# Usability von Interaktiven Bäumen für Remote-Konfigurationsmanagement

## mit einer Webapplikation via Elektra

BACHELORARBEIT

zur Erlangung des akademischen Grades

**Bachelor of Science**

im Rahmen des Studiums

**Wirtschaftsinformatik**

eingereicht von

**Daniel Bugl**
Matrikelnummer 01425285

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Franz Puntigam
Mitwirkung: Univ.Ass. Dipl.-Ing. Markus Raab

Wien, 25. Juni 2018

| | |
|---|---|
| Daniel Bugl | Franz Puntigam |

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

# Usability of Interactive Trees for Remote Configuration Management

## through a Web Application using Elektra

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

**Bachelor of Science**

in

**Business Informatics**

by

**Daniel Bugl**
Registration Number 01425285

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Franz Puntigam
Assistance: Univ.Ass. Dipl.-Ing. Markus Raab

Vienna, 25th June, 2018

Daniel Bugl | Franz Puntigam

# Erklärung zur Verfassung der Arbeit

Daniel Bugl
Breitenfurterstraße 516/2/1
1230 Wien

Wien, 25. Juni 2018

Daniel Bugl

# Acknowledgements

I want to thank Markus Raab and everyone from the Elektra team for their support and reviews during the writing of this thesis. Their ideas and knowledge really helped me shape my thesis and the project resulting from it. I also want to thank my family and friends for supporting me during my study. Finally, I want to thank everyone who participated in the usability test. This thesis would not have been possible without all these people - thank you so much!

# Abstract

Most programs offer a way to tweak their behavior or looks. The process of tweaking is called configuration. The goal of this thesis is to develop a Web User Interface (Web UI) with high usability to remotely configure programs. To achieve better usability compared to existing configuration editors, we use metadata, such as data types, to generate special input fields. To verify that better usability has been achieved, the Web UI was compared to an existing graphical configuration editor, the Elektra QT GUI (QT GUI), in a usability test. The results are promising. Participants were much more confident in their decisions when using the Web UI. Complex scenarios, which resulted in various small mistakes with the QT GUI, were solved correctly with the Web UI.

# Contents

# CHAPTER 1

# Introduction

## 1.1 Motivation

While some programs offer a specifically designed graphical interface to configure them, others only provide configuration files or environment variables. Especially programs running on Linux servers often do not offer a graphical way to configure them, they only provide configuration files.

Configuration via configuration files works fine for advanced users that are familiar with the to-be-configured program. Novice users, however, need to read the documentation and tutorials to figure out how to configure the program (or fail to do so).

Furthermore, it is easy to make mistakes and enter invalid data, which results in bugs or breaks the program. For example, the program might crash when configuration files are not validated properly. Configuration can be messy—different programs use different configuration file formats (JSON, INI, YAML, XML, etc) and store their files in various locations.

## 1.2 Elektra

*Elektra* is a library that implements a hierarchical configuration database [Raa10], the Elektra Key Database (KDB). The KDB allows various programs to be configured in a standardized way. Existing configuration files can be mounted in the KDB and accessed with a simple get/set interface.

In addition to having one standardized system to configure all programs, Elektra offers metadata[1]. Metadata is data that provides information about other data. In Elektra, metadata provides information about *configuration options*. Configuration options are

[1]`http://master.libelektra.org/doc/help/elektra-metadata.md`

key/value pairs that define the configuration of a certain program. Metadata includes a short description, the type of the value and other constraints, such as maximum length, forbidden characters, etc.

### 1.2.1 Keys

Each key in the Elektra KDB can have the following properties (name is the only required property):

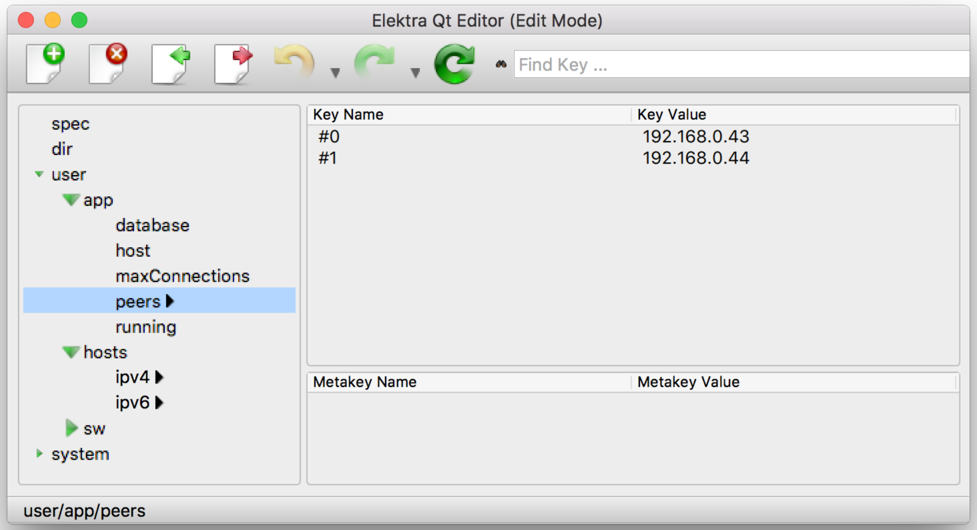
- **Name:** this is the key name in the KDB path. For example, `hosts` in `user/hosts`.
- **Value:** each key can have a value.
- **Subtree:** each key can have a subtree, as in, other keys below it. These keys are called *subkeys*.
- **Metadata:** as described in the previous section, each key can have keys which further describe the key, called *metakeys*.

## 1.3 State of the Art

At the time of writing the following interfaces for Elektra exist:

- a Command-Line Interface (CLI), which can also be used remotely via Secure Shell (SSH)
- a Graphical User Interface (GUI), which cannot be used remotely (without using technologies like X forwarding or a remote desktop application): the Elektra QT GUI (QT GUI)[2]

[2]http://master.libelektra.org/src/tools/qt-gui



| Key Name | Key Value |
|---|---|
| #0 | 192.168.0.43 |
| #1 | 192.168.0.44 |

| Metakey Name | Metakey Value |
|---|---|

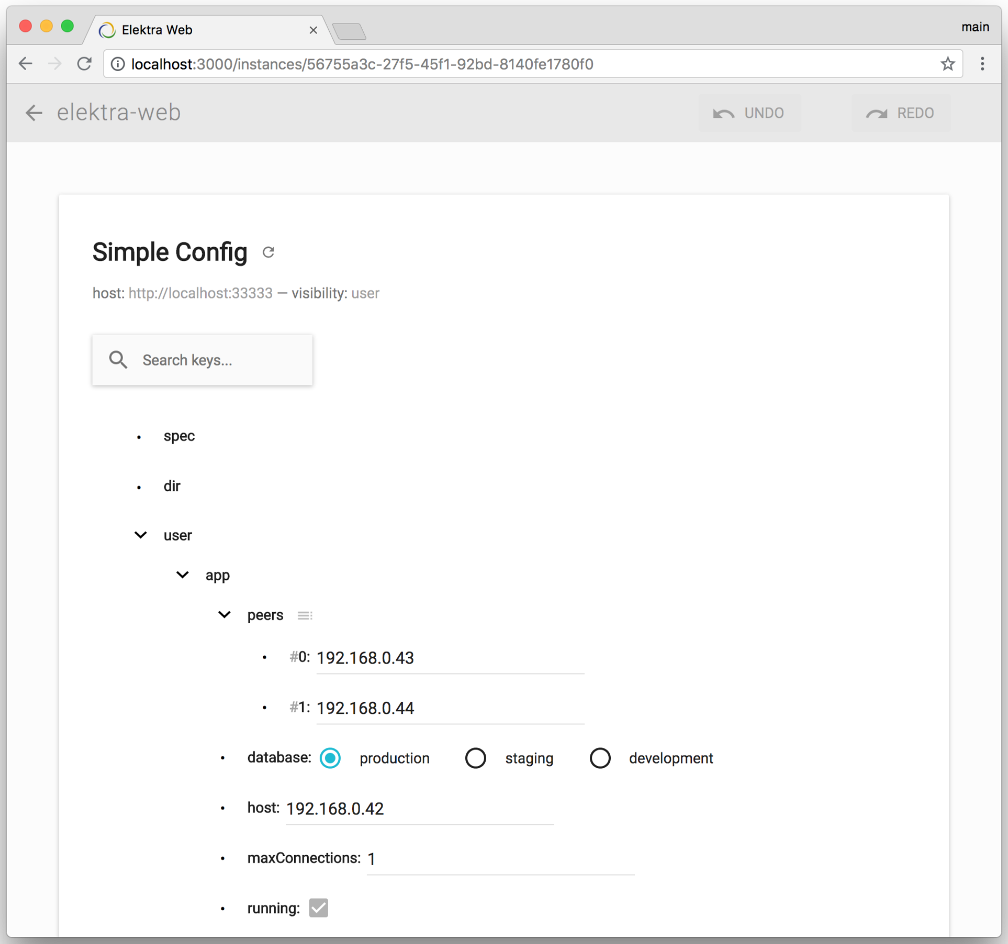
Figure 1.1: A screenshot of the Elektra QT GUI, a graphical configuration editor.

The QT GUI makes it easy to get an overview of all possible configuration options in comparison to the CLI. However, it is still possible to make mistakes in the values. Additionally, users need to consult documentation or tutorials to figure out which configuration options to tweak in order to get a specific result. Furthermore, it is not easily possible to remotely configure Linux servers via the QT GUI, as a running X server (with X forwarding enabled) would be required. This is not feasible for Linux servers, which should be minimal and thus cannot have a graphical environment.

## 1.4 Goal

The aim of this thesis is to implement and evaluate an *interactive tree view*, as part of a Web User Interface (Web UI). The interactive tree view uses metadata to generate a user interface that is easy to use, with self-contained documentation.

Instead of simple text input fields, the interactive tree view displays *special input fields*, as shown in Figure 1.2, to restrict which data can be entered. For example, displaying boolean values as a checkbox or allowing only numbers to be entered. Additionally, the interactive tree view shows short descriptions as tooltips in the interface, so that the user does not need to look up the documentation. Furthermore, the Web UI allows for remote configuration.


Elektra Web
main
localhost:3000/instances/56755a3c-27f5-45f1-92bd-8140fe1780f0
elektra-web
UNDO
REDO
Search keys...

Simple Config

host: http://localhost:33333 — visibility: user

- spec
- dir
- user
  - app
    - peers
      - #0: 192.168.0.43
      - #1: 192.168.0.44
    - database: [✓ production] staging development
    - host: 192.168.0.42
    - maxConnections: 1
    - running: ✓

Figure 1.2: A screenshot of the interactive tree view, with special input fields.

## 1.5 Usability

Due to the wide availability and resulting diversity of the web, usability of web applications is even more important than in other software. Usually, applications belong to a certain ecosystem (for example, QT [qt517], GTK [gno17], Windows [win17] and Mac Applications [mac17] all have certain standards and guidelines, as well as a certain kind of users). Most applications are also used in a certain context, for example, in the office. Web applications, however, can be used *any time* in *any place* and are usually used by a wide variety of people [Kap00]. Furthermore, the Web UI should be easy to use in order to accomplish the users' goals in a simple and efficient manner.

To measure usability [RC08], multiple attributes [Nie93] will be considered:

**Usefulness** Does the product enable users to achieve their goals?

**Efficiency** How fast can goals be accomplished?

**Effectiveness** Does the product behave the way the users expect it to?

**Satisfaction** How does the user feel while using the product?

The usability tests and improvements focus on the interactive tree view, as it is the main part of the Web UI. The challenge here is to visualize various kinds of data (with constraints) and build an interactive user interface from the metadata provided by Elektra—while focusing on usability and making sure the interface does not get too complicated.

