changes performed on a software component" (Capraro, Dorner & Riehle, 2018). A code contribution across organizational boundaries is called a patch. (Capraro et al., 2018).

Economic events

The central aim of the accountancy tool is to monitor all events, where an inner source project functions as an economic agent. Since pre-existing domain only focused on patches, economic events were the acts of providing and receiving patches. In this model, these events were called Patch Contributions (decrement) and Patch Acceptances (increment), and they depicted the duality of relationships, that is an essential part of the REA model.

The flow of patches across organizational boundaries within a company is called patch-flow (Capraro et al., 2018). This patch-flow is critical information for accountancy purposes and thus builds the foundation of the Event Journal.

Economic agents

An economic agent is an entity that participates in an economic event (McCarthy, 1982). In the context of this thesis, economic agents can be persons, organizational elements or inner source projects (ISP).

- Persons shall be defined as members of the observed organization.
- An inner source project is "a software project with the goal to develop and maintain inner source software" (Capraro & Riehle, 2016).
- Organizational elements are economic units that contain other economic agents. As a deduction, organizational elements implement the composite design pattern: an organizational element can have multiple child elements.

Since persons and ISPs have to be assigned to organizational elements, a tree can be spanned, that models the structure of the company. However, said structure can be multidimensional, as there can be more than one way to view it. For those instances, the model also supports organizational dimensions, that are assigned to the links between economic agents. As a result, an inner source project can, for example, be assigned to different organizational elements for different dimensions. Therefore every organizational dimension can have a unique organization tree.

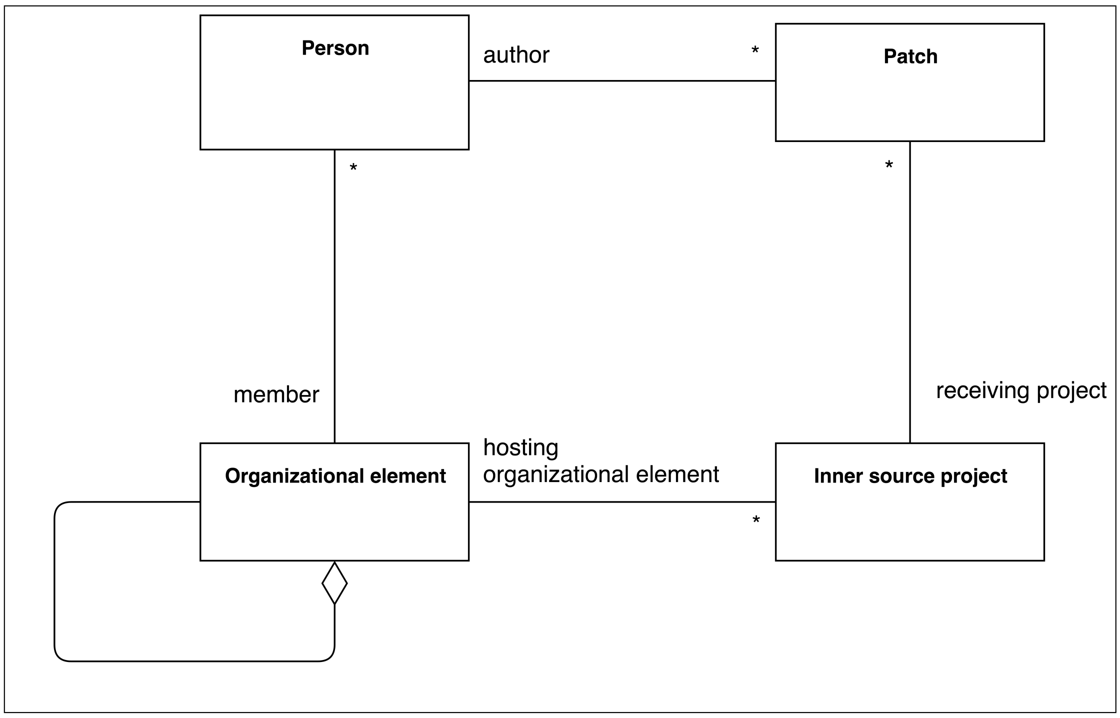


Figure 2.3: A simple patch-flow data model (Capraro, Dorner & Riehle, 2018)

2.2.3 Pre-Existing Features

At the beginning of this thesis, CMSuite's essential Accountancy Journal showed patch related events for an organizational element. More specifically it showed all external patches with the involvement of current subordinate Economic Agents.

To allow a selection of the context organizational element, the accountancy tool featured a sidebar with a tree that visualized the hierarchy of the economic agents in one organizational dimension. Using the drop-down menu at the bottom of the sidebar, a user could select the organizational dimension of interest. This tree menu is a common element of CMSuite as it also appears in other components, e.g., the dashboard.

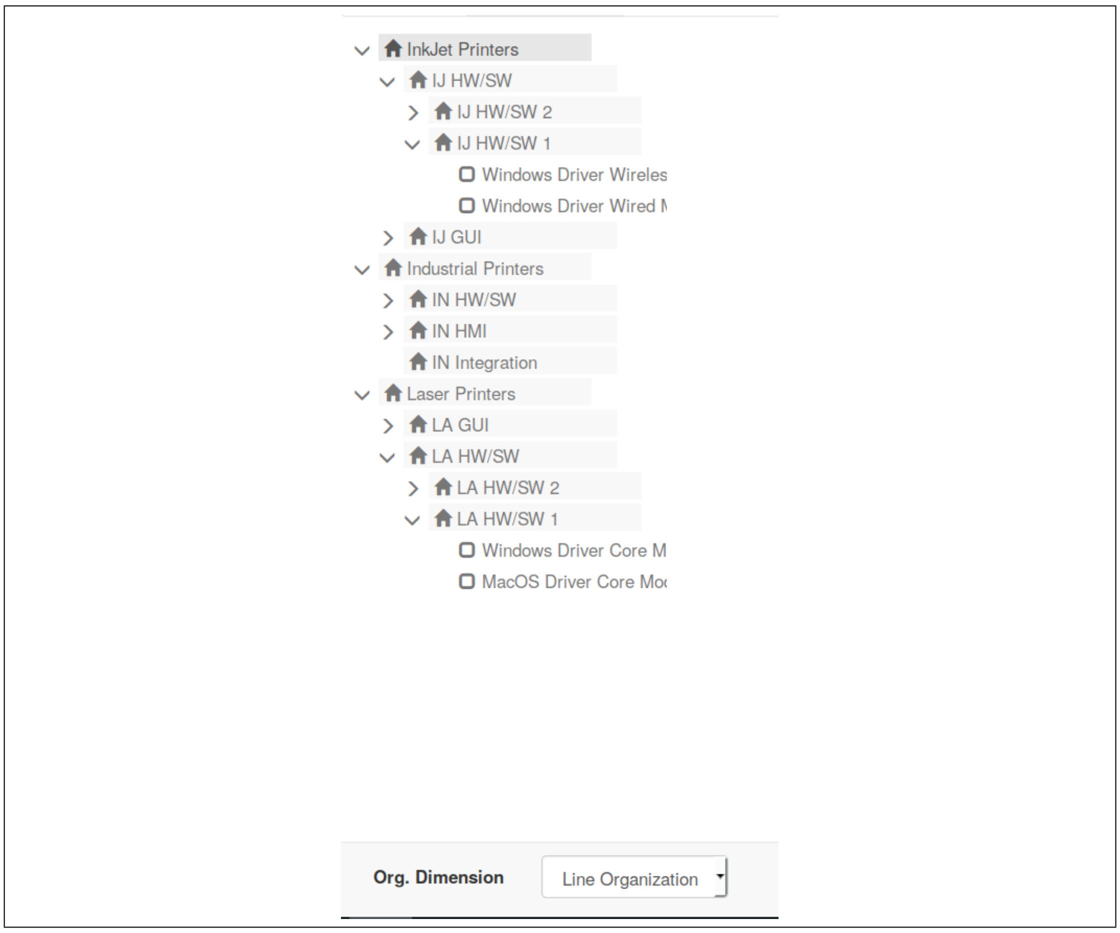
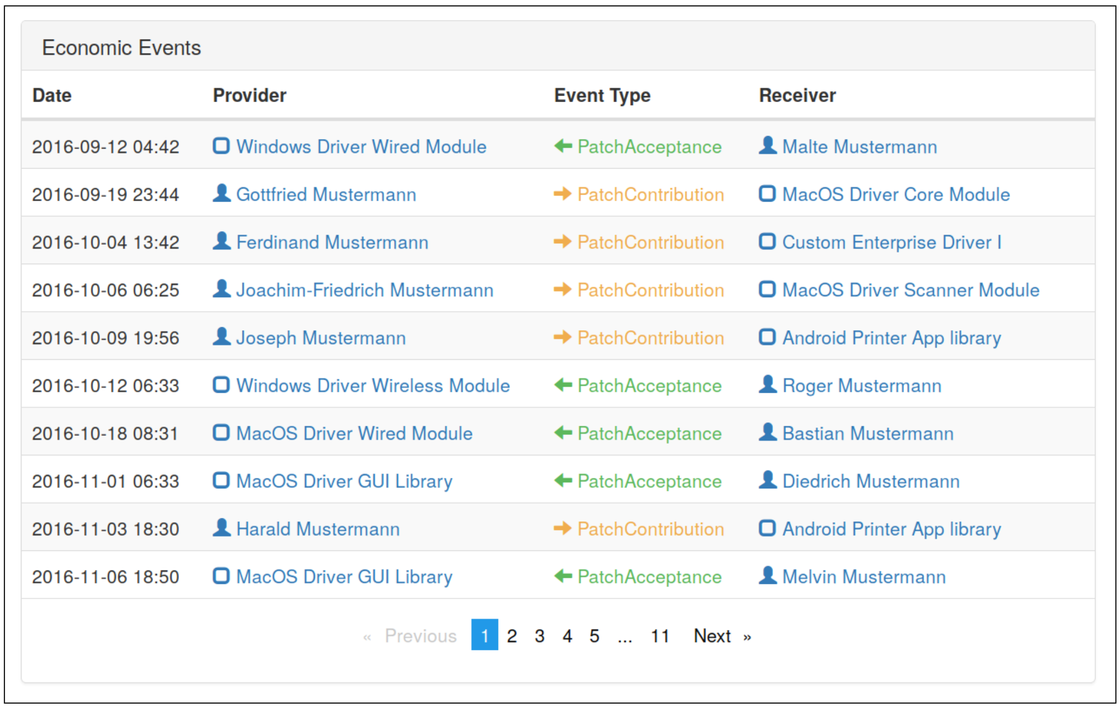


Figure 2.4: The tree menu showing an example organization tree

When the user selected a context organizational element, the event journal for that element appeared. The event journal was a table labeled as “Economic Events” that consisted of “Date,” “Provider,” “Event Type,” and “Receiver” columns. The rows consisted of economic events, that had been requested from the database. The provider and receiver were formatted as hyperlinks. If the agent was an ISP, the hyperlink led to a page that merely stated that accounting for a project was to follow. If it was a person, the link was disabled. The links were only useful when the economic agent was an organizational element. In that case, the user was brought directly to the overview of that element, when he pressed the hyperlink.

The accountancy tool sorted all entries chronologically. If two entries shared the same commit date, their order did not follow a particular schema. Additionally, the event list featured a pagination feature. All entries were automatically assigned to pages. One page fitted ten entries.



Economic Events

Date	Provider	Event Type	Receiver
2016-09-12 04:42	Windows Driver Wired Module	PatchAcceptance	Malte Mustermann
2016-09-19 23:44	Gottfried Mustermann	PatchContribution	MacOS Driver Core Module
2016-10-04 13:42	Ferdinand Mustermann	PatchContribution	Custom Enterprise Driver I
2016-10-06 06:25	Joachim-Friedrich Mustermann	PatchContribution	MacOS Driver Scanner Module
2016-10-09 19:56	Joseph Mustermann	PatchContribution	Android Printer App library
2016-10-12 06:33	Windows Driver Wireless Module	PatchAcceptance	Roger Mustermann
2016-10-18 08:31	MacOS Driver Wired Module	PatchAcceptance	Bastian Mustermann
2016-11-01 06:33	MacOS Driver GUI Library	PatchAcceptance	Diedrich Mustermann
2016-11-03 18:30	Harald Mustermann	PatchContribution	Android Printer App library
2016-11-06 18:50	MacOS Driver GUI Library	PatchAcceptance	Melvin Mustermann

Figure 2.5: The pre-existing event journal displaying demo patches

The user interface also offered a menu on the side to adapt the analysis parameters, particularly to define the wanted time granularity and also the desired organizational element granularity. The drop-down menu for time granularity contained options to filter by day, month, year and to ignore all dates. The organizational element granularity drop-down showed a list of all available organizational element types. After selecting the wanted granularities and submitting the form by pressing the refresh button, the economic events table reloaded its events. When the user had selected a time granularity, the journal displayed the events, aggregated by that granularity. If he selected an organizational element granularity, the event provider and receiver were replaced by an ancestor organizational element that was of the selected type. Subsequently, equal events were grouped.

When multiple events were aggregated into one, a resource gauge was shown, that visualized the number of aggregated events and their direction.

