Configuration Management

Markus Raab

Institute of Information Systems Engineering, TU Wien

10.04.2019

Lecture is every week Wednesday 09:00 - 11:00.

06.03.2019:	topic, teams
13.03.2019:	TISS registration, initial PR
20.03.2019:	other registrations, guest lecture
27.03.2019:	PR for first issue done, second started
03.04.2019:	first issue done, PR for second
10.04.2019:	mid-term submission of exercises
08.05.2019:	Different Location: Complang Libary
15.05.2019:	
22.05.2019:	
29.05.2019:	
05.06.2019:	final submission of exercises
12.06.2019:	
19.06.2019:	last corrections of exercises
26.06.2019:	exam

Tasks for today

(until 10.04.2019 23:59)
mid-term submission of exercises

Task

Submit a first version of both teamwork and homework.

Does not need to be complete, important is that you get started.

Task

Second PR done, PR for third issue created and write some text in at least one other issue (if 5 issues are not yet assigned to you).

Task

Write one architectural decision for your teamwork or Elektra.

Tasks for after eastern

(until 08.05.2019 23:59)

Task

Incorporate feedback for teamwork and homework.

Task

Third PR done, PR for fourth issue created and write some text in at least one other issue (if 5 issues are not yet assigned to you).

Popular Topics

14	tools	4	design
9	testability	4	cascading
9	code-generation	4	architecture of access
7	context-awareness	3	configuration sources
6	specification	3	config-less systems
6	misconfiguration	2	secure conf
6	complexity reduction	2	architectural decisions
5	validation	1	push vs. pull
5	points in time	1	infrastructure as code
5	error messages	1	full vs. partial
5	auto-detection	1	convention over conf
4	user interface	1	CI/CD
4	introspection	0	documentation

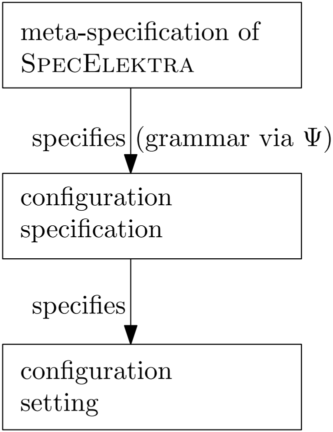
Recapitulation

- alarming trend in number and complexity of configuration
- sharing, visibility and default value calculation may help
- but also more courageous decisions and periodical reevaluation
- both need abstraction: configuration specification

Metalevels (Recapitulation)

Question

Describe the three Metalevels in Elektra.



meta-specification of SpecElektra

specifies (grammar via Ψ)

configuration specification

specifies

configuration setting

SpecElektra

SpecElektra is a modular ***configuration specification language*** for configuration settings. In SpecElektra we use properties to specify configuration settings and configuration access. SpecElektra enables us to specify different parts of Elektra.

Recapitulation (Requirements of SpecElektra)

- formal and informal
- should strive for completeness
- should be extensible
- should be external to application
- open for introspection (for tooling)
- should talk to users
- should allow generation of artefacts

Goals for today

- modularity on system level
  - horizontal
  - vertical
- system-wide introspection
- avoiding dependences
- auto-detection

Modularity

1. Modularity
   - Vertical
   - Horizontal

2. Plugins
   - Why?
   - How?

Status Quo in Free Systems

- nearly all applications use their own configuration system
- immense differences in configuration file formats and configuration access
- very high modularity

Status Quo in Frameworks and Proprietary Systems

- obvious ways how to deal with configuration
- no differences in configuration access
- very low modularity

Types of Modularity

Vertical modularity describes how strongly separated the configuration accesses of different applications is. *Horizontal modularity* describes how strongly separated modules implementing configuration access for a single application is.

Vertical Modularity [1]

Vertical modularity is the degree of separation between different applications. If all applications use the same key database with a single backend or a single configuration file, applications would be coupled tightly. [...]
If coupling between applications is low, for example every application uses a different configuration library or a different backend, we have a high degree of vertical modularity.