Such a solution would be useful in any context that requires modification of complex tree views (for example, Windows Registry Editor, Firefox/Chrome configuration, etc.). Most existing tree views do not validate the data or display fields in a special way, which makes it hard to configure programs without reading the documentation. It is also much easier to make mistakes without validation or special input fields (for example, entering text in a field that should only contain numbers).

## 1.6 Research Question

**RQ0** *How does the interactive tree view of the Web UI compare to the QT GUI in terms of usability?*

To answer this question, the following sub-questions will be answered:

- **RQ1** Which user personas are interested in remote configuration via a Web UI?
- **RQ2** How does the Web UI compare to the QT GUI in amount of errors made, help required and time needed to complete tasks?
- **RQ3** How much more or less satisfied are users with the Web UI in comparison to the QT GUI, measured by qualitative user questioning?

## 1.7 Methodological Approach

At first, a survey will be conducted to find out about the qualities of users who would be interested in remote configuration. The survey ensures that there is a need for such an application (*usefulness*) and hints at possible scenarios and user expectations (**RQ1**).

In order to ensure *efficiency* and *effectiveness*, we conduct usability tests, where potential users are asked to complete tasks to achieve predefined scenarios. While doing so, we collect metrics, such as time spent, help needed and errors made (**RQ2**).

*Satisfaction* will be measured through oral user questioning and qualitative feedback after using the Web UI and the QT GUI (**RQ3**).

### 1.7.1 Threats of Validity

Being involved in a project can lead to wrong assumptions about the users' needs and goals. This means that we cannot only consider *quantitative measures*, because numbers can be interpreted in various ways. Additionally, *qualitative measures*, like oral user questioning, will be conducted. Then, all of the user feedback needs to be evaluated and interpreted by the designer. This feedback will be used to further improve the user interface.

## 1.8 Structure

The thesis consists of 7 chapters:

**Chapter 2: Background** of the interactive tree view.

**Chapter 3: Solution** to the problems mentioned in the introduction.

**Chapter 4: Implementation** overview of the technologies involved in the solution, and why they were chosen.

**Chapter 5: Methodology** describes in detail the approach on how the user interface will be evaluated.

**Chapter 6: Evaluation** contains the results of the survey, usability tests and user questioning.

**Chapter 7: Conclusion** interprets the results and discusses further work.

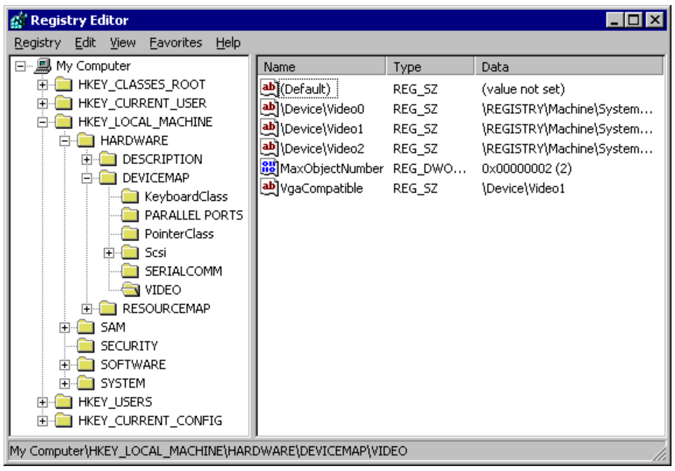
CHAPTER 2

# Background

## 2.1 Existing Configuration Editors

We are going to discuss two configuration editors with tree views: the Windows Registry Editor and the Elektra QT GUI. Furthermore, we are going to cover a configuration editor with special input fields: the Firefox `about:config` page.

### 2.1.1 Windows Registry



| Name | Type | Data |
|---|---|---|
| (Default) | REG_SZ | (value not set) |
| \Device\Video0 | REG_SZ | \REGISTRY\Machine\System... |
| \Device\Video1 | REG_SZ | \REGISTRY\Machine\System... |
| \Device\Video2 | REG_SZ | \REGISTRY\Machine\System... |
| MaxObjectNumber | REG_DWO... | 0x00000002 (2) |
| VgaCompatible | REG_SZ | \Device\Video1 |

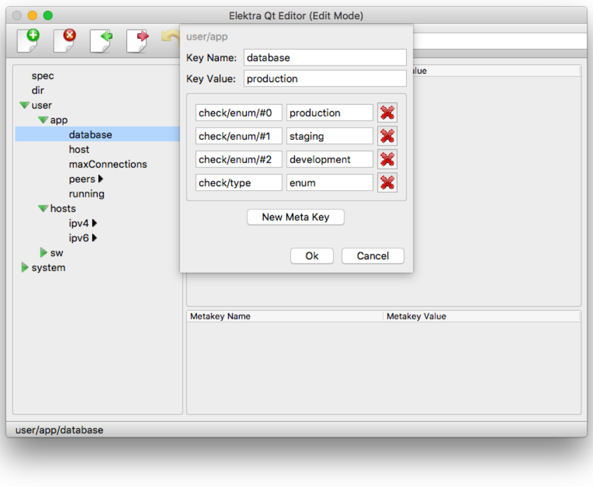
My Computer\HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\VIDEO

Figure 2.1: Registry Editor, the graphical configuration editor for the Windows Registry.

For the Microsoft Windows operating system, there exists a common configuration interface called "Windows Registry" [1]. The Windows Registry offers a GUI, the Registry Editor, as shown in Figure 2.1.

The Windows Registry implements a common configuration interface for all kinds of programs. However, there are too many options and the user has to consult some form of documentation to find out what to edit in order to configure programs properly. It is also not possible to easily get an overview of all configuration options of a certain program, as the options can be scattered across different places in the registry. Furthermore, most fields allow any value to be entered, without validation.

### 2.1.2 Elektra QT GUI



Key Name: database
Key Value: production

| | |
|---|---|
| check/enum/#0 | production |
| check/enum/#1 | staging |
| check/enum/#2 | development |
| check/type | enum |

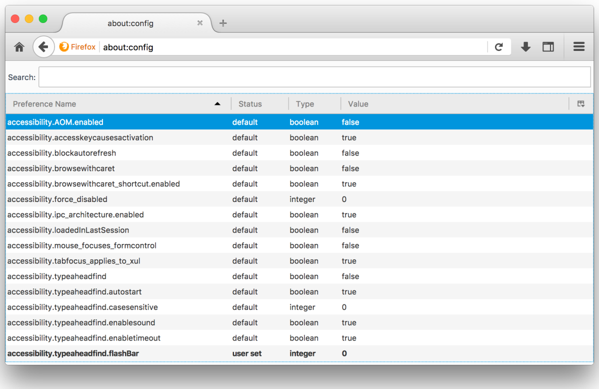
Figure 2.2: The Elektra QT GUI, showing metakeys assigned to the "database" key.

As mentioned earlier, Elektra offers a graphical user interface, the Elektra QT GUI (QT GUI). Elektra provides standards on where to store configuration options. As a result, the QT GUI offers better discoverability compared to the Windows Registry. Additionally, when editing values, the QT GUI displays metadata associated with the key, which serves as a form of self-documentation, as shown in Figure 2.2.

However, all fields are simple text fields, allowing any value to be entered, without validation. There is no way to filter the amount of configuration options presented, resulting in too many options for novice users.

[1] `https://msdn.microsoft.com/en-us/library/windows/desktop/ms724871(v=vs.85).aspx`

### 2.1.3 Firefox Configuration



| Preference Name | Status | Type | Value |
|---|---|---|---|
| accessibility.AOM.enabled | default | boolean | false |
| accessibility.accesskeycausesactivation | default | boolean | true |
| accessibility.blockautorefresh | default | boolean | false |
| accessibility.browsewithcaret | default | boolean | false |
| accessibility.browsewithcaret_shortcut.enabled | default | boolean | true |
| accessibility.force_disabled | default | integer | 0 |
| accessibility.ipc_architecture.enabled | default | boolean | true |
| accessibility.loadedInLastSession | default | boolean | false |
| accessibility.mouse_focuses_formcontrol | default | boolean | false |
| accessibility.tabfocus_applies_to_xul | default | boolean | true |
| accessibility.typeaheadfind | default | boolean | false |
| accessibility.typeaheadfind.autostart | default | boolean | true |
| accessibility.typeaheadfind.casesensitive | default | integer | 0 |
| accessibility.typeaheadfind.enablesound | default | boolean | true |
| accessibility.typeaheadfind.enabletimeout | default | boolean | true |
| **accessibility.typeaheadfind.flashBar** | **user set** | **integer** | **0** |

Figure 2.3: The Firefox `about:config` page, with special input fields.

Firefox offers a special configuration page, `about:config`. Technically, it is not a tree view, because all configuration options are displayed as a simple list. However, `about:config` does offer special input fields depending on the type of data. For example, booleans can be toggled by double clicking the value. This is a great improvement compared to the other configuration editors.

Unfortunately, there are too many configuration options, resulting in a long list. These options are namespaced by a certain syntax, for example, *network.proxy.http*. Elektra already supports interfacing with Firefox configuration. As a result, Firefox configuration could be configured via the interactive tree view of the Web UI.

### 2.1.4 Summary

As we can see, most existing configuration editors only display simple text input fields. These simple input fields allow any text to be entered, usually without validation. As a result, users do not know if the value they entered was correct or not. Additionally, it is not easy to get an overview of all possible configuration options. Having many configuration options is a good idea for advanced users. However, too many options end up cluttering the user interface and harming discoverability of important options for novice users.

## 2.2 Functional Reactive Programming

Instead of expressing the "how" (like traditional, imperative approaches), functional solutions use a declarative approach and allow the programmer to express *what* they want. This sounds especially promising when dealing with graphical user interfaces (GUIs)

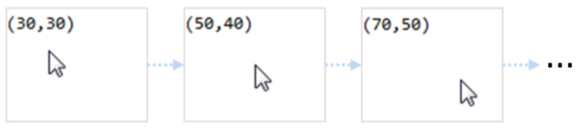


Figure 2.4: Mouse pointer position, an example of a time-varying, reactive value.

[CC13]. A declarative approach to GUIs would mean that we can specify *what* we want to display, instead of *how* to render the user interface.

Functional *reactive* solutions, in addition to being declarative, also describe values that change over time [WTH01]. In order for the user interface to be interactive, it has to *react* to user input. User input could be, for example, the position of the mouse pointer (Figure 2.4), which is a value that changes over time.

To implement an interactive tree view with special input fields, we use a functional reactive approach. By using a declarative approach, we can specify the "what" via metadata to, for example, display special input fields depending on the data type. Through reactiveness, when metadata changes the input field automatically updates itself.

### 2.2.1 Functional Programming

Functional programming is a programming paradigm based on the concept of programs consisting entirely of functions. These functions are very similar to and based on the familiar concept of mathematical functions [Hug89].

#### Imperative vs. Declarative

Functional programming is a *declarative* programming paradigm. Declarative means that programmers specify *what* they want done instead of telling the machine *how* to do it.