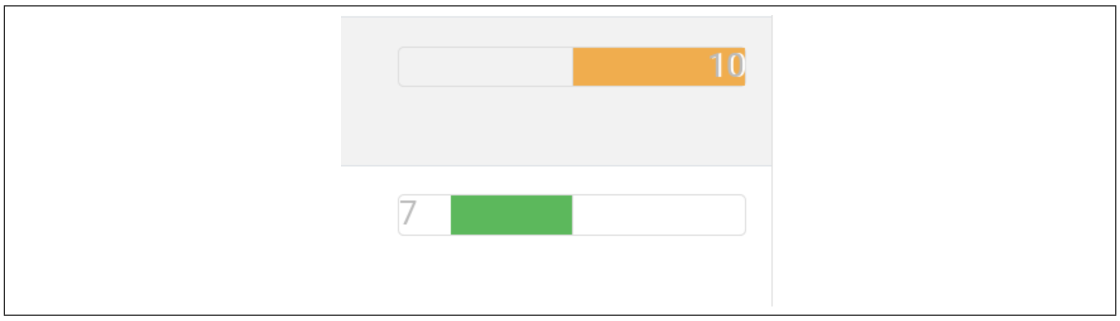


Figure 2.6: Resource gauges that visualize aggregated events

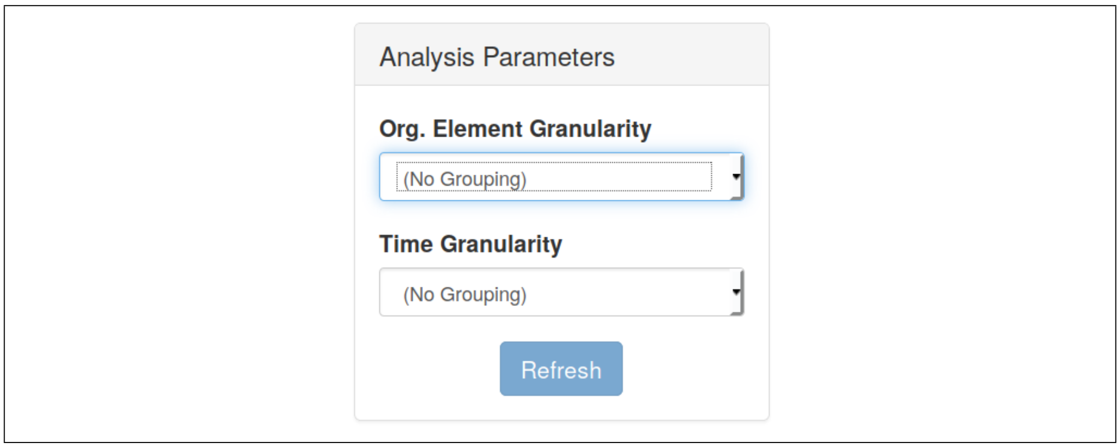
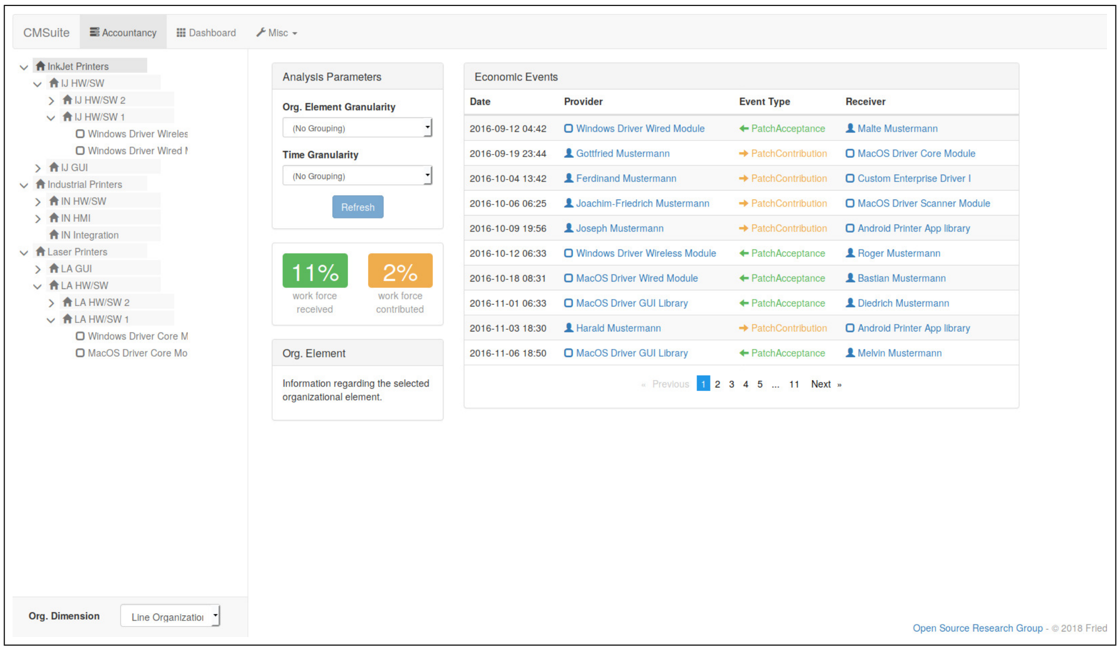


Figure 2.7: The pre-existing event journal menu



Information regarding the selected organizational element.

Date	Provider	Event Type	Receiver
2016-09-12 04:42	Windows Driver Wired Module	PatchAcceptance	Malte Mustermann
2016-09-19 23:44	Gottfried Mustermann	PatchContribution	MacOS Driver Core Module
2016-10-04 13:42	Ferdinand Mustermann	PatchContribution	Custom Enterprise Driver I
2016-10-06 06:25	Joachim-Friedrich Mustermann	PatchContribution	MacOS Driver Scanner Module
2016-10-09 19:56	Joseph Mustermann	PatchContribution	Android Printer App library
2016-10-12 06:33	Windows Driver Wireless Module	PatchAcceptance	Roger Mustermann
2016-10-18 08:31	MacOS Driver Wired Module	PatchAcceptance	Bastian Mustermann
2016-11-01 06:33	MacOS Driver GUI Library	PatchAcceptance	Diedrich Mustermann
2016-11-03 18:30	Harald Mustermann	PatchContribution	Android Printer App library
2016-11-06 18:50	MacOS Driver GUI Library	PatchAcceptance	Melvin Mustermann

« Previous 1 2 3 4 5 ... 11 Next »

Figure 2.8: The accountancy tool showing an event journal

3 Requirements

3.1 Purpose of the Artifact

Prior to this thesis, CMSuite already featured an essential accountancy tool. However, it did have significant shortcomings. It had not been adapted to some data model changes, so features like the organizational element filter did not work correctly in many cases. Furthermore, its functions were minimal. Apart from a time and org element granularity filter, it did not have any options to adapt the event journal to personal needs. The tool also needed some overall fine-tuning.

The purpose of this thesis is to improve and extend the functionality of the accountancy tool, in order to provide a better user experience and overall user satisfaction.

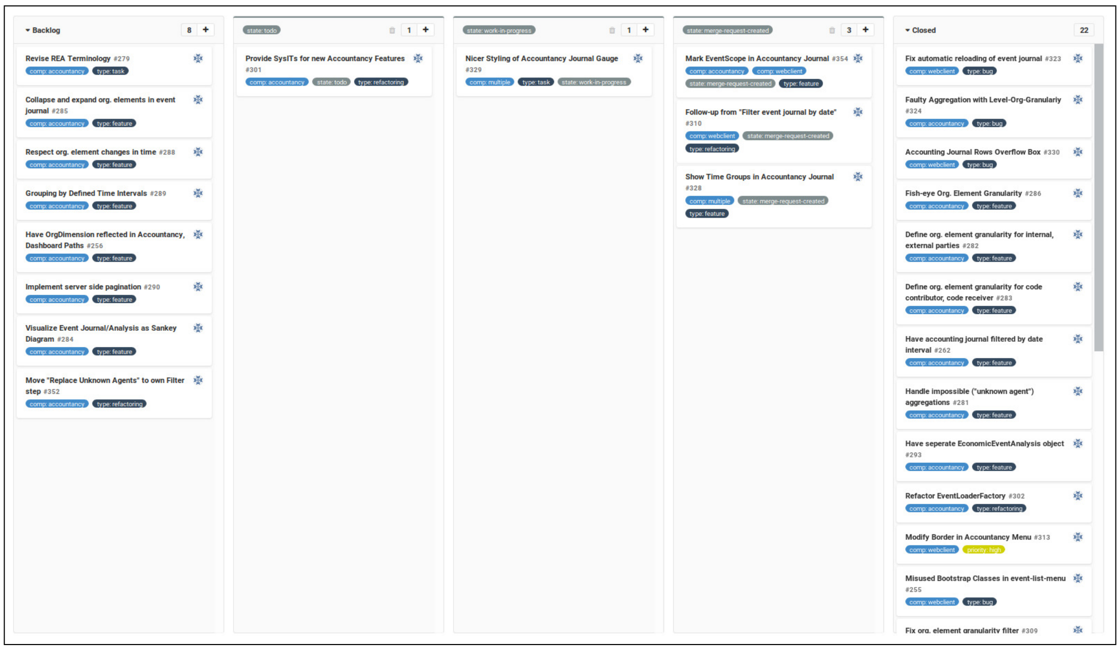
To achieve this, we started with locating problems with the pre-existing implementation. Next, we formulated fitting requirements to fix the issues we found. Eventually, we designed and implemented changes, based on our requirements.

3.2 Development Approach

Incipiently, we needed to identify the shortcomings of the pre-existing event journal. To do so, we started with an intuitive method: We ran CMSuite with production data and analyzed the user experience and features from the viewpoint of managers of an organization, the prospective user group of the accountancy tool. The result of this analysis was a list of possible UI improvements and potentially useful additional features. We also used this initial testing to locate errors, that we also thoroughly documented afterward.

Next, we inspected the implementation of the tool. We focused on determining inconsistencies with the data model, potential refactorings, to improve code quality and deficiencies in code, that could lead to runtime errors. Based on our outcomes, we invented an example data set. With this data set, we could model edge cases, to accurately illustrate malfunctions of the event journal and to allow us to get a better understanding of the data model.

All previously described steps led to a list of tasks, we could perform, to improve the code quality, features and the user experience of the accountancy tool. To organize these tasks, we created issues for them on CMSuite's Gitlab project and assigned each of them to one of the following categories: "Bug," "Feature," "Refactoring" or "Task." We also utilized Gitlabs Board feature, to sort the issues similar to a Kanban Board. The Board featured lists for "Backlog," "Todo," "Work in Progress," "Merge Request Created" and "Closed." These lists enabled us to keep an overview of the current project state and to track our progress.



Backlog 8 +

- Revise REA Terminology #279 — comp: accountancy, type: task
- Collapse and expand org. elements in event journal #285 — comp: accountancy, type: feature
- Respect org. element changes in time #288 — comp: accountancy, type: feature
- Grouping by Defined Time Intervals #289 — comp: accountancy, type: feature
- Have OrgDimension reflected in Accountancy, Dashboard Paths #256 — comp: accountancy, type: feature
- Implement server side pagination #290 — comp: accountancy, type: feature
- Visualize Event Journal/Analysis as Sankey Diagram #284 — comp: accountancy, type: feature
- Move "Replace Unknown Agents" to own Filter step #352 — comp: accountancy, type: refactoring

state: todo 1 +

- Provide SysITs for new Accountancy Features #301 — comp: accountancy, state: todo, type: refactoring

state: work-in-progress 1 +

- Nicer Styling of Accountancy Journal Gauge #329 — comp: multiple, type: task, state: work-in-progress

state: merge-request-created 3 +

- Mark EventScope in Accountancy Journal #354 — comp: accountancy, comp: webclient, state: merge-request-created, type: feature
- Follow-up from "Filter event journal by date" #310 — comp: webclient, state: merge-request-created, type: refactoring
- Show Time Groups in Accountancy Journal #328 — comp: multiple, state: merge-request-created, type: feature

Closed 22

- Fix automatic reloading of event journal #323 — comp: webclient, type: bug
- Faulty Aggregation with Level-Org-Granulariy #324 — comp: accountancy, type: bug
- Accounting Journal Rows Overflow Box #330 — comp: webclient, type: bug
- Fish-eye Org. Element Granularity #286 — comp: accountancy, type: feature
- Define org. element granularity for internal, external parties #282 — comp: accountancy, type: feature
- Define org. element granularity for code contributor, code receiver #283 — comp: accountancy, type: feature
- Have accounting journal filtered by date interval #262 — comp: accountancy, type: feature
- Handle impossible ("unknown agent") aggregations #281 — comp: accountancy, type: feature
- Have seperate EconomicEventAnalysis object #293 — comp: accountancy, type: feature
- Refactor EventLoaderFactory #302 — comp: accountancy, type: refactoring
- Modify Border in Accountancy Menu #313 — comp: webclient, priority: high
- Misused Bootstrap Classes in event-list-menu #255 — comp: webclient, type: bug
- Fix org. element granularity filter #309

Figure 3.1: An example of the github issue board

Throughout the whole project, we maintained the board and continuously extended the issue list based on new insights we gained.

Very quickly it became adamant, that the number of issues, we created exceeded the scope of a bachelors thesis, so we prioritized them and decided to tackle the most important ones first.

3.3 Identified Problems

In this section, we will describe the problems we found with the pre-existing accountancy tool. This does not only include bugs or errors but also missing features or inconsistencies in the used domain model.

Limitations of the event journal

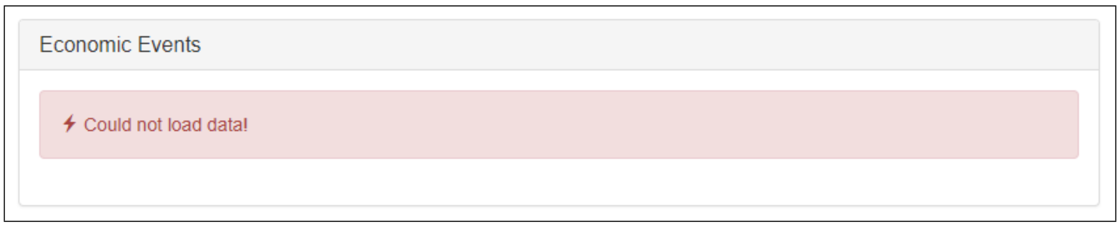
The event journal reliably showed all patches where a subordinate economic agent of the context organizational element participated. However, a detailed analysis of its functionality revealed some shortcomings.

1. Only context organizational elements were supported. An event journal for ISPs was foreseen, but not implemented. Analyzing the patch-flow towards an ISP can be of great interest for accountancy use cases.
2. The pre-existing journal only listed patches. The possibility to also include internal code contributions was missing entirely, even though they are also a crucial aspect of inner source collaboration.
3. The journal displayed all events for a context organizational element, independent of the event date. This means that if we have, e.g., five years of inner source data saved, the journal always displays all events that had happened in that time, even though most of these events are not of interest anymore.
4. Displayed events had no defined sorting order. Coincidentally, the events were automatically sorted by their date, but this order was nowhere explicitly defined. This became a problem when we set the time granularity to "Ignore all dates," the sorting appeared to be completely arbitrary.

Errors of the organizational element granularity filter

When we tested the accountancy tool with production data, we quickly realized that the organizational element granularity filter did not work as wanted. This was exposed the following way:

1. The drop-down menu showed too many organizational element types. It even showed types that were not available in the currently selected organizational dimension. When we picked those wrongly added types, the accountancy tool showed an error message that merely stated: "Could not load data."
2. This error message was even shown when we had selected an available organizational element type. This happened with both the production dataset and the demo dataset.


Economic Events

Could not load data!

Figure 3.2: The error message, the event journal showed, when the organizational element type filter was used

The code analysis quickly revealed the problem: The organizational element granularity filter did not take organizational dimensions into account. The filter showed all types because it did not differentiate between dimensions and it did thus not sort out unavailable types. The error message we received, when we applied the filter was also related to the missing support of multiple dimensions.

Limitations of the organizational element granularity filter

Users of the accountancy tool profit from filtering by the organizational element type. However, sometimes the current filter seems insufficient. We found the following demands, a user could make on the filter, that it did not meet:

1. Filter by organizational level

   Generally, organizational trees for dimensions are not always normalized. Subtrees vary in size and persons, or inner source projects are not necessarily assigned to leaf organizational elements. Therefore it is of interest for a user to show the agents based on their level in the hierarchy tree.