Retain Vertical Modularity [1]

Elektra provides two mechanisms to retain vertical modularity:

- **Mounting** configuration files facilitates different applications to use their own backend and their own configuration file. Furthermore, mounting enables integrating existing configuration files into the key database. Configuration specifications written in SpecElektra allow different applications to share their configuration files with each other in a controlled way.
- Having frontends that implement existing **APIs** decouple applications from each other. These applications continue to use their specific configuration accesses, but Elektra redirects their configuration accesses to the shared key database.

Vertical Modularity [1]

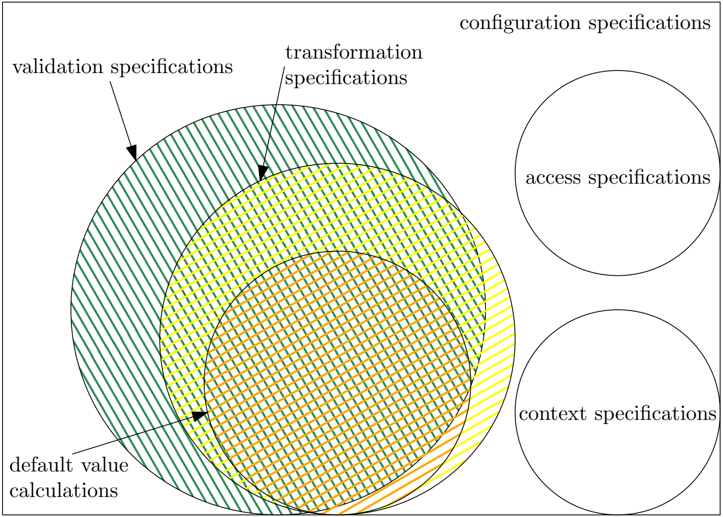
Mountpoints can also be a part of the specification:

```
[ntp]
  mountpoint:=ntp.conf
[sw/libreoffice]
  mountpoint:=libreoffice.conf
```

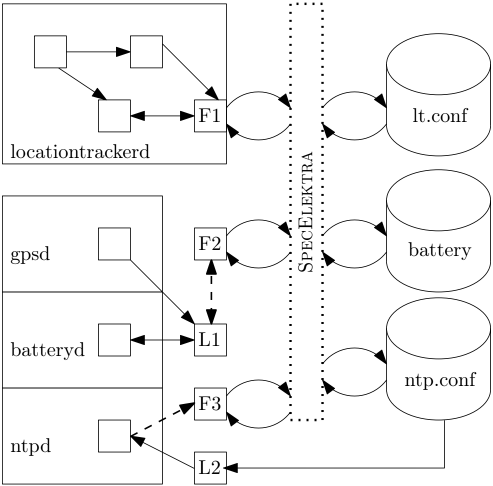
Task

Which type of specification is this?

Types of Specifications


configuration specifications
validation specifications
transformation specifications
access specifications
context specifications
default value calculations

Vertical Modularity



Needed to keep applications independently.
Boxes are applications, cylinders are configuration files, F? are frontends or frontend adapters, L? are configuration libraries [1].

Task

Break.

Horizontal Modularity [1]

Horizontal modularity is “the degree of separation in configuration access code” [1]. A higher degree of horizontal modularity allows us to better separate configuration access code and plug the code together as needed.