An example of *imperative* programming looks as follows:

```
let ns = [1, 2, 3]

let result = []
for (let i = 0; i < ns.length; ++i) {
  result.push(ns[i] * 2)
}

console.log(result) // [2, 4, 6]
```

As we can see, the imperative example tells the machine to:

- create an empty array
- loop through the `ns` array
- multiply each number by 2 and add it to the `result` array

We could write the same example, in a more *declarative* way, as follows:

```
let ns = [1, 2, 3]

let result = ns.map(n => n * 2)

console.log(result) // [2, 4, 6]
```

The first thing we notice is that the declarative example is much more concise. We are making use of the `map` function, which is a function that takes a function as its argument and applies that function to all elements of the array. In the declarative example, we are defining a function that takes a number and multiplies it by two (`n => n * 2`). This function is then applied to every element of the array, returning a new array with the results of each function application. We are using a function to apply another function to all elements of the array. As implied by the name, functions are heavily used in functional programming [BW88].

The declarative example also does not give the machine many instructions, and in a more functional programming language than JavaScript, we would not be giving any instructions on the "how".

### Pure Functions

Another important concept in functional programming are *pure functions*. Given the same input, pure functions always return the same output. This makes them very predictable [LPJ94].

An example of an *impure* function would be:

```
let i = 0

function increment () {
  return ++i
}

increment() // 1
increment() // 2
increment() // 3
```

As we can see, giving the same input (no input) twice, gives us different results. This happens because the function accesses global state, a side effect which makes it impure. To make this function *pure*, we would have to pass `i` as an argument to it:

```
function increment (i) {
  return i + 1
}

increment(0) // 1
increment(1) // 2
increment(0) // 1
```

In the example above, giving the same input (`0`), always results in the same output (`1`). This function is pure, making it very predictable.

#### Reactivity

Now that we have an understanding of what the "functional" part in "Functional Reactive Programming" means, we are going to have a closer look at the "reactive" part.

Functional programming builds programs that only consist of functions [Hug89]. These functions can accept static values as input, and return a static value as output. However, if functions were limited to processing static values only, functional programming would not be very useful in the real world. In the real world, we need programs that *react* to user input, such as the user pressing a button in our program.

### 2.2.2 Functional Reactive Animation

To model reactiveness, Functional Reactive Animation (Fran) introduces time-varying, reactive values, called *behaviors*, to functional programming [EH97]. Behaviors can be anything that changes over time, like the mouse pointer position, the user clicking on something, or typing on the keyboard. However, behaviors are not limited to user input, even animations, timers, network requests and many more operations can be modeled with behaviors.

#### Modeling over Presentation

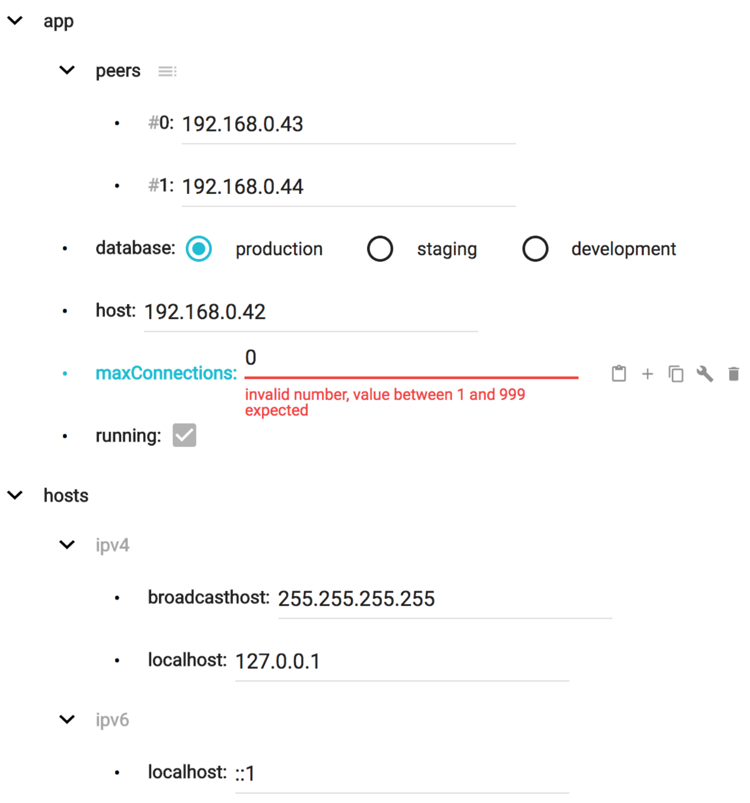
The Fran paper [EH97] discusses the benefits of a modeling approach to animation over a presentation approach. Modeling is a declarative approach, allowing programmers to specify the "what" of an interactive animation. Presentation deals with *how* the animation is presented.

As we can see, the goal of the Fran paper was to apply a declarative approach to animation, or more generically speaking: Reactivity. It turns out that the same approach can be applied to much more than just animations, such as physics simulations, which are discussed in detail in the paper [EH97].

CHAPTER 3

# Solution

We are now going to discuss the goals of the interactive tree view and solutions to the aforementioned problems.

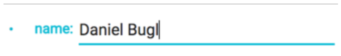
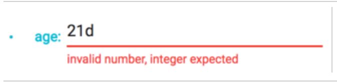
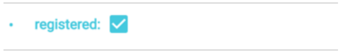
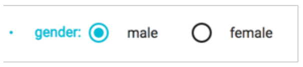


app
- peers
  - #0: 192.168.0.43
  - #1: 192.168.0.44
- database: [✓ production] staging development
- host: 192.168.0.42
- maxConnections: 0
  invalid number, value between 1 and 999 expected
- running: ✓

hosts
- ipv4
  - broadcasthost: 255.255.255.255
  - localhost: 127.0.0.1
- ipv6
  - localhost: ::1

Figure 3.1: A screenshot of the interactive tree view in the Elektra Web UI.

Table 3.1: Special input fields provided by the Web UI

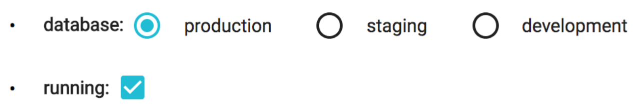
| Name | Screenshot | Metadata |
|---|---|---|
| text |  name: Daniel Bugl | n/a |
| number |  age: 21d invalid number, integer expected | `check/type = short` or `long`, etc |
| checkbox |  registered: ✓ | `check/type = boolean` |
| radio |  gender: ● male ○ female | `check/type = enum` |

## 3.1 Goal 1: Avoid Easy to Make Mistakes

When using configuration files, there is no way of knowing what to edit without consulting some form of documentation. Documentation comes in the form of a wiki, a README file, or comments in the configuration file. Because any text can be entered in a configuration file, it is easy to enter invalid values or produce a completely invalid configuration file.

In most existing configuration editors, there is no validation of the values. Some editors offer data types, which improves the user experience a little by giving users a hint of what they might be supposed to enter in the input field. However, in most configuration editors there is no feedback if the entered value is correct.

### 3.1.1 Solution 1: Special Input Fields



database: ● production ○ staging ○ development

running: ✓

Figure 3.2: Special input fields, implemented by the Web UI.

To prevent users from making mistakes when entering values, the Web UI implements special input fields, as shown in Table 3.1. Through metadata it is possible to know which kind of data should be entered. For example, instead of manually entering "yes" or "no" (or "true"/"false"), the Web UI shows a checkbox. If there are multiple possible values, radio buttons are displayed, as shown in Figure 3.2.

Elektra and the Web UI provide validation for data types (such as only numbers) as well as string validation via regular expressions (regex) [Fri02]. Regex validation can be defined by setting the `check/validation` metakey. The results of the validation process are displayed next to the corresponding input fields, so that the user knows what went wrong. String validation can be seen in Figure 3.1 at the beginning of this chapter. The Web UI also makes it possible to restrict the removal (`restrict/remove` metakey) or editing of keys (`restrict/write` metakey).

Additionally, Elektra provides further validation by describing complex relationships between values. For example, a host cannot be entered if the service has not been enabled yet. When complex validation fails, an error message is displayed in the user interface, as shown in Figure 3.3. Furthermore, Elektra validation is extensible, users might add plugins to achieve various kinds of complex validation.

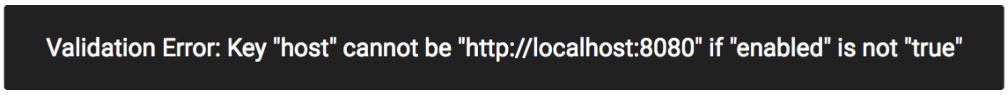
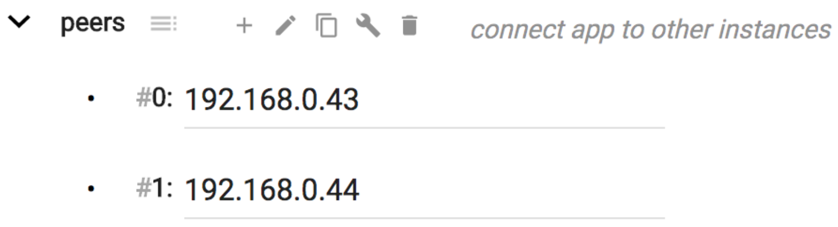

Validation Error: Key "host" cannot be "http://localhost:8080" if "enabled" is not "true"

Figure 3.3: Complex validation, implemented by Elektra and the Web UI.

## 3.2 Goal 2: Improve Discoverability

Configuration files require the user to consult some form of documentation to figure out which configuration options are possible. Graphical configuration editors provide better discoverability by displaying all possible configuration options. However, most editors do not provide descriptions or special input fields for the values. As a result, users still need to consult some form of documentation to be able to configure programs.

### 3.2.1 Solution 2: Visualization of Configuration Options


peers

connect app to other instances

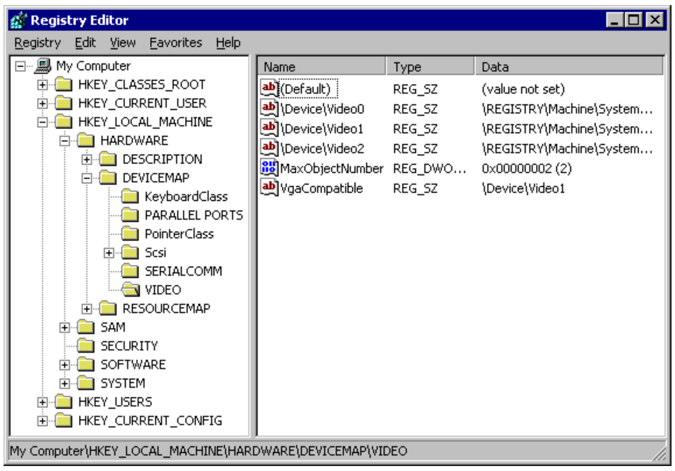
- #0: 192.168.0.43
- #1: 192.168.0.44

Figure 3.4: Documentation in the interactive tree view, implemented by the Web UI.

To improve discoverability, the Web UI displays an interactive tree view, which can be browsed to get an overview of possible configuration options. Next to configuration options, a short description is displayed, as shown in Figure 3.4.

Most existing configuration editors already have decent discoverability. However, the Web UI provides more information about configuration options by displaying special input fields and a description.