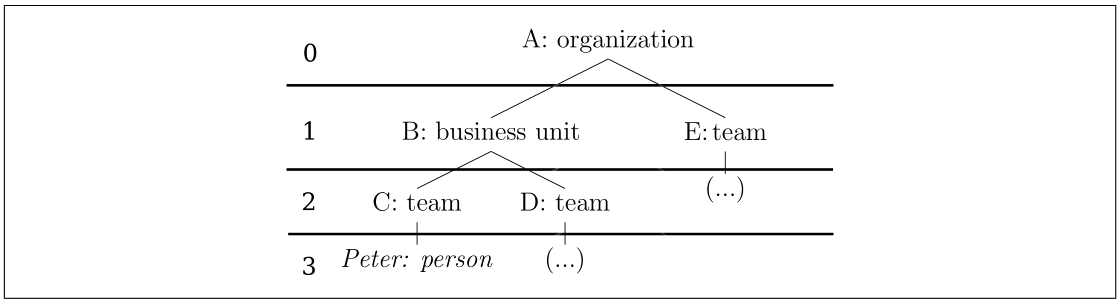


Figure 3.3: An example of an organization tree with indicated levels

2. Only filter Contributors/Receivers

   The filter used only to filter all economic agents. This all-or-nothing approach restricts users in some cases. Sometimes, the user might want to apply different filters to contributor agents than to receiver agents.

3. Only filter Inside/Outside agents

   Similarly to (2), the user might need to filter inside and outside agents differently. Usually, managers are interested in fine granular inside agents, while they mostly do not care about the details of the outside agents, as they are not their subordinates. Whether an agent is an inside or outside agent is defined in relation to the selected context.

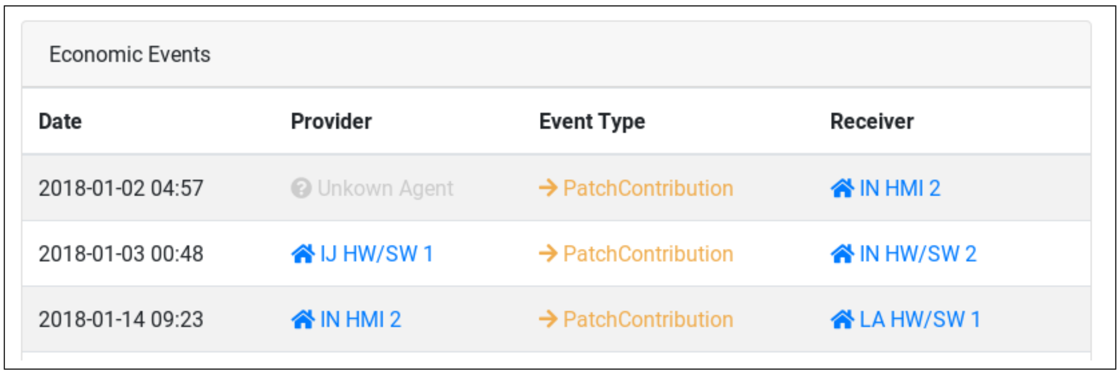
4. Fisheye view

   The fisheye view is a natural way to look at the company structure from the viewpoint of a manager. It hides information the manager does not care about, to reduce the information overload. The view shows all close agents detailed, while far away agents are shown more coarse-grained. A fisheye filter replaces the participating agents of an event by ancestor elements of these agents that are in one of the following relations with the context element:

   - child
   - sibling
   - uncle
   - grand-uncle
   - (...)
   - (n*grand)-uncles

5. Replace unknown agents

   It can happen, that an organizational element of the searched type or level does not exist. The event journal used to display those irreplaceable agents as “Unknown Agents.” This is correct behavior, though in some cases more information about these agents could be helpful. The accountancy tool lacks an option to replace unknown agents with practical information about them, to grant users insight on what exactly hides behind them.


Economic Events

Date	Provider	Event Type	Receiver
2018-01-02 04:57	Unkown Agent	PatchContribution	IN HMI 2
2018-01-03 00:48	IJ HW/SW 1	PatchContribution	IN HW/SW 2
2018-01-14 09:23	IN HMI 2	PatchContribution	LA HW/SW 1

Figure 3.4: An example of an unknown agent

Domain model inconsistencies

From the current model, we observed the following limitations:

- Internal and external vs. inside and outside agent

  Currently, we sometimes label agents as internal or external agents (e.g., in the organization element filter) REA, however, uses the terms “Inside Agent” and “Outside Agent.”

- Receiver and provider terminology

  The terminology “receiver” and “provider” of an economic event tends to be confusing. For a patch acceptance event, the receiver of the event is not the receiver of the contribution.

- Patch vs. code contribution

  In the pre-existing domain model, everything was modeled as a patch, even internal code contributions. In chapter 2 we defined patches as code contributions, that are made by a developer, who is external to a project. So, technically internal code contributions cannot be called patches.

- Inner source projects are not agents

  Currently, the inner source projects are modeled as economic agents. Technically they, (or better: the inner source components) are economic resources, as they are in the control of the organization and usually also scarce and with utility. Thus they fit the REA definition of economic resources. One could, however, argue that inner source project communities are in the position to accept patches and act as an agent. Ideally, the components (resources) and project communities (agents) are modeled as separate elements.

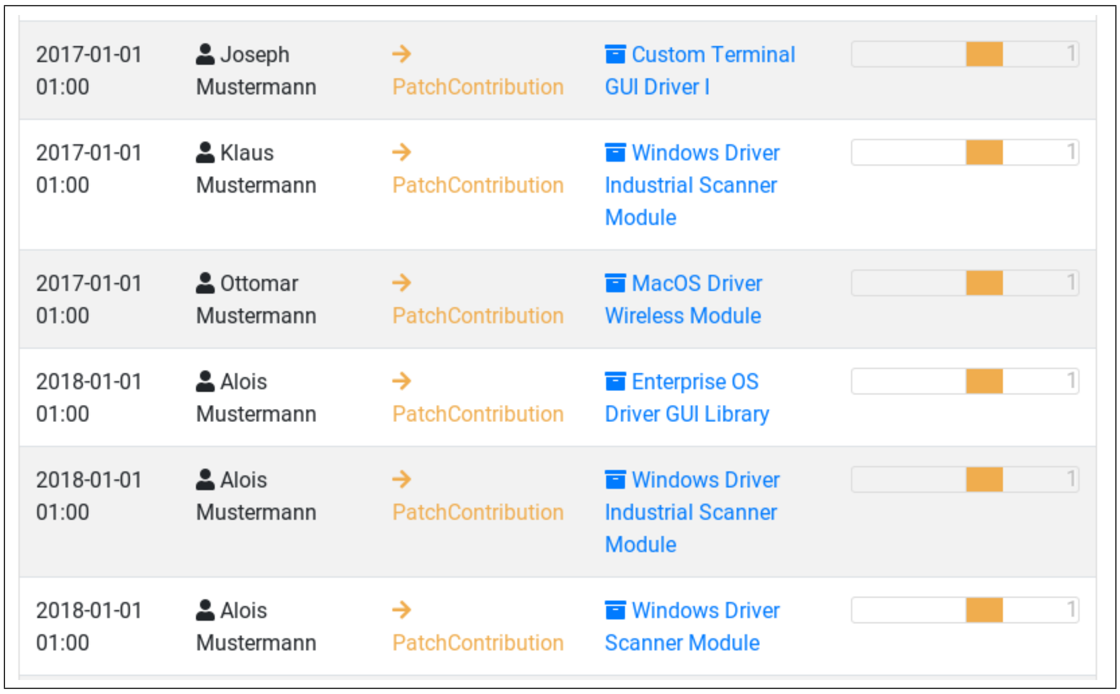
- No consequently modeled concept of duality

REA typically models economic events as exchanges (give and take) We currently do not have such a thing (contributions are just done) We used acceptance events to partly model this duality. However, this is a simplification. In the future, the model might be used for more complex duality relationships.

Visual deficiencies

While using the Accountancy Tool, we also located some unpleasant details of its visuals:

1. The journal showed dates in a yy-MM-dd hh-mm format. If we show all events unfiltered, this format provides all the information we need. However, if we apply a time granularity filter, the format does not adapt to the chosen granularities and thus shows unnecessary information.



2017-01-01 01:00	Joseph Mustermann	→ PatchContribution	Custom Terminal GUI Driver I	1
2017-01-01 01:00	Klaus Mustermann	→ PatchContribution	Windows Driver Industrial Scanner Module	1
2017-01-01 01:00	Ottomar Mustermann	→ PatchContribution	MacOS Driver Wireless Module	1
2018-01-01 01:00	Alois Mustermann	→ PatchContribution	Enterprise OS Driver GUI Library	1
2018-01-01 01:00	Alois Mustermann	→ PatchContribution	Windows Driver Industrial Scanner Module	1
2018-01-01 01:00	Alois Mustermann	→ PatchContribution	Windows Driver Scanner Module	1

Figure 3.5: The event journal, when events were sorted by year

2. The resource gauges that visualized aggregated code contributions looked unpleasant and were also hardly readable at times.

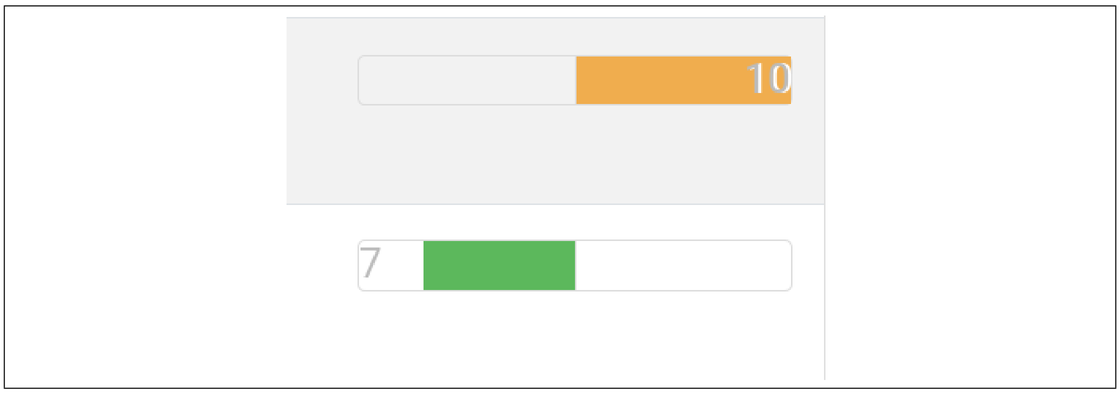
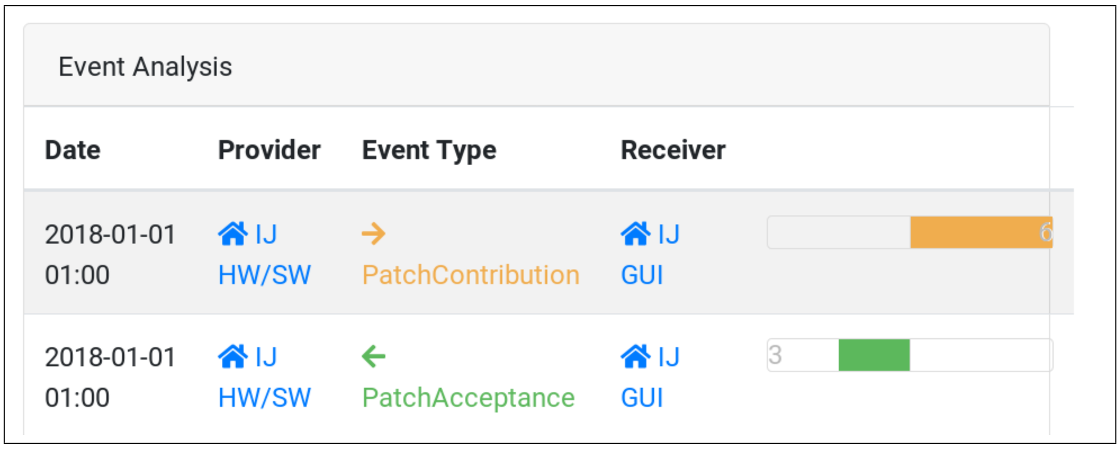


Figure 3.6: The old resource gauge, that visualizes patch contributions/acceptances

3. The content of the event journal table overflowed the table borders when the name of an economic agent was too long or when the browser window was too small.



Event Analysis

Date	Provider	Event Type	Receiver	
2018-01-01 01:00	IJ HW/SW	PatchContribution	IJ GUI	6
2018-01-01 01:00	IJ HW/SW	PatchAcceptance	IJ GUI	3

Figure 3.7: The table content overflowing the table bounds

3.4 Requirements for Artifact

The primary goal of this thesis is to extend the existing software tool to account for and to visualize contributions within an organization. After we have thoroughly analyzed the weaknesses of the existing tool, we will now specify our requirements for this thesis.

3.4.1 Stakeholders

We will define our requirements from the view of the following stakeholders:

- Users of the accountancy tool
  - Managers of an organization unit
  - Managers of an inner source project
- Developers

  Developers who service the accountancy tool or who extend its functionality

3.4.2 Functional Requirements

Event journal for inner source projects

Functional Requirement 1. *As the manager of an inner source project, I want the accountancy tool to include an event journal for inner source projects so that I can keep an overview over who contributes to my project.*

Include internal events

Functional Requirement 2. *As the manager of an organizational unit, I want the accountancy tool's event journal to have an option to also include internal events so that I have an insight into the flow of code contributions within my unit.*

Filter events by date

Functional Requirement 3. *As a user of the accountancy tool, I want the event journal to have the option to filter its events by their date so that I can hide useless information.*

Filter by business year by default

Functional Requirement 4. *As a manager of an organizational unit, I want the event journal to show only the events of one business year by default so that I save time when using the accountancy tool.*

Define an explicit sorting order

Functional Requirement 5. *As a user of the accountancy tool, I want the event journal to sort the output by an explicitly defined order so that I have a better overview of the data.*

Fully support multiple organizational dimensions

Functional Requirement 6. *As a user of the accountancy tool, I want the organization element filters to fully support multiple organizational dimensions so that they are better adapted to my organization's structures.*

Organizational level filter

Functional Requirement 7. *As a user of the accountancy tool, I want the event journal to have an option to show the economic agents, based on their organizational level so I can adapt the event journal's information to my needs.*

Apply filters to internal/external parties

Functional Requirement 8. *As a user of the accountancy tool, I want the event journal to have an option to apply filters to internal and external parties independently so that I can adapt the event journal's information to my needs.*

Apply filters to contributors/receivers

Functional Requirement 9. *As a user of the accountancy tool, I want the event journal to have an option to apply filters to contributing and receiving parties independently so that I can adapt the event journal's information to my needs.*

Fisheye filter

Functional Requirement 10. *As a user of the accountancy tool, I want the event journal to have an option to give me a fisheye view on the events so that I can have a natural overview of economic events.*

Replace unknown agents

Functional Requirement 11. *As a user of the accountancy tool, I want the event journal to have an option to replace unknown agents so that I can see who contributed to projects or what ISP received contributions.*

3.4.3 Nonfunctional requirements

Functionality independent from client implementation

Nonfunctional Requirement 1. *As a developer of the accountancy tool, I want the functionality of the tool to be as independent as possible from the client implementation, so that I can add other clients more easily in the future.*

Consistent coding style

Nonfunctional Requirement 2. *As a developer of the accountancy tool, I want new code to follow the given style guidelines so that all code has a uniform style.*

Appropriate loading times

Nonfunctional Requirement 3. *As a user of the accountancy tool, I want the tool to have fast loading times so that I can finish my work quicker.*

Good visuals

Nonfunctional Requirement 4. *As a user I want the accountancy tool to be visually appealing so that I can have a high-quality user experience.*

Thorough testing

Nonfunctional Requirement 5. *As a developer, I want all the features of the accountancy tool to be extensively tested so that I can find bugs before deploying the code.*

Consistent domain model and terminology

Nonfunctional Requirement 6. *As a stakeholder, I want the accountancy tool to use a consistent domain model and terminology.*

3.5 Evaluation Scheme for Requirements

In order to evaluate our requirements, we will use a scale from one to three. The three levels are color coded and follow the principle of traffic lights. We define the levels as follows:

🟥 Level 1) Not satisfied No requirements are met.
🟨 Level 2) Partly satisfied Only parts of the requirements are met.
🟩 Level 3) Completely satisfied All requirements of the group are met.

Figure 3.8: The three levels, we use for requirement evaluation

We will explain how every requirement was met or discuss why it was only met partly or not at all. Eventually, we will present a conclusion for this thesis, considering all requirements we evaluated.

4 Architecture and Design

In this chapter, we will discuss our overall architecture and design, as well as some design decisions, that we deem interesting to the reader.

We decided to keep the architecture of the accountancy tool consistent with the architecture of the other components of CMSuite. As a result, the accountancy tool is based on a client-server architecture. The client is a web app that is developed with angular 6. In the future, more clients can be added to keep up with current developments or to extend the reach of the tool. The server is developed in Java 8 and communicates with the clients over a REST API.

4.1 Server Side

4.1.1 Frameworks and Systems

Jersey

The server utilizes the Jersey Framework to provide the REST interface, that is used for communication with the clients. Jersey is an open source framework for creating RESTful web services in Java that support common JAX-RS APIs. It also extends those APIs with additional features and utilities to simplify the creation of RESTful web services. ("Jersey", 2018)

PostgreSQL

The server uses PostgreSQL, an open source object-relational database system. PostgreSQL uses and also extends the SQL language. ("PostgreSQL", n.d)

Hibernate

The Object/Relational Mapping (ORM) framework Hibernate is used to persist data with the database. Hibernate implements the Java Persistence API (JPA) and also extends it with its own “native” API. Therefore it can be combined with any environment that supports JPA. (“Hibernate ORM”, n.d)

4.1.2 The REST API

The server of the accountancy tool offers the following REST resources to the clients:

Analysis specifications	baseurl: accountancy/analysisspecifications	
Verb	URL	Description
POST	/	Save an analysisspecification
Organizational elements	baseurl: accountancy/orgelements	
Verb	URL	Description
GET	/id/events	Get all events for an organizational element
Inner source projects	baseurl: accountancy/innersourceprojects	
Verb	URL	Description
GET	/id/events	Get all events for an inner source project
Persons	baseurl: accountancy/persons	
Verb	URL	Description
GET	/id/events	Get all events for a person

Figure 4.1: The accountancy tool’s REST resources

4.1.3 Services

When a client communicates with the server, the Rest Service Application forwards the request to service classes. The accountancy tool has the following Service classes:

- AnalysisSpecificationService
- OrgElementService
- InnerSourceProjectService
- PersonService

Economic event services

The OrgElementService, InnerSourceProjectService, and PersonService classes are used to retrieve economic events, and they implement the Economic Event Service Interface, that consists of the following methods:

```
public List<? extends EconomicEvent> getEvents(Integer _id,
        Integer _orgDimensionId,
        Integer _analysisSpecificationId);

public List<? extends EconomicEvent> getEvents(Integer _id,
        Integer _orgDimensionId);
```

Figure 4.2: The economic event service interface

Since the only difference between those services is the type of the context agent, their overall structure is very similar. So similar that they all share a common abstract base class, the AbstractEconomicEventService class.

The job of the economic event services is to analyze the requests, retrieve the economic events from the database, apply filters and then return the events. All this functionality is implemented in the abstract base class. The extending classes only implement hook methods for the event loader and context agent retrieval.

Analysis specification service

The Analysis Specification Service is used to store Analysis Specification objects on the database. At this point, we will explain first what Analysis Specification objects are and why they are needed.

Initially, the interface of the economic event service classes looked like this:

```
public List<? extends EconomicEvent> getExternalEvents(Integer _id,
        TimeInterval _granularity, Integer _oeTypeId);

public List<? extends EconomicEvent> getExternalEvents(Integer _id);
```

Figure 4.3: The economic event service interface prior to this thesis

Since we only had time granularity and organizational element filters, this was an easy to use signature. However, throughout this work, the number of parameters the user could adapt grew and quickly exceeded the maximum number of method parameters the style guidelines allowed. A parameter object was needed.

Ideally, this parameter object would also be used in the client and could then be transmitted over the REST API.

Hence, the analysis specification object was created. We decided to persist analysis specification objects in the database. This offers the advantage that we could develop a “saved views” feature in the future, that would only have to store the id of an analysis specification object to save view preferences. The analysis specification service is needed to allow clients to save analysis specification objects. It persists the objects and then returns the id of the saved objects.

4.1.4 The Filter Chain Factory

All filters that can be applied in the accountancy tool implement the following interface:

```
public interface EconomicEventFilter {

    public List<? extends EconomicEvent> filter(
            List<? extends EconomicEvent> _events);

}
```

Figure 4.4: The economic event filter interface

Prior to this thesis, only three filter classes existed:

- The date truncation filter that exchanged the dates of economic events with truncated dates
- The organizational element type filter, that swapped the participating agents of an event by an ancestor element of a selected type
- The group filter that grouped equal economic events

These filters were all instantiated in the economic event service class. However, as the functionality of the accountancy tool grew, the filter instantiation became more complex, and we then decided to move it into a separate factory class. The Filter Chain Factory has one public method, shown in Figure 4.5.

```
public List<EconomicEventFilter> getFilterChain(
        AnalysisSpecification _spec,
        OrgElement _contextOrgElement,
        OrgDimension _orgDimension) {

    List<EconomicEventFilter> filters = new ArrayList<>();

    addDateTruncFilter(filters, _spec);
    addOrgElementFilter(filters,
            _contextOrgElement, _orgDimension, _spec);
    addGroupFilter(filters);
    addSortingFilter(filters);

    return filters;
}
```

Figure 4.5: The implementation of the `getFilterChain getFilterChain( AnalysisSpecification _spec, OrgElement _contextOrgElement, OrgDimension _orgDimension)` method

This method instantiates all filter objects based on the analysis specification object it receives and inserts them into a list, that we named filter chain. It then returns this filter chain. The services that use this factory can then traverse the list and apply the filters. This way, the code stays overseeable, even when multiple filters are used.

4.1.5 Organizational Element Finders

Initially, the organizational element type filter traversed the list of events and then replaced the participating agents with ancestors of the specified type. To do so, it instantiated a so-called organizational element finder object, that offered two public methods.

```
OrgElement find(Person _person);

OrgElement find(InnerSourceProject _isp);
```

Figure 4.6: The organizational element finder interface

On method call, the finder object retrieved organizational elements based on the specified type and returned them. Furthermore, it provided caches for the replacement methods to speed up the filter process.

During this thesis, we generalized the filter to an organizational element filter, since we had added multiple similar filter functions. We then designed the structure that is displayed in Figure 4.8.

The finder interface now includes the two finder methods, we mentioned earlier.

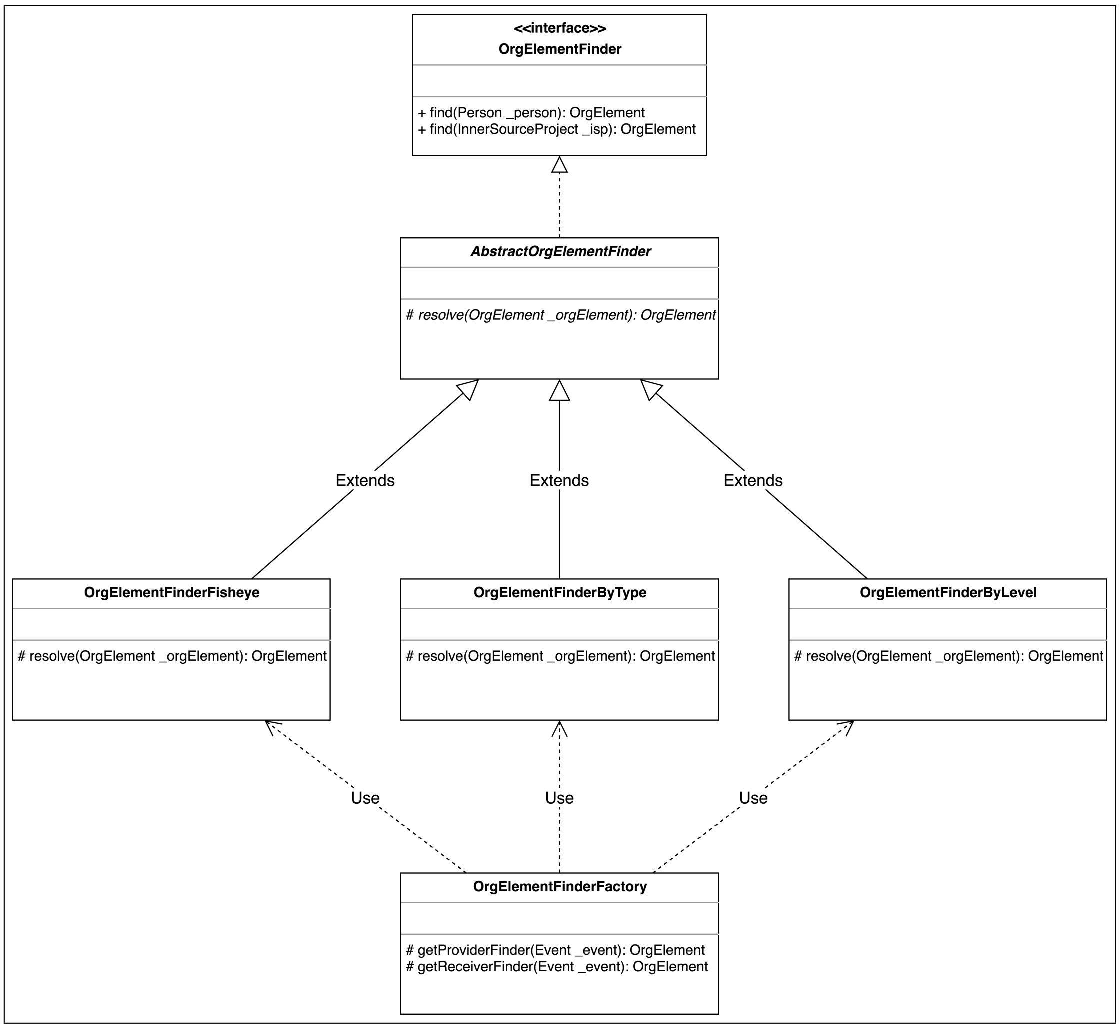
An abstract organizational element finder class now implements this interface. Finders that extend this abstract class have to implement a resolve method, that returns a specific replacement organizational element. With this structure, we kept code redundancy to a minimum and simplified adding new finders in the future.

The finder factory is initialized with a granularity specification object that is part of the analysis specification, and that specifies how replacement elements should be selected. The factory has the following interface:

```
protected OrgElementFinder getProviderFinder(EconomicEvent _event);

protected OrgElementFinder getReceiverFinder(EconomicEvent _event);
```

Figure 4.7: The finder factory's interface methods

The need for two different get*Finder methods arose when we added the feature to separately filter by contributors/receivers or inside/outside agents. The finder factory has to analyze the granularity specification and the given event, before instantiating and returning the correct finder. The factory also caches the created filters.



<<interface>>
OrgElementFinder
+ find(Person _person): OrgElement
+ find(InnerSourceProject _isp): OrgElement
AbstractOrgElementFinder
# resolve(OrgElement _orgElement): OrgElement
Extends
Extends
Extends
OrgElementFinderFisheye
# resolve(OrgElement _orgElement): OrgElement
OrgElementFinderByType
# resolve(OrgElement _orgElement): OrgElement
OrgElementFinderByLevel
# resolve(OrgElement _orgElement): OrgElement
Use
Use
Use
OrgElementFinderFactory
# getProviderFinder(Event _event): OrgElement
# getReceiverFinder(Event _event): OrgElement

Figure 4.8: An UML model of the finder classes

4.2 Client side

We implemented the client side of the accountancy tool as a module of the CM-Suite angular web app. The accountancy module's main components are components for organizational element accountancy and inner source project accountancy. These components define the main functionality and user interface of the accountancy tool and extend an abstract accountancy component.