## 3.3 Goal 3: Ensure Complex Tree Views Do Not Get Confusing



```
My Computer
  HKEY_CLASSES_ROOT
  HKEY_CURRENT_USER
  HKEY_LOCAL_MACHINE
    HARDWARE
      DESCRIPTION
      DEVICEMAP
        KeyboardClass
        PARALLEL PORTS
        PointerClass
        Scsi
        SERIALCOMM
        VIDEO
      RESOURCEMAP
    SAM
    SECURITY
    SOFTWARE
    SYSTEM
  HKEY_USERS
  HKEY_CURRENT_CONFIG
```

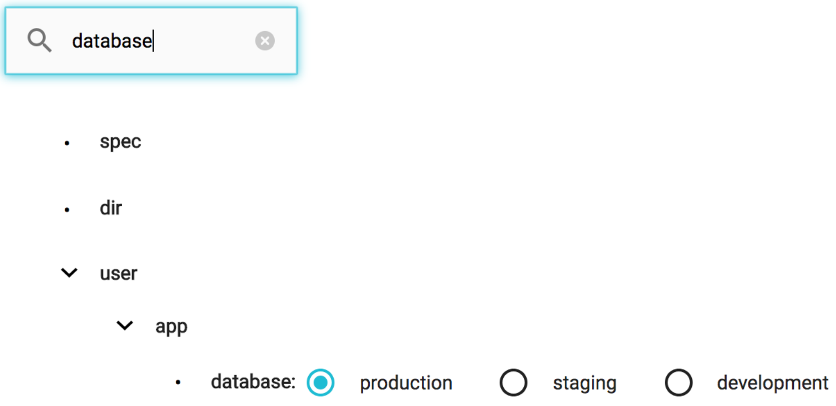
| Name | Type | Data |
|---|---|---|
| (Default) | REG_SZ | (value not set) |
| \Device\Video0 | REG_SZ | \REGISTRY\Machine\System... |
| \Device\Video1 | REG_SZ | \REGISTRY\Machine\System... |
| \Device\Video2 | REG_SZ | \REGISTRY\Machine\System... |
| MaxObjectNumber | REG_DWO... | 0x00000002 (2) |
| VgaCompatible | REG_SZ | \Device\Video1 |

My Computer\HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\VIDEO

Figure 3.5: Registry Editor, a configuration editor with a complex tree view.

Configuration editors usually display a tree view of configuration options. However, most programs have a lot of configuration options. Some of these options are only used internally and are not meant to be edited by the user. As a result, complex tree views can get confusing and lead to worse discoverability of the options that matter to the user, as shown in Figure 3.5.

### 3.3.1 Solution 3.1: Interactive Tree View



database

- spec
- dir
- user
  - app
    - database: [✓ production] staging development

Figure 3.6: Filtering the tree view via search, implemented by the Web UI.

To prevent complex tree views from confusing the user, in the Web UI keys/paths can be filtered via a search input field, as shown in Figure 3.6. As a result, configuration options can be filtered to only display options related to one program, or even just one feature of a program.

### 3.3.2 Solution 3.2: Visibility Levels

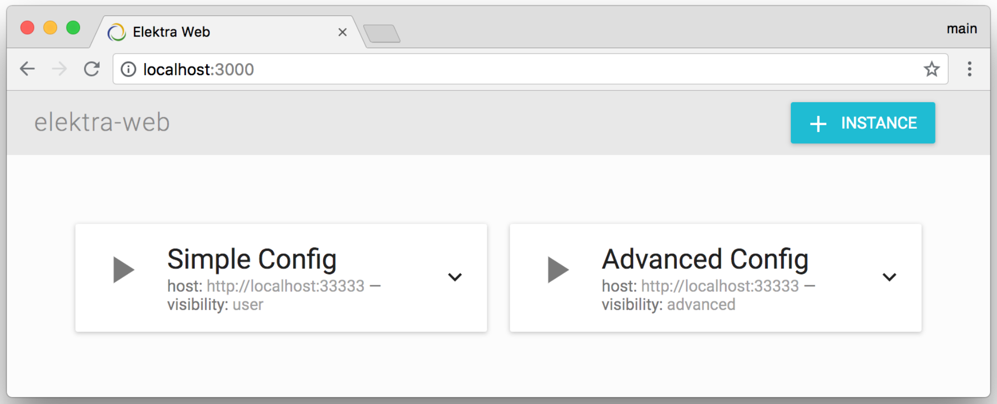


Figure 3.7: Instances with different visibility levels, implemented by the Web UI.

The Web UI implements visibility levels, as shown in Figure 3.7. For example, by tagging keys as "advanced", they will be hidden from novice users, and only shown to users who haven chosen to see the "advanced" visibility level. Through visibility levels, it is possible to keep the tree view simple for novice users, while still offering all configuration options for advanced users.

## 3.4 Goal 4: Unify Configuration Formats

Configuration files come in various different formats, placed all across the filesystem. While there are some platform-specific standards on where to place configuration files, configuration file locations are not standardized across platforms. Furthermore, even on a single platform, there are often multiple possible locations for configuration files.

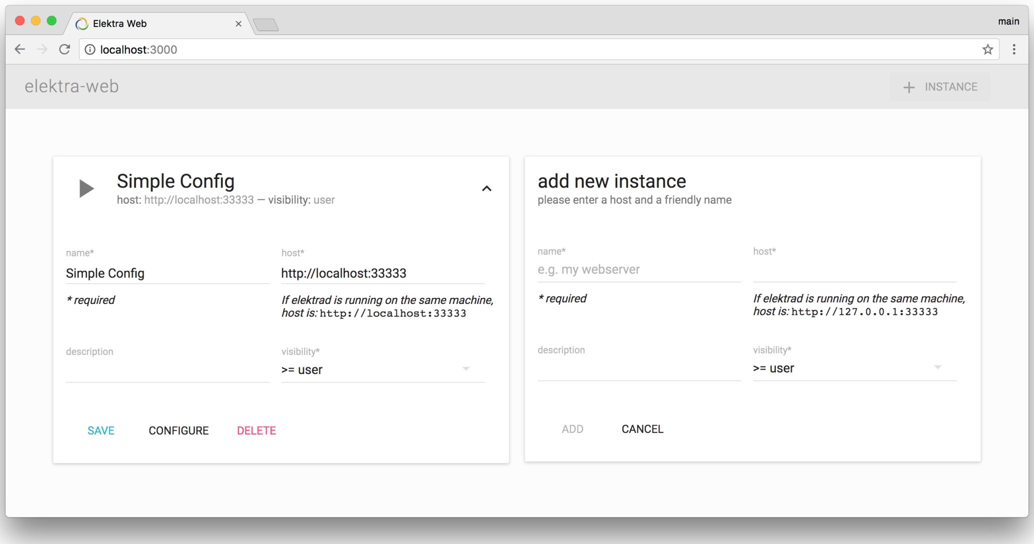
### 3.4.1 Solution 4: Standardized Configuration Interface

To support various different configuration formats, the Web UI is based on Elektra. Elektra provides an interface that allows various programs to be configured in a standardized way. Existing configuration files can be mounted and accessed with a simple get/set interface.

## 3.5 Goal 5: Remote Configuration

While Elektra achieves Goal 4 by providing a standardized and cross-platform way to store and access configuration, there is currently no way to remotely configure machines (with a GUI). The Web UI supports connecting to multiple instances via one interface. Multiple instances that connect to the same host with different visibility levels can also be created, as shown in Figure 3.8.

### 3.5.1 Solution 5: Remote Configuration via Elektra Web



Simple Config
host: http://localhost:33333 — visibility: user

name*: Simple Config
host*: http://localhost:33333
* required
If elektrad is running on the same machine, host is: http://localhost:33333
description:
visibility*: >= user

add new instance
please enter a host and a friendly name

name*: e.g. my webserver
host*:
* required
If elektrad is running on the same machine, host is: http://127.0.0.1:33333
description:
visibility*: >= user

Figure 3.8: Remote configuration, implemented by Elektra Web and the Web UI.

Elektra Web consists of multiple components: One is placed on the to-be-configured machine, the other is placed on a web server and serves a web interface (the Web UI). As a result, machines can be remotely configured with a graphical user interface via a web browser, as shown in Figure 3.8.

CHAPTER 4

# Implementation

## 4.1 Overview

*Elektra Web* consists of four components:

- the Web UI with an interactive tree view
- a server (to be configured) running an Elektra daemon (elektrad)
- a relay daemon (webd) to communicate with the elektrads and serve the Web UI
- a client (web browser) that accesses the Web UI on the webd

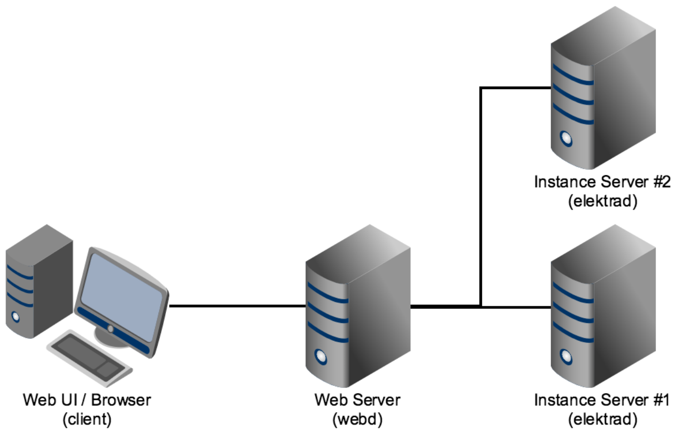


Figure 4.1: A deployment diagram of all Elektra Web components.

## 4.2 elektrad

elektrad is a server that provides an Hypertext Transfer Protocol (HTTP) Application Programming Interface (API) to access Elektra remotely.

elektrad communicates with the KDB by turning API requests into commands that access the KDB. Afterwards, the result of the command is turned into a JSON format and returned as a response of the API request.

### 4.2.1 API Example

An HTTP request to get the app key from elektrad looks as follows:

```
GET /kdb/user/app
```

And results in the following JSON response:

```
{
  "exists": true,
  "name": "app",
  "path": "user/app",
  "ls": [
    "user/app",
    "user/app/database",
    "user/app/host",
    "user/app/maxConnections",
    "user/app/peers",
    "user/app/peers/#0",
    "user/app/peers/#1",
    "user/app/running"
  ]
}
```

If a key has a value or metadata, these properties are also contained in the JSON response.

For example, the following HTTP request:

```
GET /kdb/user/app/database
```

Results in the following JSON response:

```
{
  "exists": true,
  "name": "database",
  "path": "user/app/database",
  "ls": [ "user/app/database" ],
  "value": "production",
  "meta": {
      "check/enum/#0": "production",
      "check/enum/#1": "staging",
      "check/enum/#2": "development",
      "check/type": "enum"
   }
}
```

The metadata, stored in the `meta` key, is used to generate special input fields. In this example, the metadata would turn the text input field into radio buttons with the options "production", "staging" and "development".

### 4.2.2 API Documentation

- API documentation: `http://docs.elektrad.apiary.io/`
- API blueprint: `http://master.libelektra.org/doc/api_blueprints/elektrad.apib`

By default, elektrad serves its API on `http://localhost:33333`.

## 4.3 webd

The relay daemon (webd) for Elektra Web accesses the API of elektrad instances and provides an API to manage them.

Essentially, webd works like a relay, it forwards API requests to the elektrad instances. The idea is to have a central access point which the client can connect to. Furthermore, webd serves the client on the same port as the API. This makes it possible to send requests from the client to webd without having to deal with Cross-Origin Resource Sharing (CORS) security restrictions.

### 4.3.1 API Example

An HTTP request to get the `app` key from an instance looks as follows:

```
GET /api/instances/:instance_id/kdb/user/app
```

The request gets relayed to the elektrad instance, then webd returns the same result as elektrad:

```
{
   "exists": true,
   "name": "app",
   "path": "user/app",
   "ls": [
      "user/app",
      "user/app/database",
      "user/app/host",
      ...
    ]
}
```

### 4.3.2 API Documentation

- API documentation: `http://docs.webd.apiary.io/`
- API blueprint: `http://master.libelektra.org/doc/api_blueprints/webd.apib`

By default, webd serves its API and the client on `http://localhost:33334`.