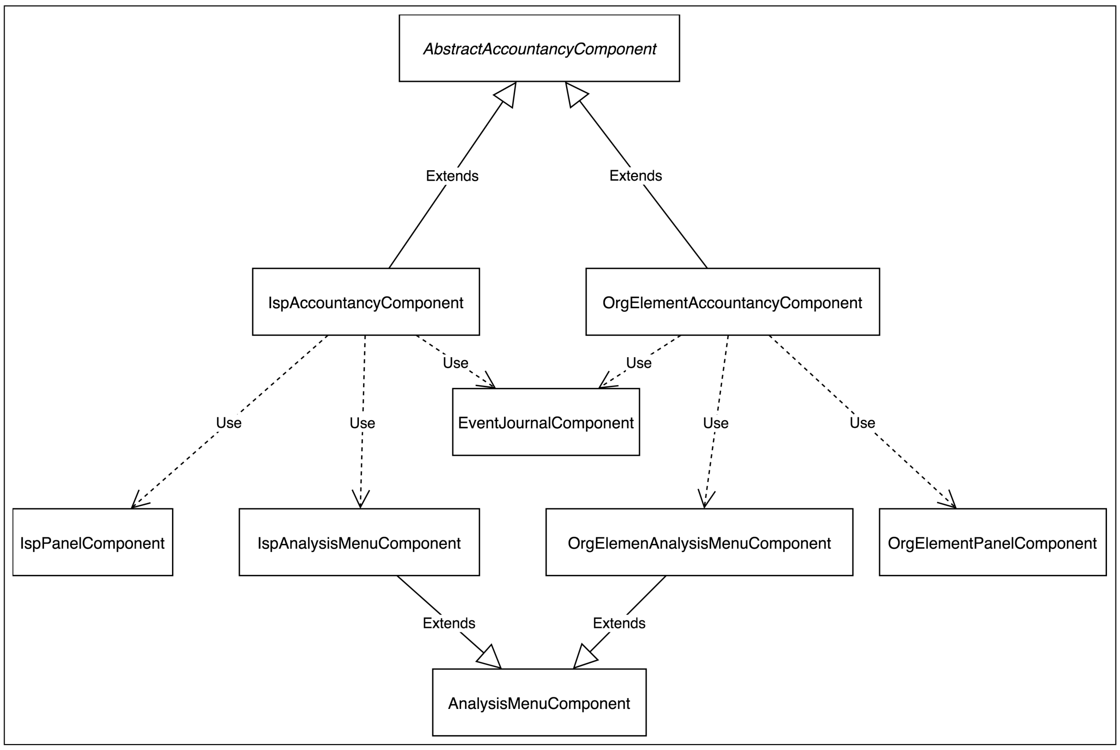


Figure 4.9: A model of the component structure

Both components consist of the following subcomponents:

- Panel

  The panel components display information about the currently selected context economic agent.

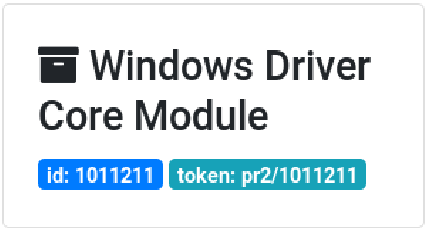
- Event journal

  The event journal components are responsible for creating the event journal. They display all economic events in a table, together with additional information about them.

- Analysis menu

  The analysis menu components create the analysis menu for the currently selected context agent.

The implementation of all three components differs based on the currently used context economic agent. However yet again, the implementations are related, so again we used abstract base classes.



Windows Driver Core Module

id: 1011211 token: pr2/1011211

Figure 4.10: The Panel, that shows information like the name, id and token of the current context agent. In this case the context agent is an inner source project

5 Implementation

In this section, we want to outline some of our implementation details.

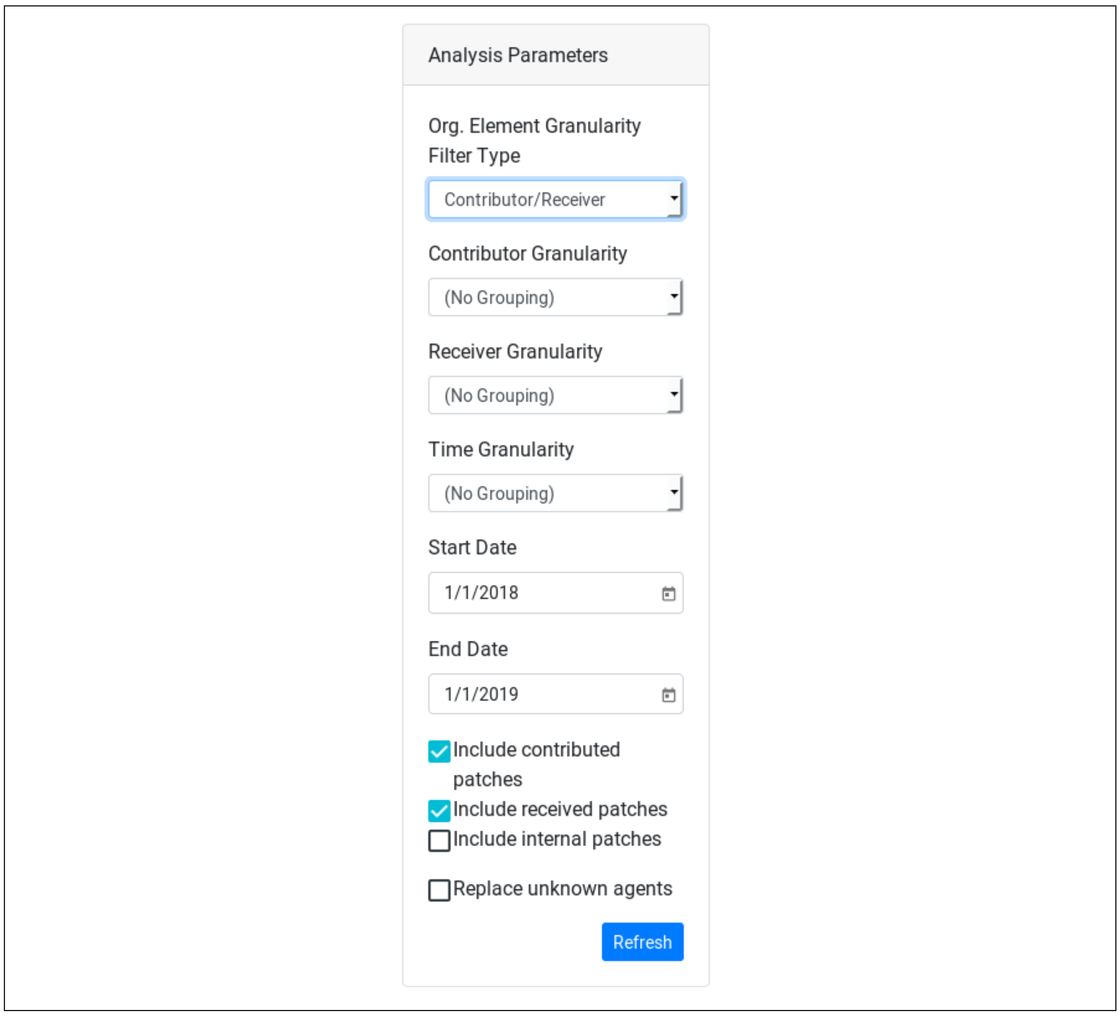
5.1 Analysis Specification

As explained in the design chapter, we use analysis specification objects to store and transport parameters of an analysis request. This way, we encapsulate the data, to make it persistable and to reduce the number of parameters we need to define in our method signatures. The analysis specification object on the client has the following attributes:

```
id: number;

timeGranularity: string;

eventScope: EventScope;

replaceUnknownAgents: Boolean;

startDate: Date;

endDate: Date;

granularitySpecification: GranularitySpecification;
```

Figure 5.1: The client's analysis specification object

A corresponding object in java exists on the server-side. When the user specifies the analysis parameters and clicks the refresh button, the analysis menu component analyses the input and then creates a specification object.


Analysis Parameters

Org. Element Granularity Filter Type: Contributor/Receiver

Contributor Granularity: (No Grouping)

Receiver Granularity: (No Grouping)

Time Granularity: (No Grouping)

Start Date: 1/1/2018

End Date: 1/1/2019

[✓] Include contributed patches
[✓] Include received patches
[ ] Include internal patches

[ ] Replace unknown agents

Figure 5.2: The new analysis menu

It emits an event that holds the specification. The currently active accountancy component receives this event and then uses the analysis specification service to load the specified economic events. Loading events consists of two steps:

1. Sending one request to the server in order to store the analysis specification object. The server then returns the id of an object. If the specification already exists on the server, it returns the existing id.

2. Taking the analysis specification id, that was received in step 1 to request the events.

We chose this solution because it will allow us to introduce saved event journal views in the future. If the client already has the id of the specification, it does not need to send the specification to the server again.

5.2 Event Loader

The event loader is used in the economic event service classes of the server to load economic events. Similar to the event service classes, three types of event loaders exist for the different context economic agents:

- Organizational element loader
- Inner source project event loader
- Person event loader

All event loaders implement the same interface and extend the same abstract event loader class. Therefore, event loader classes have to implement the following two abstract hook methods:

```
protected abstract Set<Person> getPersons();

protected abstract Set<InnerSourceProject> getProjects();
```

Figure 5.3: The two hook methods, eventloader classes have to implement

The `getPersons()` and `getProject()` methods have to return all persons or projects that are assigned to the current agent. All three event loader classes implement these methods differently:

- Person event loader
  - `getPersons()`:
    Returns a list that only contains the context person.
  - `getProjects()`:
    Returns an empty list, as there are no inner source projects assigned to a person.
- Inner source project event loader
  - `getPersons()`:
    Returns an empty list, as there are no persons assigned to inner source projects.
  - `getProjects()`:
    Returns a list that only contains the context project.
- Organizational element event loader
  Traverses the subtree and recursively adds all assigned persons or inner source projects to a list. It then returns that list.

In the abstract class, these hook methods are used to retrieve patches from the database, where the persons or projects are involved.

The event scope object of the analysis specification determines which events to load.

```
@Override
public List<? extends EconomicEvent> getEvents(EventScope _scope,
        Date _startDate, Date _endDate) {

    List<EconomicEvent> events = new ArrayList<>();

    if (_scope.isIncludeReceivedEvents()) {
            merge(events, getReceivedEvents(_startDate, _endDate));
    }

    if (_scope.isIncludeContributedEvents()) {
            merge(events,
                getContributedEvents(_startDate, _endDate));
    }

    if (_scope.isIncludeInternalEvents()) {
            merge(events, getInternalEvents(_startDate, _endDate));
    }

    return events;
}
```

Figure 5.4: The `getEvents(EventScope _scope, Date _startDate, Date _endDate)` method retrieves events based on the given date range and event scope

The `getInternalEvents()` method retrieves all events where both participating parties are assigned to the context agent. `GetContributedEvents()` returns all events, where the contributor is one of the assigned persons, while the receiver is not in the inner source project list. `GetReceivedEvents()` returns all events where the receiver is an assigned, and the contributor is external.

5.3 Sorting Filter

Before implementing the sorting filter, we had to specify a sorting order. We decided to order by date first. If two events happened on the same date, we order alphabetically by the provider name. If this does not specify a definite order, we order alphabetically by the receiver name. If the events are equal in all these topics, we order by direction of the event (increment > decrement).

First, we planned to implement the sorting into the database requests, to guarantee optimal performance. However, we realized that this was impossible with our current filter structure. For example, the date truncation filter changes the event dates after they have been read from the database. Eventually, we resorted to implementing the sorting as an organizational element filter. The sorting filter is always the last element in the filter chain.

```
@Override
public List<? extends EconomicEvent> filter(
        List<? extends EconomicEvent> _events) {

    _events.sort((EconomicEvent _a, EconomicEvent _b) -> {
        // try to sort by date
        int dateCompare = compareEventDates(_a, _b);
        if (dateCompare != 0) {
            return dateCompare;
        }
        // if date is equal sort by provider name
        int providerNameCompare = compareEventProvider(_a, _b);
        if (providerNameCompare != 0) {
            return providerNameCompare;
        }
        // if provider name is equal sort by receiver name
        int receiverNameCompare = compareEventReceiver(_a, _b);
        if (receiverNameCompare != 0) {
            return receiverNameCompare;
        }
        // if receiver name is equal sort by direction
        int directionCompare = compareEventDirection(_a, _b);
        return directionCompare;
        });
    return _events;

}
```

Figure 5.5: The implementation of the sorting filter

5.4 Finders

Organizational element finders are used by the organizational element filter, to replace agents that are participating in economic events. As already described in chapter 4, all finder classes have to extend the abstract finder class and thus implement the following method, that is used to find replacement agents:

```
protected abstract OrgElement resolve(OrgElement _orgElement);
```

Figure 5.6: The hook method, that all finder classes have to implement

5.4.1 By-Type-Finder

The `resolve(OrgElement _orgElement)` method of the by-type-finder returns an ancestor of the given organizational element, that is of a specified type.

As an example, we have an organization tree as in Figure 5.7. We instantiate the by-type-finder with the type "business unit." When we now call the finder's resolve method with team C as a parameter, the method will return the organizational element B, that is of the type business unit.

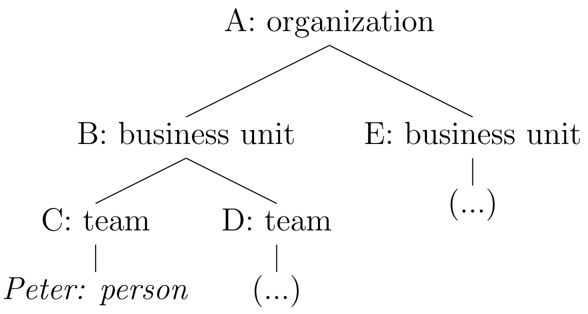


Figure 5.7: An example organization tree

To collect ancestors, the filter repeatedly retrieves the parent elements from the database.

```
protected OrgElement resolve(OrgElement _orgElement) {

    // Iterate up the tree to find OrgElement of orgElementType
    OrgElement current = _orgElement;
    for (int i = 0; i < MAX_SEARCH_LEVELS; i++) {

        if (current == null) {
        return current;
    }

    if (current.getType() != null &&
            current.getType().equals(orgElementType)) {
        return current;
    }

    current = getParentLoadingCache()
            .getUnchecked(current).orNull();

    }

    return null;

}
```

Figure 5.8: The `resolve(OrgElement _orgElement)` method of the by-type-finder

This algorithm already worked fine in the pre-existing tool. However, it assumed that every agent only had one parent element, so it stopped working when agents had multiple parent elements in multiple dimensions. Since the finder did not take these dimensions into account when retrieving the data from the database, it received more than one organizational element. The filter did not expect this and hence returned the error message we had described in chapter 3. To resolve this problem, we had to add the current dimension to the `getParent()` calls that are cached in a loading cache.

```
protected void buildParentLoadingCache() {
    if (parentLoadingCache == null) {
        parentLoadingCache = CacheBuilder.newBuilder().build(
            new CacheLoader<OrgElement, Optional<OrgElement>>() {

            @Override
        public Optional<OrgElement> load(
                OrgElement _orgElement) throws Exception {

            return Optional.fromNullable(
            orgElementDao.getParent(_orgElement,
                    orgDimension));

        }

    });
    }
}
```

Figure 5.9: The initialization of the parent loading cache with the added dimension

5.4.2 By-Level-Finder

The `resolve(OrgElement _orgElement)` method of the by-level-finder should return an ancestor element of the given organizational element. To achieve this, it initially creates a list of all ancestors, using the `getAncestorList(OrgElement _orgElement)` method. This method creates a list of all ancestors by traversing the organizational tree upwards until the root is reached.

```
protected List<OrgElement> getAncestorList(OrgElement _orgElement) {
    ArrayList<OrgElement> ancestors = new ArrayList<>();
    OrgElement current = _orgElement;

    while (current != null) {
        ancestors.add(0, current);
        current = getParentLoadingCache()
                .getUnchecked(current).orNull();
    }

    return ancestors;
}
```

Figure 5.10: The `getAncestorList(OrgElement _orgElement)` method creates a list, that contains all ancestor element of the parameter organizational element

Since we always add all ancestors as the first element in the list, we end up with a list, that is automatically sorted by level. Retrieving the correct element is now as simple as taking the element from the list, that has the searched level as the index. If no such element exists, because the list is too short, we return null.

```
protected OrgElement resolve(OrgElement _orgElement) {

    if (_orgElement == null) {
        return null;
    }

    List<OrgElement> path = getAncestorList(_orgElement);

    if (path.size() <= level) {
        return null;
    } else {
        return path.get(level);
    }

}
```

Figure 5.11: The by-level-finder's implementation of the `resolve(OrgElement _orgElement)` method

5.4.3 Fisheye Finder

The Fisheye Finder's resolve algorithm searches for a common ancestor element of the context agent and the participating agent. It then replaces the agent by a child of the found element that is also an ancestor of the original participating agent.

We will use the organization tree of Figure 5.12 for a short example.

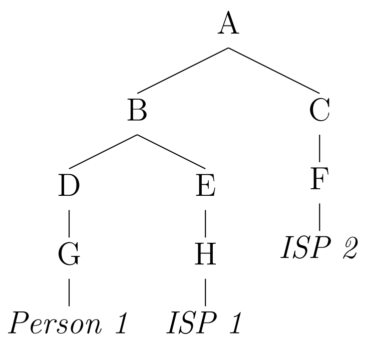


Figure 5.12: Another example organization tree

We assume that the following two patch contribution events exist:

$$\begin{aligned} &(1) \qquad Person\,1 \rightarrow ISP\,1 \\ &(2) \qquad Person\,1 \rightarrow ISP\,2 \end{aligned}$$

When we now open the event journal for the context element D, the journal shows us the two events, because person one is assigned to G which is a subordinate organization element of D. When we select the fisheye filter, the journal should display both events as follows:

$$(1) \qquad G \rightarrow E$$

$$(2) \qquad G \rightarrow C$$

On creation, the fisheye finder is initialized with a context organizational element. The `resolve(OrgElement _orgelement)` method begins with a call to the `initialize()` method, that creates a list of all ancestors of the current context organizational element if it doesn't already exist.

Next, the finder has to find the shared ancestor of the context element and the current element that is of the lowest organizational level. To do so, it creates a list of all ancestors of the current organizational element. Then it traverses said list backward and checks if the elements are also included in the context element's ancestors. If it is, it has found the shared ancestor with the lowest level. All it has to do now is to return the element of the current element's ancestor list, that comes right after the common ancestor. If there is no such element, it returns the common ancestor.

```
protected OrgElement doResolve(OrgElement _orgElement) {

    if (_orgElement == null) {
        return null;
    }

    List<OrgElement> agentPath = getAncestorList(_orgElement);

    ListIterator li = agentPath.listIterator(agentPath.size());

    // Iterate in reverse.
    while (li.hasPrevious()) {
        if (contextPath.contains(li.previous())) {
            OrgElement commonAncestor =
                    (OrgElement) li.next();
            if (li.hasNext()) {
                return (OrgElement) li.next();
            } else {
                return commonAncestor;
            }
        }
    }
    return null;

}
```

Figure 5.13: The `doResolve(OrgElement _orgElement)` method, to retrieve replacement organizational elements based on the fisheye filter definition

5.4.4 Finder Factory

The job of the finder factory is to create finder objects based on the input it receives. On creation, the finder factory receives, among other objects, a granularity specification. The granularity specification object specifies the granularity of groups of economic agents (event provider, event receiver, intern agents, extern agents). The granularity for each group is specified by a granularity object. This object specifies if the granularity is defined by type or level and what value it has.

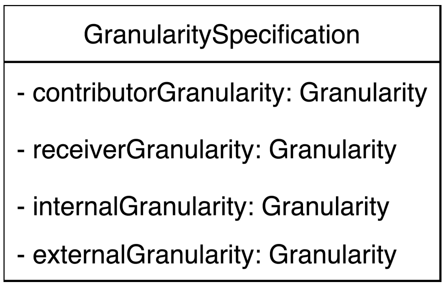

GranularitySpecification
- contributorGranularity: Granularity
- receiverGranularity: Granularity
- internalGranularity: Granularity
- externalGranularity: Granularity

Figure 5.14: A simplified class diagram of the granularity specification object

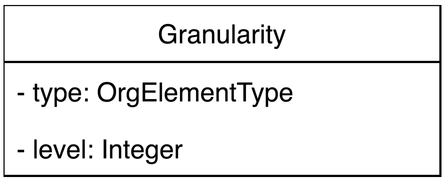


Figure 5.15: A simplified class diagram of the granularity object

When the `getProviderFilter(EconomicEvent _event)` or `getReceiverFilter (EconomicEvent _event)` methods are called, they have to decide which granularity object should be used for determining the correct finder. For this decision, it takes both the granularity specification and the event direction into account. The code looks as shown in Figure 5.16 and Figure 5.17.

The `getFinder(Granularity _gran)` method takes a granularity object and then returns either a by-type-finder or a by-level-finder object.

```
protected OrgElementFinder getProviderFinder(EconomicEvent _event) {

    OrgElementFinder providerFinder = null;

    if (granularitySpecification.hasInternalGranularity()) {
        providerFinder = getFinder(granularitySpecification
                .getInternalGranularity());
    }

    if (granularitySpecification.hasContributorGranularity()
            && (_event.getDirection() == EconomicEvent
                .Direction.DECREMENT)) {

        providerFinder = getFinder(granularitySpecification
                .getContributorGranularity());

    }

    if (granularitySpecification.hasReceiverGranularity()
            && (_event.getDirection() == EconomicEvent
                    .Direction.INCREMENT)) {

        providerFinder = getFinder(granularitySpecification
                .getReceiverGranularity());

    }

    if (granularitySpecification.getFisheyeGranularity()) {
        providerFinder = fisheyeFinder;
    }

    return providerFinder;
}
```

Figure 5.16: The `getProviderFinder(EconomicEvent _event)` method returns a finder to retrieve a replacement provider for the given event

```
protected OrgElementFinder getReceiverFinder(EconomicEvent _event) {
    OrgElementFinder receiverFinder = null;

    if (granularitySpecification.hasExternalGranularity()) {
        receiverFinder = getFinder(granularitySpecification
                .getExternalGranularity());
    }

    if (granularitySpecification.hasContributorGranularity()
            && (_event.getDirection() == EconomicEvent
                    .Direction.INCREMENT)) {

        receiverFinder = getFinder(granularitySpecification
                .getContributorGranularity());

    }

    if (granularitySpecification.hasReceiverGranularity()
            && (_event.getDirection() == EconomicEvent
                    .Direction.DECREMENT)) {

        receiverFinder = getFinder(granularitySpecification
                .getReceiverGranularity());

    }

    if (granularitySpecification.getFisheyeGranularity()) {
        receiverFinder = fisheyeFinder;
    }

    return receiverFinder;
}
```

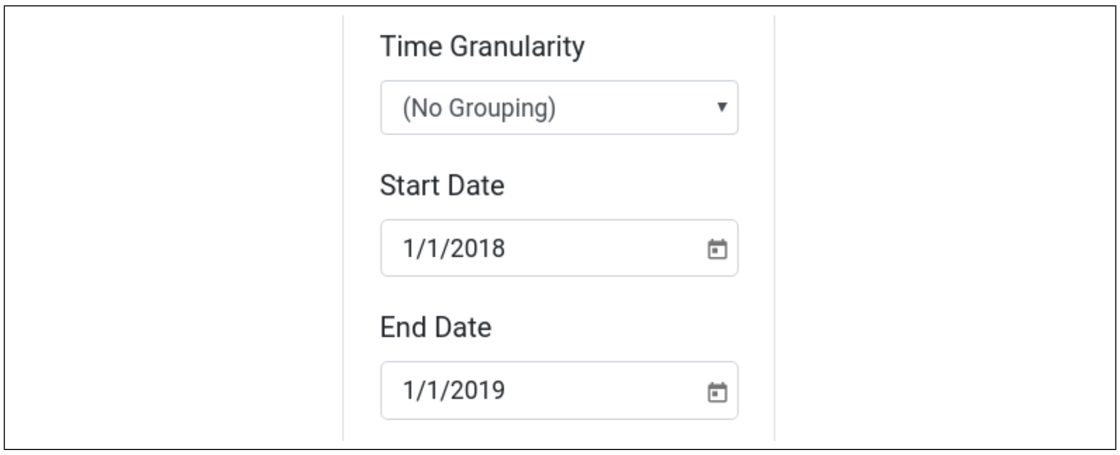
Figure 5.17: The `getReceiverFinder(EconomicEvent _event)` method returns a finder to retrieve a replacement receiver for the given event

5.5 Filter by Date

The date filter limits the shown economic events, by events that happened during a custom time period.

To allow the user to select a time period, we added two date pickers to the menu. Even though the web app so far used bootstrap design and components, we used the date picker components from Google's material angular component collection. We decided on the material component because there were already plans to switch the whole CMSuite web app to Google's material design. However, we added a

CSS stylesheet to make the input field blend in with the other fields for now.


Time Granularity

(No Grouping)

Start Date

1/1/2018

End Date

1/1/2019

Figure 5.18: The datepicker input fields, blending in with other bootstrap inputs

The analysis menu component reads the input from the date picker forms and adds it to the analysis specification object. To detain the user from invalid inputs, we use the date picker component's filter function. The component allows us to specify a callback function for a filter that validates the date input. We defined the callback functions that are shown in Figure 5.19.

On the server side, the start and end dates are passed through and eventually added to the database query that retrieves the events. This way we profit from the increased performance of optimized database queries.

```
public validStartDateFilter(date: Date): boolean {
    if (date == null) {
      return false;
    }
    let endDate: Date = this.form.get('endDate').value;
    if (endDate != null) {
      if (date > endDate) {
        return false;
      }
    }
    return true;
  }

public validEndDateFilter(date: Date): boolean {
    if (date == null) {
      return false;
    }
    let startDate: Date = this.form.get('startDate').value;
    if (startDate != null) {
      if (date < startDate) {
        return false;
      }
    }
    return true;
  }
```

Figure 5.19: The callback functions, to determine date validity

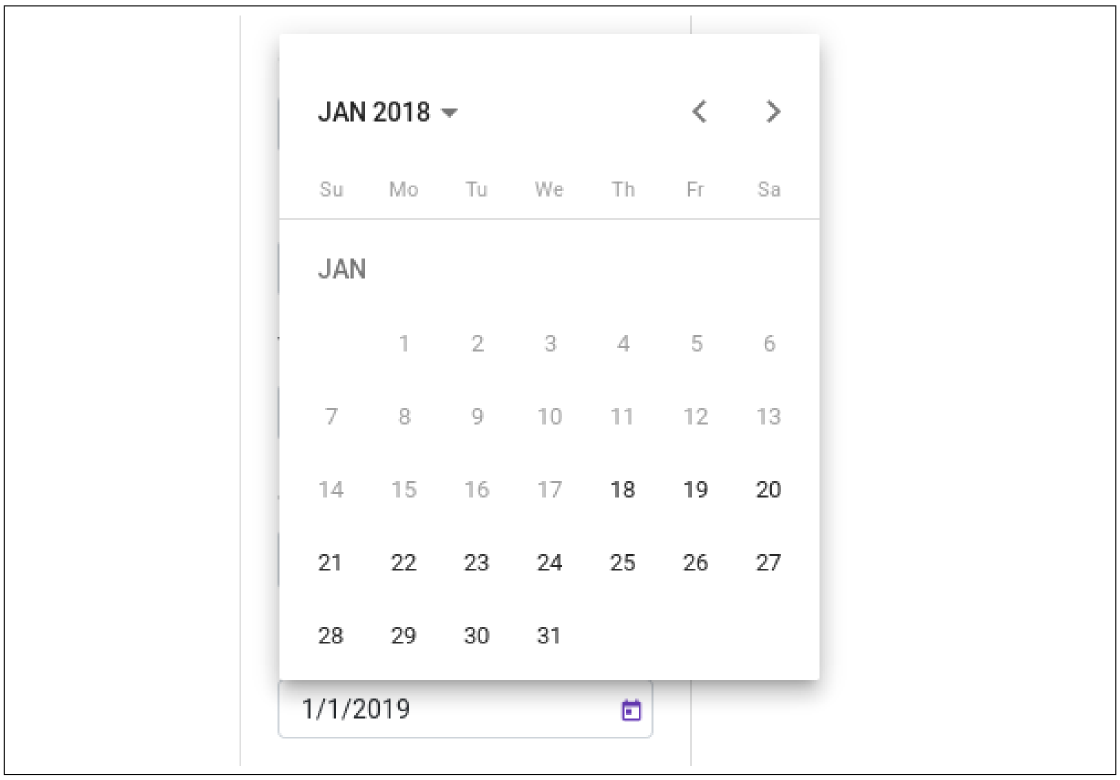


Figure 5.20: The date picker in expanded. The defined start date was the 18th of January 2018, so all dates before that are disabled, as they are invalid end dates

6 Evaluation

In this chapter, we will now evaluate our work with the evaluation scheme, that we described in chapter 3.