## 4.4 Web UI

The Web UI for Elektra Web is created using the React[1] and Redux[2] libraries. The Web UI accesses the API that webd provides. The Web UI is served by webd.

### 4.4.1 Redux

The Web UI uses functional concepts, like pure functions, to manage the application state [LPJ94]. Redux protects the application state, ensuring that it can only be changed through actions (which are simple JavaScript objects with a `type` property that defines the action name). These actions get processed by pure functions (`(state, action) => state`). As a result, state changes are easy to test and debug [Abr18]. These concepts are used in the JavaScript library *Redux*. Furthermore, Redux decouples state-changing logic from user interface (UI)-logic, which makes adjusting the API simple.

### 4.4.2 React

The Web UI uses *React* to render the user interface. One of the reasons for this decision is that Redux works very well with React. Furthermore, React also uses functional concepts to render the UI - *React components* can be pure `render` functions that accept (a part of) the application state (also called *properties* in React-terms) and return a so-called virtual Document Object Model (DOM) [Psa08], which is a JavaScript object that describes the to-be-rendered user interface. Before rendering to the DOM, diffing algorithms are used to efficiently re-render only the parts of the user interface that changed [Inc18].

[1] `https://reactjs.org/`
[2] `http://redux.js.org/`

CHAPTER 5

# Methodology

## 5.1 Initial Survey

To ensure a product is useful, it is important to design with a certain user persona in mind [Nor13]. To find out about the qualities of a user that would be interested in remote configuration with a GUI, a survey will be conducted. Afterwards, the answers to a question about remote configuration scenarios ("In which remote configuration scenarios would you be interested?") will be correlated with the answers to the other questions in the survey, for example, "How important is it to expose configuration options in the following ways? Web UI, CLI, etc.". The survey will hint at possible scenarios and user expectations. A user persona and use cases will be created from the results of the survey.

The survey questions relevant for this thesis were part of a general Elektra survey, which was conducted from *June 20th, 2016* to *July 18th, 2016*.

The participants that answered "via a web interface" to the following question were inspected more closely to create potential user groups for the Web UI:

*Imagine you have one or multiple machines that you want to configure/setup without having physical access. In which remote configuration scenarios would you be interested?*

- via a web interface
- via ssh
- in groups/clusters of devices with similar configuration (e.g. multiple instances of web servers)
- in non-clustered devices with independent configurations
- in a single computer (e.g. configuring your home server)
- not interested

## 5.2 Usability Inspection

After the first prototype of the Web UI has been developed, a usability inspection will be conducted to ensure that the whole user experience is consistent. This will also eliminate the main issues with the user interface before the actual usability tests are carried out. For this usability inspection, the cognitive walkthrough [Nie94] method will be used. In this method, the earlier defined use cases are analyzed, then tasks are created and broken up into multiple steps. While walking through these tasks, four questions [Nie94] will be asked during each step:

- Will the user try to achieve the effect that the subtask has? For example: Does the user understand that this subtask is needed to reach the user's goal?
- Will the user notice that the correct action is available? For example: Is the button visible?
- Will the user understand that the wanted subtask can be achieved by the action? For example: The right button is visible but the user does not understand the text and will therefore not click on it.
- Does the user get appropriate feedback? For example: Will the user know that they have done the right thing after performing the action?

Additionally, a small number of people (who are already familiar with Elektra) will review the application and their feedback will be implemented before the first usability test. All in all, this initial evaluation should eliminate most rough edges in the Web UI, so that the usability tests can focus on deeper usability issues.

## 5.3 Usability Test

The Web UI will be iteratively designed [May99] and improved through user feedback to achieve high usability.

Usability tests [DR99], where potential users are asked to complete tasks to achieve predefined scenarios, will be conducted. Some of these scenarios will be target-based, for example, the user has to achieve a certain goal, others will be explorative, for example, the user tries to get an overview over configuration options of a program. For such a test, it will be important not to introduce bias by, for example, letting people configure a program that they already know about. Thus, fictive configuration options will be developed to ensure that nobody knows about the configuration options before starting the test. During the usability tests, metrics (such as time spent, help needed and errors made) while completing tasks are collected.

Scenarios that caused many errors, took a long time or confused the participant will be discussed after the test through oral user questioning. All feedback will be used to improve the user interface in the next iteration.

The study will be conducted with a working prototype of the Web UI and the latest version of the QT GUI (Elektra 0.8.23 release), in a controlled environment (lab), with participants that have at least basic knowledge in system administration. Within-subject design [KL14] will be used to minimize individual bias and the number of people required for the tests. In order to reduce bias from learning effects, latin-square design [Win62] will be used to distribute the tasks. A comparison between the interactive tree view and the Elektra QT GUI will be made to ensure usability has improved. Furthermore, observations during the test will be noted and qualitatively evaluated to further improve the usability of the interface.

During usability tests, time to complete the tasks will be measured. Additionally, we note if the task was completed successfully (Y), required help from the test moderator (H) or was not completed successfully (N). The total for this metric is in the following format: Y/H/N.

### 5.3.1 Within-Subject Design

In a within-subject design usability test, all participants receive all variants of the test [CGK12]. The usability test will have two variants:

- the Web UI
- the QT GUI

Each participant will complete all scenarios with both variants. As a result of this design, there might be learning effects, because the participant already solved the same scenarios in a similar application. Learning effects will be counteracted via latin-square design.

### 5.3.2 Latin-Square Design

Latin-square design changes the order in which variants are tested for each test person. Latin-square design is useful to counterbalance immediate sequential effects, such as learning effects from using a similar program [Bra58]. Latin-square design can be used if there is an even number of variants.

Since the usability tests will have two variants, latin-square balancing will be quite simple: Half of the participants test the Web UI first, and the other half test the QT GUI first.

Each scenario is turned into a task. The tasks will also be latin-square balanced [Bra58] to minimize bias from learning effects, as shown in Table 5.1.

### 5.3.3 Context

In order to have a context that the test participants are already familiar with, the usability test simulates managing a university course via Elektra. Participants will have to do common tasks like editing student data or adding grades. This is a form of configuration

Table 5.1: Task balancing per test participant (TP)

| | **Task 1** | **Task 2** | **Task 3** | **Task 4** | **Task 5** | **Task 6** |
|---|---|---|---|---|---|---|
| **TP0 (Pilot Test)** | A | B | C | D | E | F |
| **TP1** | A | B | F | C | E | D |
| **TP2** | B | C | A | D | F | E |
| **TP3** | C | D | B | E | A | F |
| **TP4** | D | E | C | F | B | A |
| **TP5** | E | F | D | A | C | B |
| **TP6** | F | A | E | B | D | C |

for the programs that process student data. Furthermore, the tasks cover all of the usability improvements of the Web UI.

### 5.3.4 Roles

There will be 3 people involved in each test:

- Test moderator
- Observer
- Participant

### 5.3.5 Equipment

Sessions will be completed on a laptop with an external mouse and keyboard. Screen recording and audio recording software will be set up on the laptop.

### 5.3.6 Duration

Each session is expected to last around 40 minutes (20 minutes per variant), with a 20 minute break in-between sessions.

### 5.3.7 Schedule

Friday, April 27th 2018.

## 5.4 Expected Results

The interactive tree view should lead to better discoverability compared to existing configuration editors (like the QT GUI) or using a text editor to modify configuration files manually. Unlike in a text file, where the user would have to consult the documentation to figure out which configuration options are possible, the user can now browse an interactive tree view, with descriptions added to the keys, as well as special input fields that give hints about possible values (for example: only numbers, yes/no checkbox).

These special input fields also restrict values that can be entered (for example: only numbers), which should reduce the amount of errors a user makes while configuring a program. In existing configuration editors, it is easy to make mistakes (for example: text instead of number). Sometimes there are multiple ways to write a value, for example, boolean values can be written as yes/no, true/false, 0/1, on/off, enabled/disabled and so on. In the interactive tree view, such a field would be displayed as a checkbox.

Overall, the interactive tree view should improve the user experience during configuration of various programs.

**Hypothesis 1:** The interactive tree view leads to better discoverability of configuration options.

*Independent Variables:* Design of the user interface (QT GUI versus Web UI)

*Dependent Variables:* Time required to find options, impression of how hard it was to find an option

*Split:* n/a

*Filter:* all tasks completed

**Hypothesis 2:** The special input fields reduce the amount of errors a user makes, and help needed.

*Independent Variables:* Design of the field (simple text input versus special input fields)

*Dependent Variables:* Amount of errors made, participant's confidence that the change made was correct

*Split:* n/a

*Filter:* all tasks completed

**Hypothesis 3:** The Web UI improves the user experience (satisfaction).

*Independent Variables:* Design of the user interface (QT GUI versus Web UI)

*Dependent Variables:* User experience according to oral user questioning

*Split:* n/a

*Filter:* n/a

## 5.5 Variables

**Variable 1:** Survey responses

*Role in the study design:* dependent variable

*Type of characteristic attribute:* discrete

*Scale niveau:* ratio scale

*Measurement:* user survey

**Variable 2:** Design of the user interface (QT GUI versus Web UI)

*Role in the study design:* independent variable

*Type of characteristic attribute:* discrete

*Scale niveau:* nominal scale

*Measurement:* predefined

**Variable 3:** Time required to find options

*Role in the study design:* dependent variable

*Type of characteristic attribute:* continuous

*Scale niveau:* ratio scale

*Measurement:* logging during usability test

**Variable 4:** Impression of how hard it was to find an option

*Role in the study design:* dependent variable

*Type of characteristic attribute:* discrete

*Scale niveau:* ordinal scale

*Measurement:* observation during the usability test

**Variable 5:** Amount of errors made and help needed

*Role in the study design:* dependent variable

*Type of characteristic attribute:* discrete

*Scale niveau:* ratio scale

*Measurement:* logging during usability test

**Variable 6:** Participant's confidence that the change made was correct

*Role in the study design:* dependent variable

*Type of characteristic attribute:* discrete

*Scale niveau:* ordinal scale

*Measurement:* oral user questioning

**Variable 7:** User experience and satisfaction

*Role in the study design:* dependent variable

*Type of characteristic attribute:* continuous

*Scale niveau:* ratio scale

*Measurement:* oral user questioning

**Variable 8:** Current condition of the user (mood and existing knowledge)

*Role in the study design:* potential confounding variable

*Type of characteristic attribute:* continuous

*Scale niveau:* ratio scale

*Measurement:* n/a

**Variable 9:** Fictive configuration options used in the tests

*Role in the study design:* controlled variable

*Type of characteristic attribute:* n/a

*Scale niveau:* n/a

*Measurement:* n/a

CHAPTER 6

# Evaluation

## 6.1 Initial Survey

672 persons visited the survey, 286 started to answer, 162 completed the questionnaire. The age of the persons (220 responses) has a mean of 32 years (standard deviation = 9). Participants were from Germany (50), Austria (41), United States (32), France (25), Australia (9), and 31 other countries (85). The reported degrees of the participants (244 responses) are: master (38 %), bachelor (25 %), student (18 %), no degree (13 %), or PhD (6 %). As occupation, 56 % of the persons selected software developer, 21 % administrator, and 16 % researcher (multiple choice question, 286 responses).