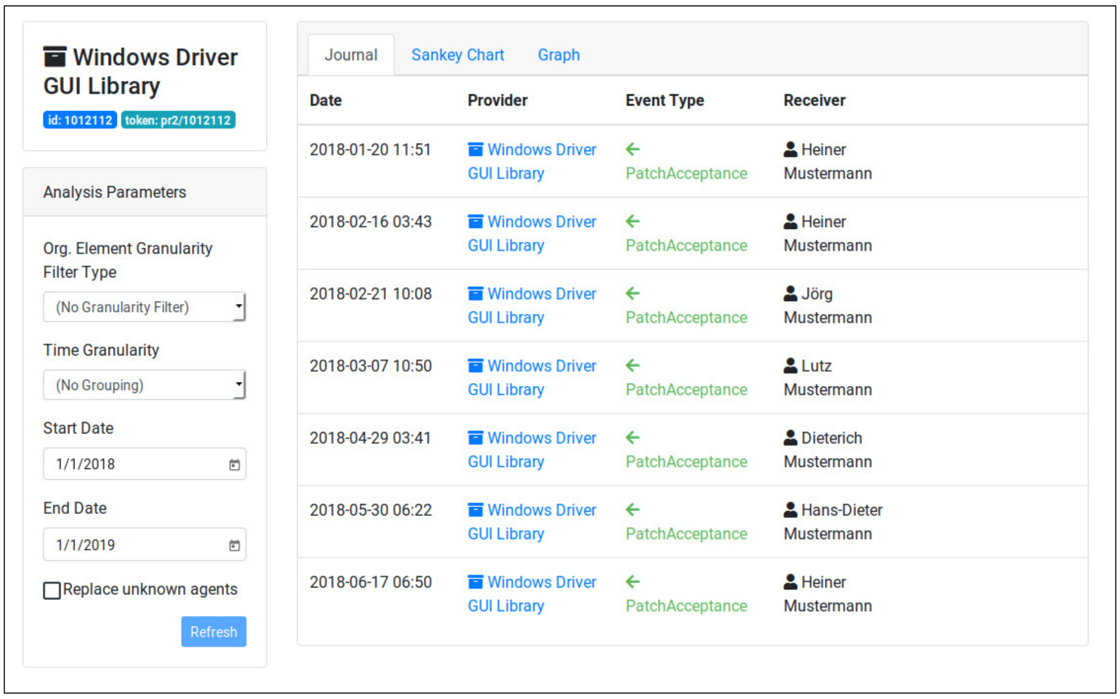
6.1 Functional Requirements

Event journal for inner source projects

Functional Requirement 1. *As the manager of an inner source project, I want the accountancy tool to include an event journal for inner source projects so that I can keep an overview over who contributes to my project.*

To fulfill this requirement, we started with the creation of a mockup event journal for inner source projects. Next, we implemented the needed components on the client. The result looks as shown in Figure 6.1.

The user can now view event journals for inner source projects, that display who contributes to them.



Windows Driver GUI Library

id: 1012112 token: pr2/1012112

Analysis Parameters

Org. Element Granularity Filter Type: (No Granularity Filter)

Time Granularity: (No Grouping)

Start Date: 1/1/2018

End Date: 1/1/2019

☐ Replace unknown agents

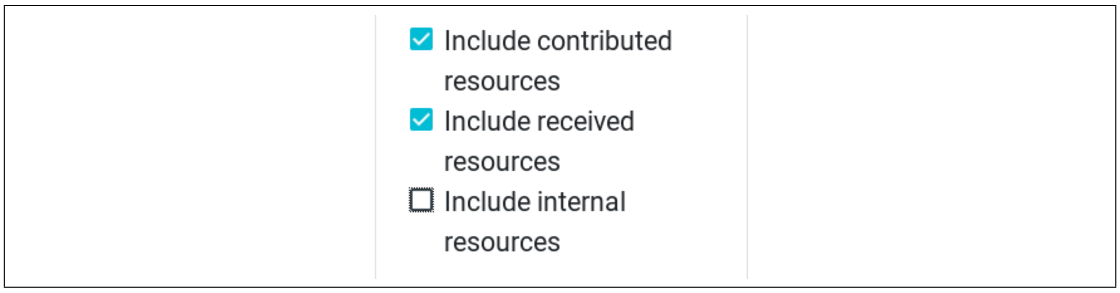
Date	Provider	Event Type	Receiver
2018-01-20 11:51	Windows Driver GUI Library	PatchAcceptance	Heiner Mustermann
2018-02-16 03:43	Windows Driver GUI Library	PatchAcceptance	Heiner Mustermann
2018-02-21 10:08	Windows Driver GUI Library	PatchAcceptance	Jörg Mustermann
2018-03-07 10:50	Windows Driver GUI Library	PatchAcceptance	Lutz Mustermann
2018-04-29 03:41	Windows Driver GUI Library	PatchAcceptance	Dieterich Mustermann
2018-05-30 06:22	Windows Driver GUI Library	PatchAcceptance	Hans-Dieter Mustermann
2018-06-17 06:50	Windows Driver GUI Library	PatchAcceptance	Heiner Mustermann

Figure 6.1: The event journal for an inner source project

Include internal events

Functional Requirement 2. *As the manager of an organizational unit, I want the accountancy tool's event journal to have an option to also include internal events so that I have an insight into the flow of code contributions within my unit.*

For this requirement, we added an option to the analysis menu to precisely select the desired scope.



☑ Include contributed resources
☑ Include received resources
☐ Include internal resources

Figure 6.2: Checkboxes to define the desired scope

By default, the options "include contributed resources" and "include received resources" are selected. If the user wants to, he can adapt the selection and also add internal events. As a result of this feature, he can view events that take place between subordinates.

Filter events by date

Functional Requirement 3. *As a user of the accountancy tool, I want the event journal to have the option to filter its events by their date so that I can hide useless information.*

The user can now select start and end dates to define a period in which all displayed events should have taken place.

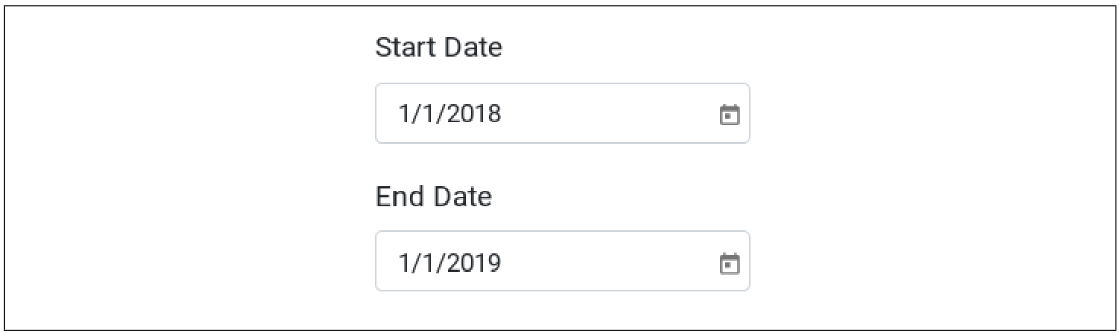


Figure 6.3: The input fields to select the start and end dates

When the user selects such a date range, the journal only displays events that happened within the range.

Filter by business year by default

Functional Requirement 4. *As a manager of an organizational unit, I want the event journal to show only the events of one business year by default so that I save time when using the accountancy tool.*

By default, the date input fields select the first of January of the current year as the start date and the first of January of the next year as the end date. Thus, the journal shows the events of one whole business year by default.

Define an explicit sorting order

Functional Requirement 5. *As a user of the accountancy tool, I want the event journal to sort the output by an explicitly defined order so that I have a better overview of the data.*

We defined a default order, as we explained in chapter 5 and implemented the sorting filter to sort the events. As a result, all events are ordered consistently.

Fully support multiple organizational dimensions

> **Functional Requirement 6.** *As a user of the accountancy tool, I want the organization element filters to fully support multiple organizational dimensions so that they are better adapted to my organizations structures.*

To fulfill this requirement, we started with the problems we found, that were connected to the missing support of dimensions. We added the dimension to the organizational element granularity filter logic. Now, it does not throw an error message anymore, when the database model uses multiple dimensions.

We also added dimension to the model of the organizational element types. We now can retrieve element types by their dimension. After we used this new interface, only the relevant organizational element types are shown in the organizational element type filter.

Additionally, we added the dimension id to the URL of the accountancy tool, so that the user can now directly navigate to the event journal of an agent in a specific dimension.

All in all, the tool now fully supports multidimensional data models.

Organizational level filter

> **Functional Requirement 7.** *As a user of the accountancy tool, I want the event journal to have an option to show the economic agents, based on their organizational level so that I can adapt the event journal's information to my needs.*

The user can now choose to filter by the organizational level. Then all agents that participate in an event are replaced by their superior agents of the given level.

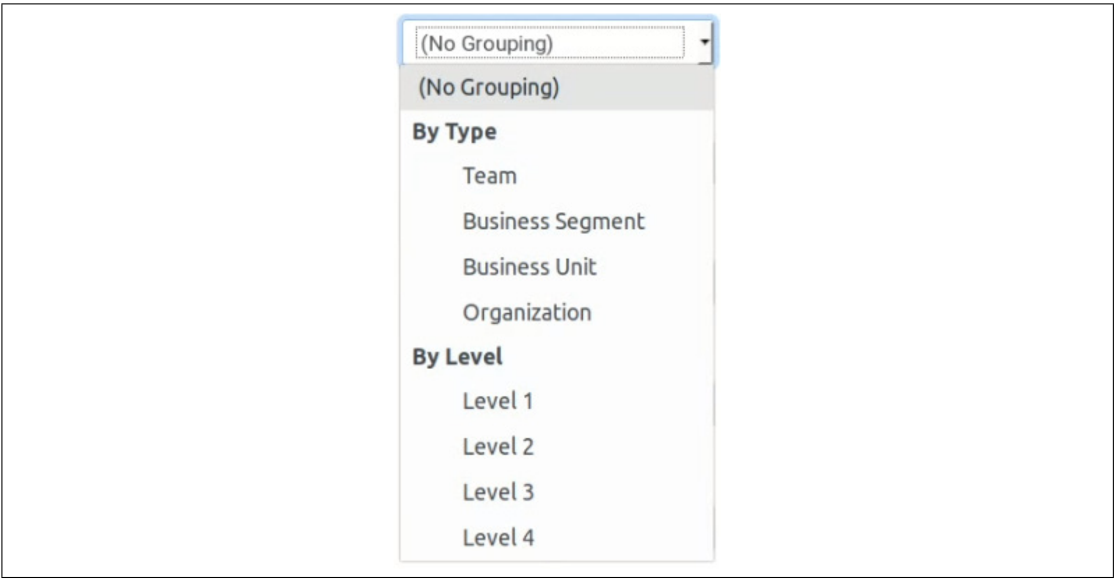
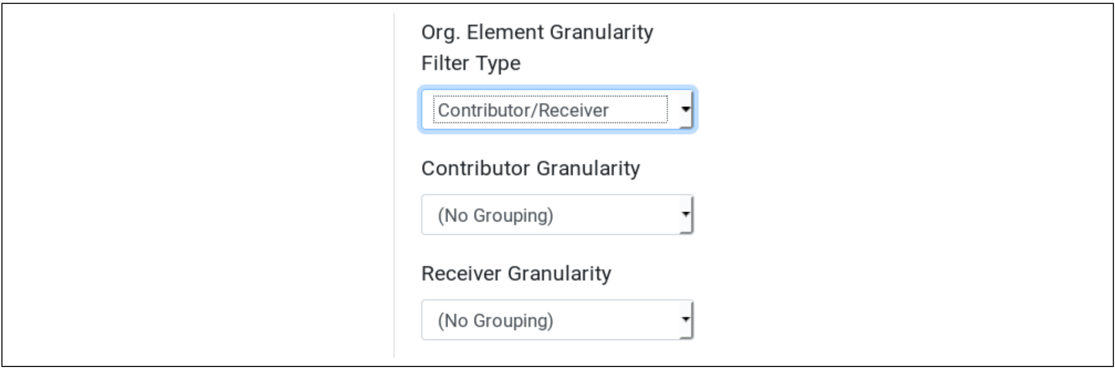


Figure 6.4: Options to select levels for the organization element filter

Apply filters to internal/ external parties

Functional Requirement 8. *As a user of the accountancy tool, I want the event journal to have an option to apply filters to internal and external parties independently so that I can adapt the event journals information to my needs.*

The user has the possibility to define the granularity of all internal or external parties independently.



Org. Element Granularity Filter Type: Contributor/Receiver

Contributor Granularity: (No Grouping)

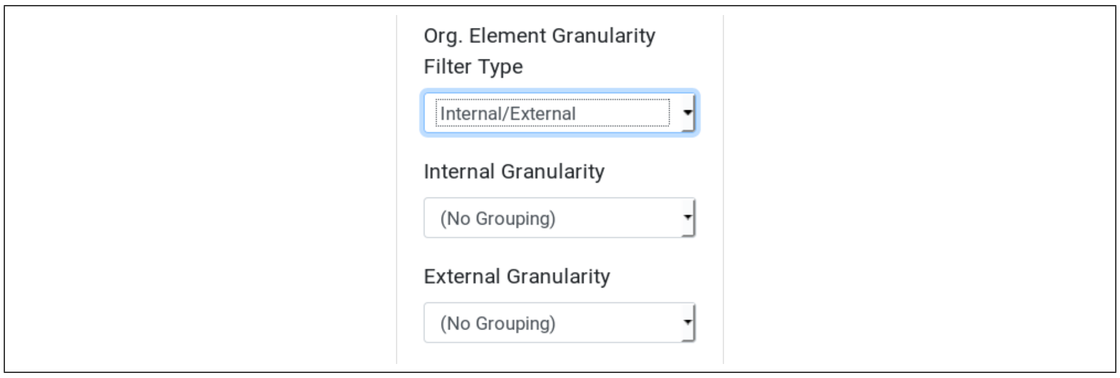
Receiver Granularity: (No Grouping)

Figure 6.5: The selection fields for separate contributor/ receiver filtering

Apply filters to contributors/ receivers

Functional Requirement 9. *As a user of the accountancy tool, I want the event journal to have an option to apply filters to contributing and receiving* *parties independently so that I can adapt the event journals information to my needs.*

The user has the possibility to define the granularity of all contributors or receivers independently.


Org. Element Granularity Filter Type: Internal/External

Internal Granularity: (No Grouping)

External Granularity: (No Grouping)

Figure 6.6: The selection fields for separate internal/ external filtering

Fisheye filter

Functional Requirement 10. *As a user of the accountancy tool, I want the event journal to have an option to give me a fisheye view on the events so that I can have a natural overview of economic events.*

The user can select a fisheye filter, that implements the filter function, we described in chapter 3. Then all participating agents are replaced by superior agents based on the filter definition.

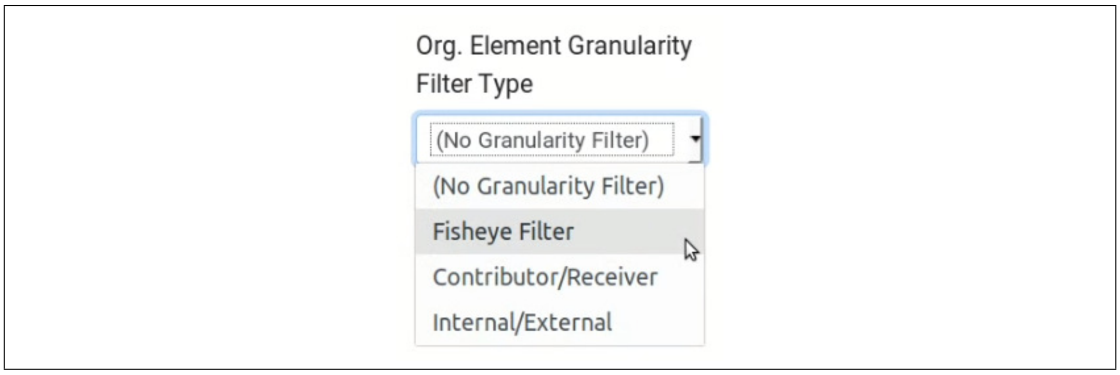

Org. Element Granularity Filter Type: (No Granularity Filter)

Figure 6.7: The selection field to choose the type of the organization element filter

Replace unknown agents

> **Functional Requirement 11.** *As a user of the accountancy tool, I want the event journal to have an option to replace unknown agents so that I can see who contributed to projects or what ISP received contributions.*

The user now has an option to replace unknown agents with the original agents. After choosing to replace unknown agents, the event journal shows the names of the original person/ inner source project in grey font, instead of showing unknown agents.