Of all 286 participants who started to answer, 52 persons were interested in remote configuration via a Web UI (18.18 %). In this group of participants, the reported occupations were: 39 software developers, 14 administrators, 14 students and 13 researchers. The large amount of software developers comes from the fact that the survey was targeted at software developers.

On a likert scale of 1-5 (not important, slightly important, moderately important, important, very important), the participants interested in remote configuration rated (on average) having a Web UI for configuration as:

- the developers: "slightly important" (mean = 2.47, median = 2)
- the administrators: "moderately important" (mean = 2.93, median = 3)
- the students: "slightly important" (mean = 2.15, median = 2)
- the researchers: "moderately important" (mean = 2.75, median = 3)

To create accurate personas, we also looked at average ages of these groups:

- the developers: mean = 33.17 (median = 32)
- the administrators: mean = 34 (median = 29)
- the students: mean = 24.38 (median = 25)
- the researchers: mean = 30.45 (median = 28)

Another result of the survey was that cluster configuration (groups of machines with similar configuration) did not seem too important. Of all 173 participants who responded to the question about remote configuration, only 38.15 % (66) were interested in cluster configuration. However, 53.18 % (92) were interested in configuration of single machines or multiple non-clustered machines. As a comparison, 80.35 % (139) were interested in configuration via SSH.

### 6.1.1 Personas

User personas represent major user groups for a product [CS00]. From the results of the survey, user personas were created. While designing the Web UI, these personas and the resulting use cases were considered.

#### Andy the Administrator

- **Age:** 34
- **Job:** Owner of a coworking space
- **Environment:** He is proficient in dealing with servers and already uses a CLI to configure his server. His server runs a service that is used to manage events and participants. He uses this service for his coworking space.
- **Goal:** He wants to use the Web UI to make quick adjustments to the service running on his server.
- **Tasks:**
  - Editing the configuration of a service
  - Editing data of a service (for example, event data)
  - Adding data to the service (for example, inserting an event)
  - Removing data from the service (for example, deleting an event)

**Marie the Researcher**

- **Age:** 28
- **Job:** Manages a university course
- **Environment:** She is proficient in dealing with computers. She uses Elektra to manage student and exam information for her university course.
- **Goal:** She wants to use the Web UI to manage her university course.
- **Tasks:**
  - Grading a student
  - Correcting the grade of a student
  - Removing a grade from a student
  - Editing exam information

### 6.1.2 Use Cases

Each persona has different goals and tasks they want to solve using the Web UI. From these tasks, we constructed generic use cases that the Web UI needs to accommodate for. The use cases can be found in the Elektra repository[1].

## 6.2 Scenarios

The administrator persona already uses a CLI to configure servers. We thus focussed on the researcher persona to create scenarios for the usability test from the use cases. However, the administrator persona was still considered while designing the user interface.

### 6.2.1 Scenario A: Updating the Grade of a Student

A student from the previous semester (WS2017) improved their grade (grades section).

Change the final grade of student 00000010 to "sehr gut".

### 6.2.2 Scenario B: Deleting a Student from the Database

A student decided not to take the course anymore.

Remove student 00000049 from the database by removing it from the students and grades sections of the current semester (SS2018).

[1] `http://master.libelektra.org/doc/usecases/elektra_web`

### 6.2.3 Scenario C: Storing Tutors in the Database

Create a new tutors key in the current semester (SS2018) and enter the following content: Marie Musterfrau, Max Mustermann

Define a regex validation (`check/validation`) for the tutors key you created. The regex should look as follows:

```
^((.*),? ?)*$
```

Also restrict the removal of the tutors key (`restrict/remove`).

### 6.2.4 Scenario D: Grading an Assignment

You just had an assignment discussion in the current semester (SS2018) and want to grade a student (grades section).

The student has the following student number ("Matrikelnummer"): 00000025. In the grades section, duplicate the exam0 key to create a new key named assignment0. The student received 80 points on the assignment.

### 6.2.5 Scenario E: Storing Tutors in the Database with Array

Create a new array key named tutorsArray in the current semester (SS2018). Insert the following two sub keys: Marie Musterfrau, Max Mustermann

### 6.2.6 Scenario F: Creating a New Exam

You are tasked with the organization of a repetition of the entrance exam.

Create a new exam in the current semester (SS2018) by duplicating the "Entrance Exam". Rename it to "Repetition of Entrance Exam" and give it a new ID "exam4". We are only going to need lab #0 for this exam, so disable the other labs. The date for this exam is next Monday (30.04.2018). Make sure the ID is write restricted.

## 6.3 Usability Test

### 6.3.1 Pilot Test

Before conducting the actual usability test, it is important to test the usability test with a pilot test. A pilot test irons out issues with the usability test itself as well as issues with the software that have not been noticed yet [RC08].

#### Learning 1: Focus on Interactive Tree View

One of the main learnings of the pilot test was that cluster configuration, which was the original focus of this thesis, confused users. First of all, the whole concept of configuring clusters of servers was confusing. Furthermore, the way creating clusters worked was even more confusing, as shown in Figure 6.1. Users were supposed to enter a cluster name, then select which instances to add to the cluster, and then press the "create cluster" button. However, most users ended up putting the name in and clicking the button immediately (it was placed right next to the input field for the name). This resulted in an empty cluster being created, confusing users even further about the feature.

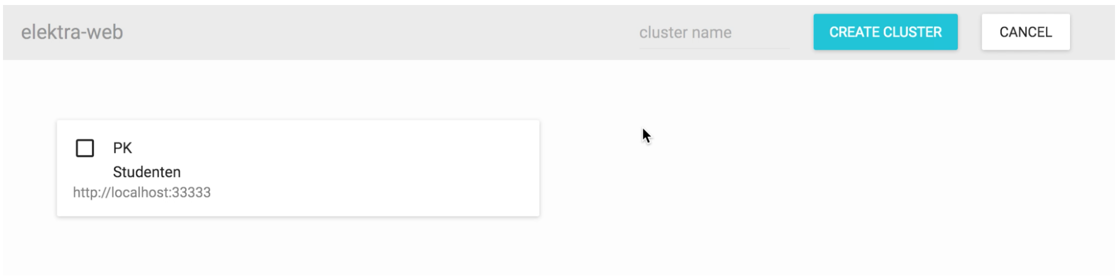


Figure 6.1: The old Elektra Web UI (v1.0), with cluster configuration.

As a result, the focus of this thesis pivoted to the interactive tree view. The overview is now simply a list of instances, as shown in Figure 6.2.

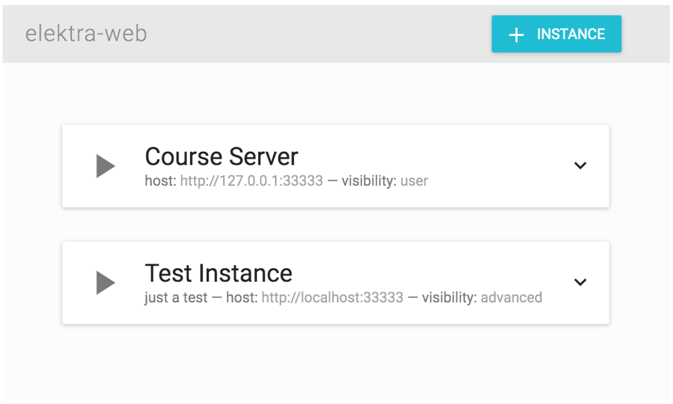
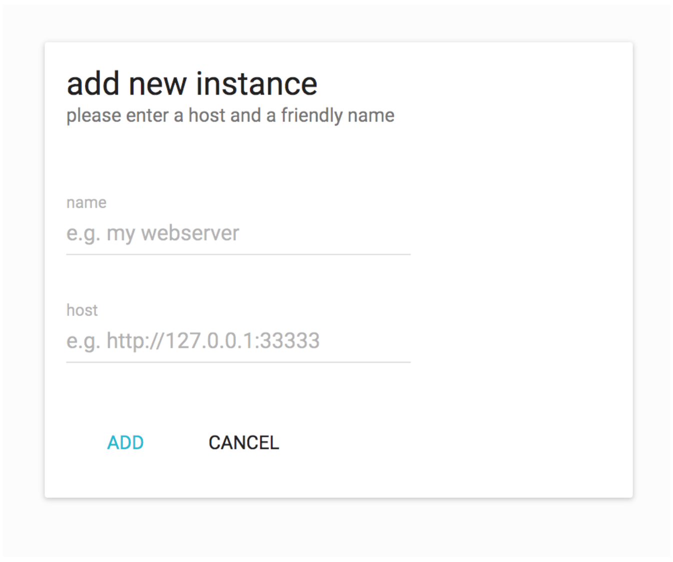

elektra-web
INSTANCE
Course Server
host: http://127.0.0.1:33333 — visibility: user
Test Instance
just a test — host: http://localhost:33333 — visibility: advanced

Figure 6.2: A later release of the Elektra Web UI (v1.5), focussing on instance configuration.

### Learning 2: Single Instance Mode

Another learning was that the instance overview screen, as shown in Figure 6.2, can be a bit confusing. Users often did not realize that they are dealing with a list of instances. Furthermore, it was hard to find the icon to extend the instance so that they can configure an instance. There was also a problem with the placeholder text in the host input field. Because the placeholder was `http://localhost:33333` (the local instance, which is what you usually want to enter here), users thought the field was already filled in properly.


add new instance
please enter a host and a friendly name

name
e.g. my webserver

host
e.g. http://127.0.0.1:33333

Figure 6.3: Placeholder text when creating a new instance in the old Web UI (v1.0).

Most users just want to configure a single machine. Configuring multiple machines is a use case for more advanced users. As a result, single instance mode was introduced. Single instance mode skips the overview screen completely and immediately opens the interactive tree view.

Single instance mode can be used by passing the `INSTANCE` environment variable when starting the client:

```
INSTANCE="http://localhost:33333" kdb run-web
```

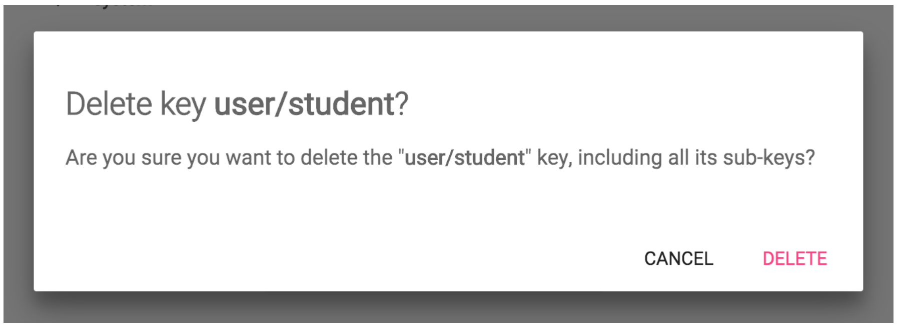
It is also possible to set the visibility by prefixing the host with `VISIBILITY@`. For example, to set the "advanced" visibility ("user" is default):

```
INSTANCE="advanced@http://localhost:33333" kdb run-web
```

### Learning 3: Confirm Before Delete

Some users accidentally pressed the delete buttons in the tree view, which, for keys without subtrees, resulted in immediate deletion of the keys. The idea here was making the deletion of multiple keys easier. However, if accidentally pressed, it results in a very bad user experience.

As a result, confirmations before deletion were introduced. If the user is trying to delete a key with a subtree, the confirmation will additionally warn them that they are going to delete the key *and* its whole subtree.


Delete key **user/student**?

Are you sure you want to delete the "**user/student**" key, including all its sub-keys?

Figure 6.4: Delete confirmation in the old Web UI (v1.0).

Later on, with the introduction of undo/redo functionality, this confirmation dialog was removed again. If the user makes a mistake now, they can simply undo their action(s).

#### Learning 4: Write Down Introduction

There were also learnings about doing usability tests, such as writing down the introduction and reading it word for word, so that every participant has exactly the same preconditions.

### 6.3.2 Results of Usability Tests

Unfortunately, the screen recording was not tested in the pilot test. The Web UI ended up being significantly slower while the screen recording software was running during the first slot (TP1). In all other slots (TP2-TP6), the screen recordings were taken in lower quality (less frames per second), which fixed the problem. First learning: Always ensure the pilot test resembles the real usability test as closely as possible!

#### Quantitative Results

Comparing the time taken to complete the tasks, the QT GUI and the Web UI had very similar results. On average, completing all tasks with the QT GUI took *11 minutes and 13 seconds*. Completing all tasks with the Web UI took, on average, only a couple seconds less: *11 minutes and 6 seconds*.

However, looking at the completion rate, it is very noticeable that the QT GUI required a lot more help from the test moderator. The completions are compared in Table 6.1.

Table 6.1: Summary of results

| | **Successful (Y)** | **With Help (H)** | **Unsuccessful (N)** |
|---|---|---|---|
| **QT GUI** | 23 | 6 | 7 |
| **Web UI** | 33 | 3 | 0 |

More complex tasks, like Scenario F, could not be completed successfully at all with the QT GUI: *0/1/5*. With the Web UI, the completion rate for Scenario F is much higher: *5/1/0*. Additionally, Scenario F took significantly less time to complete with the Web UI: *3 minutes and 7 seconds* on average. With the QT GUI, Scenario F took, on average, *4 minutes and 37 seconds* to complete. Scenario A could be solved significantly faster with the QT GUI (on average, *1 minute*) than with the Web UI (on average, *1 minute 28 seconds*). Scenario B was also, on average, solved *22 seconds* faster with the QT GUI.

Table 6.2: Results of Scenario A

| | **TP1** | **TP2** | **TP3** | **TP4** | **TP5** | **TP6** | **Total** | **Average** |
|---|---|---|---|---|---|---|---|---|
| **Completed? (QT)** | H | Y | Y | Y | Y | H | 4/2/0 | - |
| **Time (QT)** | 02:10 | 00:58 | 00:28 | 00:43 | 00:54 | 00:52 | 06:05 | 01:00 |
| **Completed? (Web)** | Y | H | Y | Y | Y | Y | 5/1/0 | - |
| **Time (Web)** | 01:43 | 02:20 | 00:37 | 00:40 | 01:07 | 02:25 | 08:52 | 01:28 |

Table 6.3: Results of Scenario B

| | **TP1** | **TP2** | **TP3** | **TP4** | **TP5** | **TP6** | **Total** | **Average** |
|---|---|---|---|---|---|---|---|---|
| **Completed? (QT)** | Y | Y | Y | Y | Y | Y | 6/0/0 | - |
| **Time (QT)** | 00:49 | 01:04 | 01:04 | 00:27 | 00:47 | 00:37 | 04:48 | 00:48 |
| **Completed? (Web)** | Y | Y | Y | Y | Y | Y | 6/0/0 | - |
| **Time (Web)** | 01:49 | 00:52 | 00:48 | 00:53 | 00:49 | 01:54 | 07:05 | 01:10 |

Table 6.4: Results of Scenario C

| | **TP1** | **TP2** | **TP3** | **TP4** | **TP5** | **TP6** | **Total** | **Average** |
|---|---|---|---|---|---|---|---|---|
| **Completed? (QT)** | Y | H | Y | N | Y | Y | 4/1/1 | - |
| **Time (QT)** | 02:19 | 01:28 | 02:06 | 01:25 | 02:07 | 03:00 | 12:25 | 02:04 |
| **Completed? (Web)** | Y | Y | Y | Y | Y | Y | 6/0/0 | - |
| **Time (Web)** | 01:58 | 02:26 | 01:16 | 02:26 | 01:22 | 02:53 | 12:21 | 02:03 |

Table 6.5: Results of Scenario D

| | **TP1** | **TP2** | **TP3** | **TP4** | **TP5** | **TP6** | **Total** | **Average** |
|---|---|---|---|---|---|---|---|---|
| **Completed? (QT)** | Y | H | Y | Y | Y | Y | 5/1/0 | - |
| **Time (QT)** | 01:11 | 01:19 | 01:26 | 00:49 | 02:00 | 01:24 | 08:09 | 01:21 |
| **Completed? (Web)** | Y | Y | Y | Y | Y | Y | 6/0/0 | - |
| **Time (Web)** | 01:11 | 01:37 | 01:05 | 01:41 | 01:14 | 01:42 | 08:30 | 01:25 |

Table 6.6: Results of Scenario E

| | TP1 | TP2 | TP3 | TP4 | TP5 | TP6 | Total | Average |
|---|---|---|---|---|---|---|---|---|
| **Completed? (QT)** | N | Y | Y | Y | H | Y | 4/1/1 | - |
| **Time (QT)** | 01:06 | 00:58 | 00:55 | 00:57 | 02:34 | 01:34 | 08:04 | 01:20 |
| **Completed? (Web)** | Y | Y | Y | H | Y | Y | 5/1/0 | - |
| **Time (Web)** | 01:45 | 01:47 | 01:00 | 02:18 | 02:17 | 01:54 | 11:01 | 01:50 |

Table 6.7: Results of Scenario F

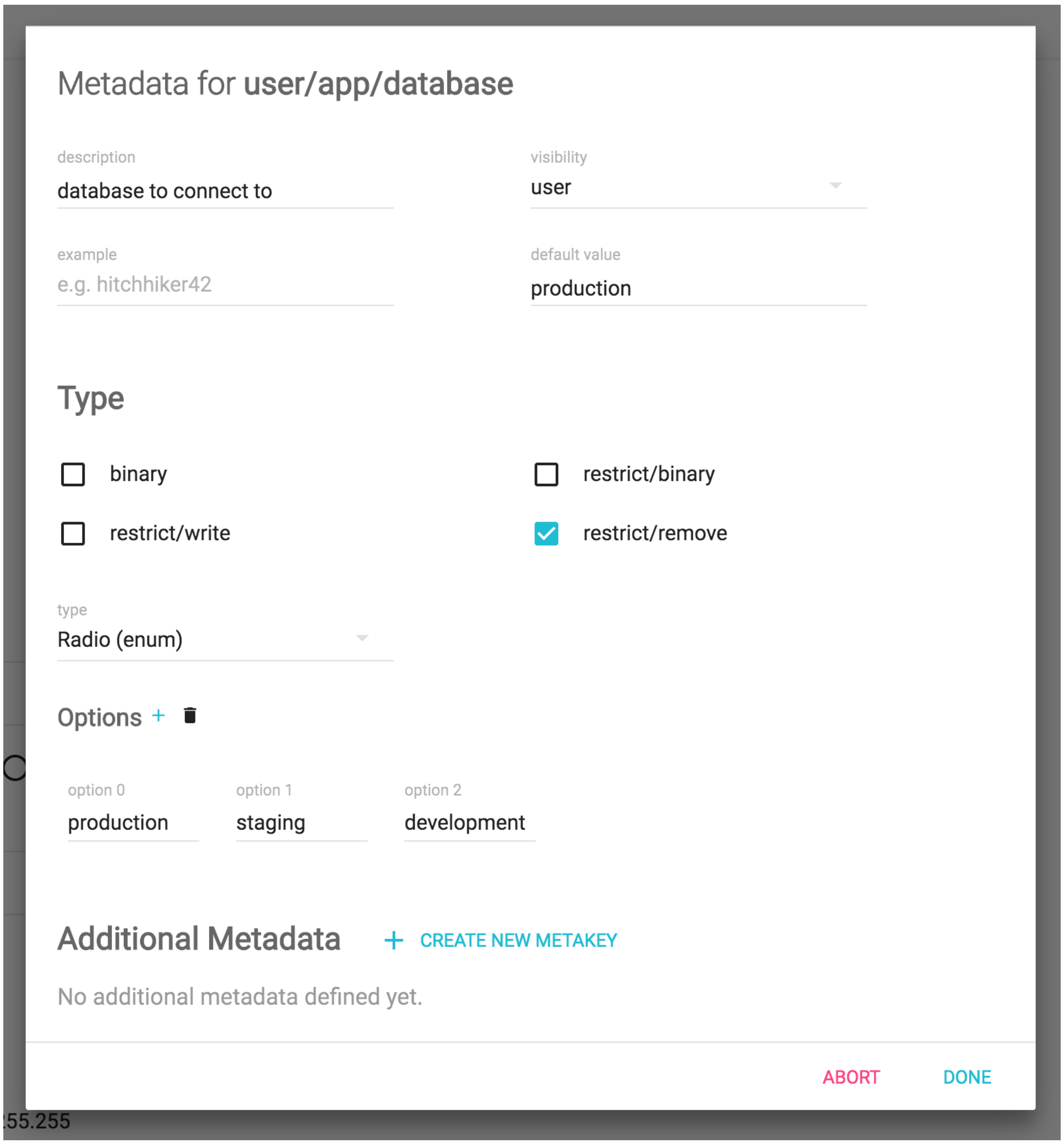
| | TP1 | TP2 | TP3 | TP4 | TP5 | TP6 | Total | Average |
|---|---|---|---|---|---|---|---|---|
| **Completed? (QT)** | N | N | N | H | N | N | 0/1/5 | - |
| **Time (QT)** | 04:03 | 02:39 | 04:05 | 03:50 | 06:01 | 07:06 | 27:44 | 04:37 |
| **Completed? (Web)** | Y | Y | H | Y | Y | Y | 5/1/0 | - |
| **Time (Web)** | 02:53 | 02:59 | 02:38 | 02:18 | 02:20 | 05:39 | 18:47 | 03:07 |

### Qualitative Results

In general, it was very noticeable that the participants were much more confident in their decisions when using the Web UI. With the QT GUI, participants often asked how values should be entered ("is a boolean represented as 'true' or '1'?", "did I enter this regex correctly?"). With the QT GUI, some participants were not sure if keys are subkeys or metakeys.

When using the QT GUI, participants often asked the test moderator if they completed tasks successfully. When using the Web UI, participants complimented that feedback in the UI is much clearer, and that features such as the duplicate option and automatic pre-filling of array key names are very useful. Automatic validation of all data that gets entered, especially metadata (Figure 6.5), was highly complimented across all participants. It was also complimented that metakeys are implemented as special input fields in the Web UI, as they often were not sure how to enter boolean values in the free text field of the QT GUI. Array keys were confusing to many participants in both the Web UI and the QT GUI.

Scenario F modeled a complex task, which could be done wrongly in many ways. In the QT GUI, some participants overlooked the "labs" section completely, others deleted labs instead of disabling them by changing the value from 1 to 0. Due to a bug in the QT GUI, participants ended up with a duplicated array key with the wrong name (#0 instead of #4). Almost none of those problems happened when using the Web UI: Only one participant asked if the labs should be deleted or just disabled.



Metadata for **user/app/database**

| | |
|---|---|
| description | database to connect to |
| visibility | user |
| example | e.g. hitchhiker42 |
| default value | production |

Type

- [ ] binary
- [ ] restrict/binary
- [ ] restrict/write
- [x] restrict/remove

| | |
|---|---|
| type | Radio (enum) |

Options

| option 0 | option 1 | option 2 |
|---|---|---|
| production | staging | development |

Additional Metadata

No additional metadata defined yet.