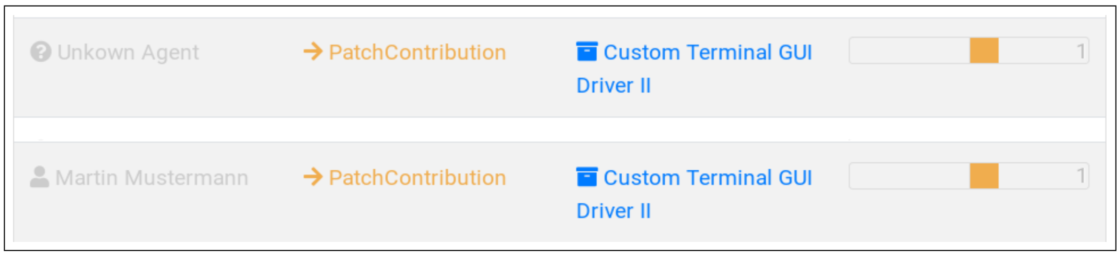


Figure 6.8: An unknown agent before and after selecting the replace-unknown-agents option

6.2 Nonfunctional Requirements

Functionality independent from client implementation

> **Nonfunctional Requirement 1.** *As a developer of the accountancy tool, I want the functionality of the tool as independent as possible from the client implementation, so that I can add other clients more easily in the future.*

Throughout this whole thesis, we focused on implementing as much as possible on the server side. The client only implements features to navigate, adapt and display the data. All tasks that change data happen on the server.

Consistent coding style

> **Nonfunctional Requirement 2.** *As a developer of the accountancy tool, I want new code to follow the given style guidelines so that all code has a uniform style.*

CMSuite uses a continuous integration pipeline, that automatically verifies that all code follows the style guidelines. The current version of the accountancy tool passes all style checks.

Appropriate loading times

Nonfunctional Requirement 3. *As a user of the accountancy tool, I want the tool to have fast loading times so that I can finish my work quicker.*

To evaluate this requirement, we performed speed tests on the event journal with our demo data set. Specifically, we used the event journal for the root organizational element, the “Rainbow Print SE.” To measure the loading times, we used Google Chrome’s network conditions tool, that allows throttling the network speeds. We used the default “fast 3G” setting, to enable consistent, reproducible data. We defined “fast loading times” as less than 5 seconds with the “fast 3G” setting. Next, we loaded the event journal for various analysis specifications. We reached the maximum loading time when we added all possible filter options: Internal/ external filtering with organizational element types, aggregate by commit day, include all events and replace unknown agents. We ended up with a loading time of 3.18 seconds, which is below our defined maximum.

Good visuals

Nonfunctional Requirement 4. *As a user I want the accountancy tool to be visually appealing so that I can have a high-quality user experience.*

Within this thesis, we located some visual deficiencies (chapter 3) and fixed them. The event journal table does not overflow the border anymore. Furthermore, the newly designed resource gauge is better readable and overall more visually appealing. We also remodeled the way dates were displayed. Now the date format depends on the used time granularity and automatically hides unnecessary information (e.g., the exact time, when the granularity is “daily”). Additionally, we added date headers, to separate different date groups.

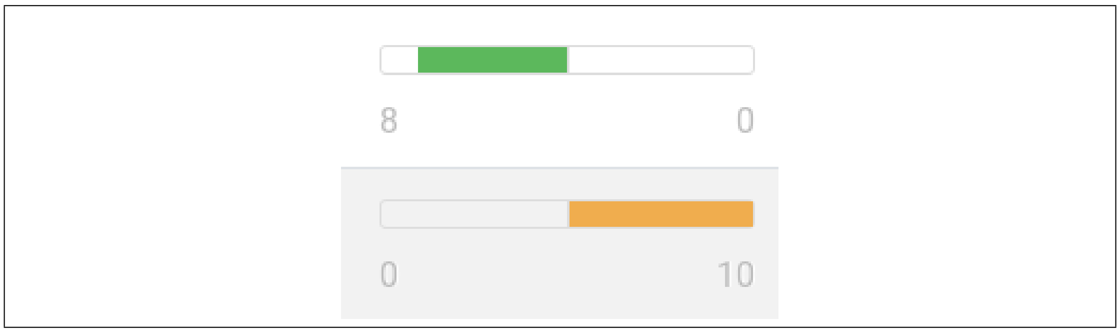
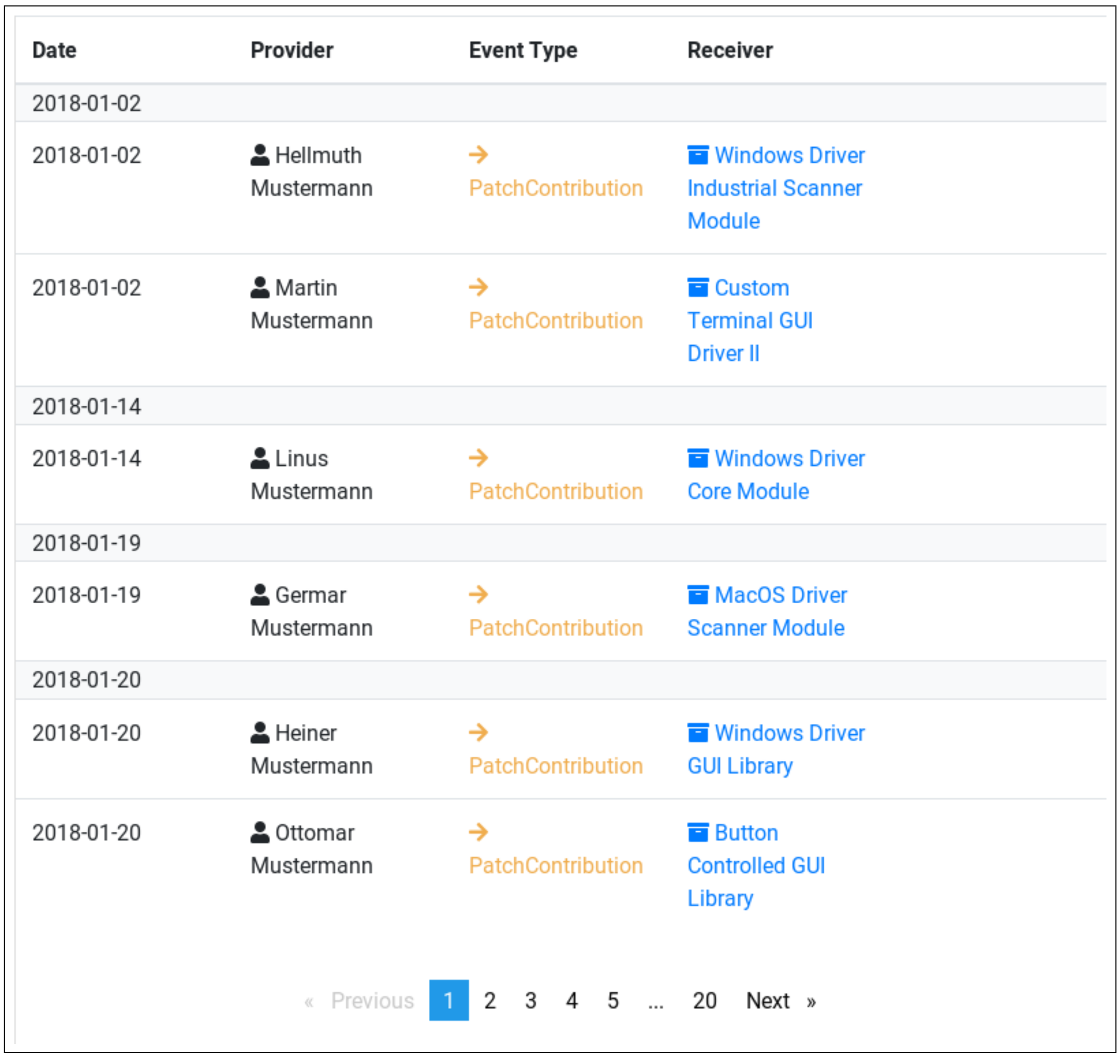


Figure 6.9: The newly designed resource gauge



Date	Provider	Event Type	Receiver
2018-01-02			
2018-01-02	Hellmuth Mustermann	→ PatchContribution	Windows Driver Industrial Scanner Module
2018-01-02	Martin Mustermann	→ PatchContribution	Custom Terminal GUI Driver II
2018-01-14			
2018-01-14	Linus Mustermann	→ PatchContribution	Windows Driver Core Module
2018-01-19			
2018-01-19	Germar Mustermann	→ PatchContribution	MacOS Driver Scanner Module
2018-01-20			
2018-01-20	Heiner Mustermann	→ PatchContribution	Windows Driver GUI Library
2018-01-20	Ottomar Mustermann	→ PatchContribution	Button Controlled GUI Library

Figure 6.10: The event journal with the time granularity set to daily

Thorough testing

> **Nonfunctional Requirement 5.** *As a developer, I want all the features of the accountancy tool to be extensively tested so that I can find bugs before deploying the code.*

The continuous integration pipeline also runs all defined unit, integration and system integration tests. We added unit tests for all new features and adapted current unit tests to our refactorings. For functionality that works with external systems, we added or modified integration tests.

Consistent domain model and terminology

Nonfunctional Requirement 6. *As a stakeholder, I want the accountancy tool to use a consistent domain model and terminology.*

In chapter 3 we identified numerous inconsistencies of CMSuite's domain model. However, we noticed those problems close to the end of this thesis. We decided to mention them, even though we could not remove them as part of this thesis. Nevertheless, we devised some solutions, that we will discuss in the chapter 7.

6.3 Conclusion

As Figure 6.11 shows, we fulfilled all but one requirement fully. Therefore, we consider our work as successful.

Functional requirements	
FR1: Event journal for inner source projects	🟩
FR2: Include internal events	🟩
FR3: Filter events by date	🟩
FR4: Filter by business year by default	🟩
FR5: Define an explicit sorting order	🟩
FR6: Fully support multiple organizational dimensions	🟩
FR7: Organizational level filter	🟩
FR8: Apply filters to internal/external parties	🟩
FR9: Apply filters to contributors/receivers	🟩
FR10: Fisheye filter	🟩
FR11: Replace unknown agents	🟩
Nonfunctional requirements	
NR1: Functionality independent from client implementation	🟩
NR2: Consistent coding style	🟩
NR3: Appropriate loading times	🟩
NR4: Good visuals	🟩
NR5: Thorough testing	🟩
NR6: Consistent domain model and terminology	🟥

Figure 6.11: The evaluation of the requirements, using our defined scheme

As a result of this thesis, managers of organizations that practice inner source and use the accountancy tool, have many new options to view, select and aggregate economic events in the event journal. We removed several existing errors and thus improved the user experience. Some of our refactorings (e.g., the analysis specification object or the organizational element finder structure) facilitate further development. So all defined stakeholders profit from this work.

7 Future Work

As we stated in chapter 6, our work in this thesis successfully improved the functionality of the accountancy tool. However, there is room for further improvement. One example are the model inconsistencies, we found in chapter 3. Unfortunately, we were not able to remove them as part of this thesis. Nevertheless, we devised some possible improvements for the model:

- Consistently use the terms “Inside/Outside Agent” instead of “Internal/External Agent,” to stick with the REA terminology.
- Define participating agents in economic events as “contributor” and “receiver” instead of “provider” and “receiver.” This way, we remove confusion for events, where the event receiver is not the receiver of the contribution.
- Use the term “code contribution” instead of “patch” in the model, to correctly term regular code contributions.
- Model inner source projects as economic resources, to stay true to REA definitions.
- Introduce proper exchanges, to properly model duality in events. One example would be to introduce transfer pricing to the model.

Furthermore, we thought of some features, that would significantly improve the functionality of the tool:

- Expand and Collapse Agents

  Allow the user to expand and collapse specific agents manually. If the user collapses an agent, it and all its siblings get replaced by the parent agent. When an agent is expanded, the agent should be split up into its children.

- Respect Organizational Element Changes in Time

  The structure of a company changes over time. The accountancy tool has to take the event date and the structure at that time into consideration when retrieving events.

- Custom Time Granularities

  Organizations often function in arbitrary rhythms. Other time granularities than the predefined ones could be needed. Therefore, the accountancy tool should have an option to allow the users to define custom granularities.

We believe that these suggested features will increase the usefulness of the accountancy tool and thus also the value to the users. Hence, further development of the accountancy tool should focus on these features.

References

Capraro, M., Dorner, M. & Riehle, D. (2018). The patch-flow method for inner source collaboration. In *15th international conference on mining software repositories*. ACM.

Capraro, M. & Riehle, D. (2016). Inner source definition, benefits, and challenges. *ACM Computing Surveys*, *49*(4), 0–36.

Dinkelacker, J., Garg, P. K., Miller, R. & Nelson, D. (2002). Progressive open source.

Hibernate ORM. (n.d). Retrieved December 15, 2018, from http://hibernate.org/orm/

Ijiri, Y. (1975). *Theory of accounting measurement*. American Accounting Association.

Jersey. (2018). Retrieved December 15, 2018, from https://jersey.github.io/

McCarthy, W. E. (1982). The rea accounting model: A generalized framework for accounting systems in a shared data environment. *The Accounting Review*, *57*(3), 554–578.

PostgreSQL. (n.d). Retrieved December 15, 2018, from https://www.postgresql.org/about/

Raymond, E. S. (2000). The cathedral and the bazaar. Retrieved December 3, 2018, from http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/

Riehle, D., Ellenberger, J., Menahem, T., Boris, M., Naveh, B. & Odenwald, T. (2009). Open collaboration within corporations using software forges. *IEEE Software*, *26*(2), 52–58.

Stol, K.-J., Avgeriou, P., Bahar, M. A., Lucas, Y. & Fitzgerald, B. (2014). Key factors for adopting inner source. *ACM Transactions on Software Engineering and Methodology*, *23*(2).

Wesselius, J. (2008). The bazaar inside the cathedral: Business models for internal markets. *IEEE Software*, *25*(3), 60–66.

Yu, S. C. (1976). *Structure of accounting theory*. University Press of Florida.