Figure 6.5: Editing metakeys in the Web UI, with special input fields and validation.

However, it was also noted that the Web UI is a little slow with large data sets and does not represent large amounts of data as nicely as the QT GUI, which displays data in a much more compact way. The search in the QT GUI was also complimented, because it works as a global search, not as a filter on already opened keys.

CHAPTER 7

# Conclusion

The goal of this thesis was to develop a Web UI with high usability to remotely configure machines with an interactive tree view. We used metadata to generate a user interface that is easy to use, with self-contained documentation.

To find out about the qualities of users who are interested in remote configuration via a Web UI, a survey was conducted. We can now answer the first research question:

**RQ1** Which user personas are interested in remote configuration via a Web UI?

It seems like configuration via a Web UI was especially important to the administrators and researchers. All participants rated (on average) having configuration files (the most popular option for configuration) as "important" (mean = 4.10, median = 4). Configuration via REST API was considered as only "slightly important" (mean = 2.29, median = 2). Administrators and researchers rated (on average) having a Web UI as "moderately important" (mean administrators = 2.93, mean researchers = 2.75).

At an earlier point, cluster configuration was the focus of this thesis. However, due to the results of the survey and the pilot test, it seems to do nothing but confuse users. Furthermore, there are not many real world use cases that could not be done in a more efficient way through other means.

From the results of the survey, we created two user personas: Andy the Administrator and Marie the Researcher. We then focussed on the researcher persona to create scenarios that describe real world tasks relevant to the persona. Finally, we conducted a usability test with participants that fit the user persona, testing the scenarios with both the QT GUI and the Web UI.

There was only a small difference in time spent to complete tasks. However, participants asked for help a lot more frequently and made more errors when using the QT GUI. We can thus answer the other research questions:

**RQ2** How does the Web UI compare to the QT GUI in amount of errors made, help required and time needed to complete tasks?

The Web UI has a similar efficiency to the QT GUI. However, the Web UI seems to be slightly more effective (less help needed, less errors made) when it comes to more complex scenarios, such as Scenario F.

**RQ3** How much more or less satisfied are users with the Web UI in comparison to the QT GUI, measured by qualitative user questioning?

Participants were much more confident in their decisions when using the Web UI. All participants stated they were more satisfied using the Web UI in comparison to the QT GUI. However, it was also stated that the global search of the QT GUI is more useful and that they liked how fast the QT GUI is. Many features of the Web UI, including special input fields and validation of all (meta)data, were highly complimented across all participants. Especially more complex scenarios were causing troubles with the QT GUI, as there were more ways to make mistakes. Additionally, a bug in the QT GUI slowed down the process of solving Scenario F.

The search functionality of the QT GUI significantly reduced time to solve certain tasks, like Scenario A and Scenario B. As a result, the filter search of the Web UI was turned into a more useful global search, which should increase efficiency for these scenarios.

Array keys were confusing in both the Web UI and the QT GUI. Instead of creating a normal key, then adding a #0 subkey to it, there is now a button in the "create key" dialog to create an array key, which automatically creates a key with a #0 subkey, as shown in Figure 7.1.

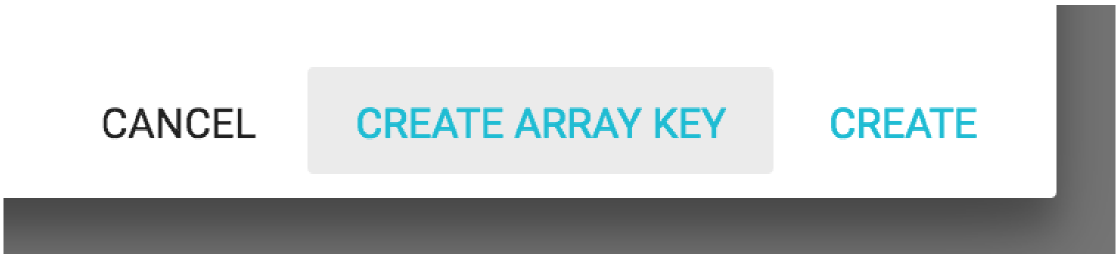


Figure 7.1: A later version of the Elektra Web UI (v1.5), with the "create array key" button.

We can now answer the main research question of this thesis:

**RQ0** How does the interactive tree view of the Web UI compare to the QT GUI in terms of usability?

The Web UI certainly has a lot of usability improvements compared to the QT GUI. Especially the amount of feedback and validation that the Web UI provides were highly complimented and made participants much more confident in their decisions. However, there were also tasks that could be completed more easily with the QT GUI, mostly as a result of the confusing filter search in the Web UI, which was turned into a global search in a later release.

All in all, the Web UI should serve as a great demonstration that it is useful to limit input fields. Furthermore, the Web UI offers a usable web user interface that allows exploration without knowing much about Elektra's internals.

## 7.1 Further Work

However, there is also a lot of room for improvement with the Web UI.

### 7.1.1 Performance

Currently, elektrad spawns *kdb*[1] processes, as there are no Node.js bindings for Elektra at the time of writing. Replacing elektrad with a daemon using the Elektra C/C++ API would greatly improve the performance of Elektra Web.

### 7.1.2 Securing Access

Currently, the only way to protect access to the Web UI is via a web server and HTTP authentication [FHBH+99]. This can be done by, for example, setting up an nginx reverse proxy[2] and protecting it via username/password authentication. It would be a better solution to implement user accounts in the relay daemon (webd).

### 7.1.3 Import/Export

Currently, data must be imported and exported via the CLI or the QT GUI. Implementing an import/export feature into elektrad and the Web UI could be very useful.

### 7.1.4 Mounting & Plugins

Elektra supports mounting existing configuration files. While it is already possible to access mounted configuration files via the Web UI, it is not possible to mount new configuration files into the KDB. Furthermore, it is not possible to interface with plugins, such as the crypto plugin.

### 7.1.5 Live Updates

There is currently some work being done on implementing notifications about changes to the KDB. If live updates are implemented in Elektra Web, it would even be possible to collaborate with multiple people by using the Web UI at the same time.

[1] `http://master.libelektra.org/doc/help/kdb.md`
[2] `https://www.nginx.com/resources/admin-guide/reverse-proxy/`

# List of Figures

# List of Tables

# Acronyms

**API** Application Programming Interface. 20–22

**CLI** Command-Line Interface. 2, 3, 23, 32, 33, 43

**CORS** Cross-Origin Resource Sharing. 21

**DOM** Document Object Model. 22

**elektrad** Elektra daemon. 19–21, 43

**GUI** Graphical User Interface. 2, 8, 18, 23

**HTTP** Hypertext Transfer Protocol. 20, 21

**KDB** Elektra Key Database. 1, 2, 20, 43

**QT GUI** Elektra QT GUI. ix, 2, 3, 5, 8, 25–28, 37–43, 45

**regex** regular expressions. 14

**SSH** Secure Shell. 2, 32

**UI** user interface. 22, 39

**Web UI** Web User Interface. ix, 3–5, 9, 13–19, 22–28, 31–33, 35–43, 45, 47

**webd** relay daemon. 19, 21, 22, 43

# Bibliography

[Abr18] Dan Abramov. Redux documentation. `https://redux.js.org/`, accessed June 16, 2018.

[Bra58] James V Bradley. Complete counterbalancing of immediate sequential effects in a latin square design. *Journal of the American Statistical Association*, 53(282):525–528, 1958.

[BW88] RS Bird and PL Wadler. *Functional Programming.* Prentice Hall, 1988.

[CC13] Evan Czaplicki and Stephen Chong. Asynchronous functional reactive programming for guis. In *ACM SIGPLAN Notices*, volume 48, pages 411–422. ACM, 2013.

[CGK12] Gary Charness, Uri Gneezy, and Michael A Kuhn. Experimental methods: Between-subject and within-subject design. *Journal of Economic Behavior & Organization*, 81(1):1–8, 2012.

[CS00] Mary B Coney and Michael Steehouder. Role playing on the web: Guidelines for designing and evaluating personas online. *Technical communication*, 47(3):327–340, 2000.

[DR99] Joseph S. Dumas and Janice C. Redish. *A Practical Guide to Usability Testing.* Intellect Books, Exeter, UK, UK, 1st edition, 1999.

[EH97] Conal Elliott and Paul Hudak. Functional reactive animation. In *ACM SIGPLAN Notices*, volume 32, pages 263–273. ACM, 1997.

[FHBH$^+$99] John Franks, Phillip Hallam-Baker, Jeffrey Hostetler, Scott Lawrence, Paul Leach, Ari Luotonen, and Lawrence Stewart. Http authentication: Basic and digest access authentication. Technical report, 1999.

[Fri02] Jeffrey EF Friedl. *Mastering regular expressions.* " O'Reilly Media, Inc.", 2002.

[gno17] Gnome human interface guidelines. `https://developer.gnome.org/hig/stable/`, accessed February 22, 2017.

[Hug89] John Hughes. Why functional programming matters. *The computer journal*, 32(2):98–107, 1989.

[Inc18] Facebook Inc. React documentation. `https://reactjs.org/`, accessed June 16, 2018.

[Kap00] Gerti Kappel. Modeling customizable web applications - a requirement's perspective. In *Kyoto International Conference on Digital Libraries: Research and Practice (ICDL)*, 2000.

[KL14] Gideon Keren and Charles Lewis. *A Handbook for Data Analysis in the Behaviorial Sciences: Volume 1: Methodological Issues Volume 2: Statistical Issues*. Psychology Press, 2014.

[LPJ94] John Launchbury and Simon L Peyton Jones. Lazy functional state threads. In *ACM SIGPLAN Notices*, volume 29, pages 24–35. ACM, 1994.

[mac17] macos human interface guidelines. `https://developer.apple.com/library/content/documentation/UserExperience/Conceptual/OSXHIGuidelines/`, accessed February 22, 2017.

[May99] Deborah J. Mayhew. The usability engineering lifecycle. In *CHI '99 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '99, pages 147–148, New York, NY, USA, 1999. ACM.

[Nie93] Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann, 1993.

[Nie94] Jakob Nielsen. Usability inspection methods. In *Conference companion on Human factors in computing systems*, pages 413–414. ACM, 1994.

[Nor13] Donald A. Norman. *The design of everyday things: Revised and expanded edition*. Basic Books, 2013.

[Psa08] Giuseppe Psaila. Virtual dom: An efficient virtual memory representation for large xml documents. In *Database and Expert Systems Application, 2008. DEXA'08. 19th International Workshop on*, pages 233–237. IEEE, 2008.

[qt517] Qt best practice guides. `https://doc.qt.io/qt-5/best-practices.html`, accessed February 22, 2017.

[Raa10] Markus Raab. A modular approach to configuration storage. Master's thesis, Vienna University of Technology, 2010.

[RC08] Jeffrey Rubin and Dana Chisnell. *Handbook of usability testing: how to plan, design and conduct effective tests*. John Wiley & Sons, 2008.

[Win62] Benjamin J Winer. Latin squares and related designs. 1962.

[win17] Windows user interface principles. `https://msdn.microsoft.com/en-us/library/windows/desktop/ff728831(v=vs.85).aspx`, accessed February 22, 2017.

[WTH01] Zhanyong Wan, Walid Taha, and Paul Hudak. Real-time frp. *ACM SIGPLAN Notices*, 36(10):146–156, 2001.