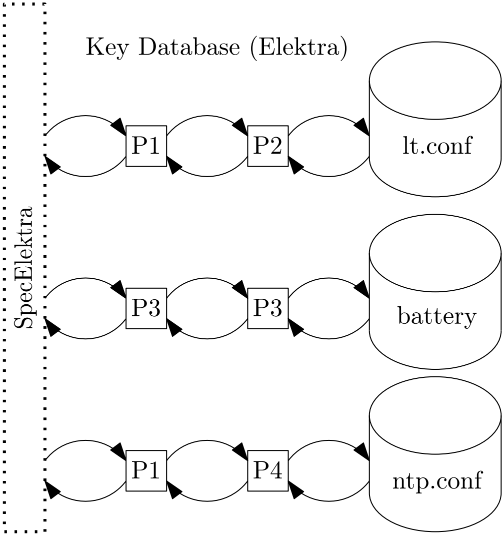
Three factors of SpecElektra improve horizontal modularity:

1. Using SpecElektra, applications are completely decoupled from configuration specifications.
2. Specifications and their implementation are decoupled.
3. Abstract dependences within the implementation of specifications.

Task

This is very vague.
Can you describe a system that would (not) fulfil this?

Horizontal Modularity



Needed for validation, auto-detection, . . .

Cylinders are configuration files, P? are plugins [1]

Plugins

1. Modularity
   - Vertical
   - Horizontal
2. Plugins
   - Why?
   - How?

Acceptable Effort

Q: “Which effort do you think is worthwhile for providing better configuration experience?”

- 44 % would use other configuration access APIs next to `getenv`.
- 30 % would use OS-specific sources.
- 21 % would use dedicated libraries.
- 19 % would read other application’s configuration settings,
- 16 % would use external configuration access APIs that add new dependences.

Why?

Finding

Q: Most developers have concerns adding dependences for more validation (84 %) but consider good defaults important (80 %).

Requirement

Dependences exclusively needed to validate configuration settings must be avoided.

Rationale

Why is it difficult to have good defaults?

- **Modularity:** diverse and conflicting requirements between applications. Especially in validation, for example, constraint solvers vs. type systems vs. model checkers.
- **System-level:** specification must always be enforced. Examples:
  - which desktop is the application started in?
  - how many CPUs does the system have?
  - get the correct proxy of the system.
  - get available network bandwidth.
  - is the filesystem local?

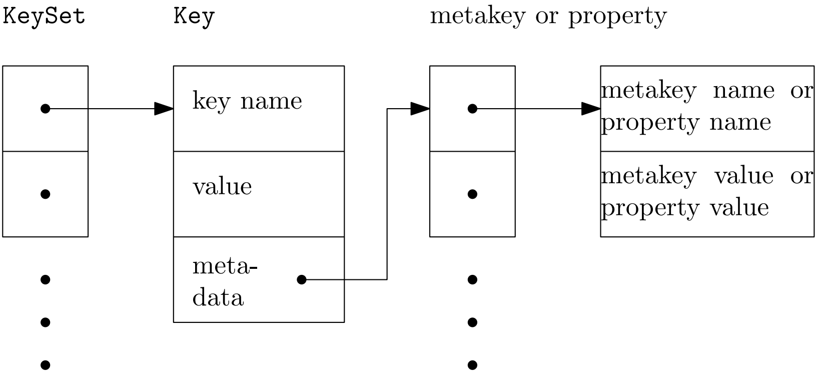
How?

Plugins are filters, sinks, and sources processing a key set. We aim at SpecElektra to be as modular as possible and make extensive use of plugins:

1. SpecElektra does not have any built-in feature, all features are (or can be) implemented as plugins.
2. Elektra works completely without SpecElektra's specifications.
3. Configuration specifications are present within the execution environment. Thus any tool and plugin can introspect and use the specifications.

KeySet

The common data structure between plugins:


KeySet
Key
metakey or property
key name
value
meta-data
metakey name or property name
metakey value or property value

Plugin Assembly

automatic assembling of plugins:

- iterate over the specification and collect all key words
- iterate over all plugins and check if they offer key words
- check contract between plugins and specification
- of the remaining plugins: use best suited or rated

(implemented in `kdb mount` / `kdb spec-mount` in Elektra)

SpecElektra is a dependency injection mechanism:

- By extending the specification, new plugins are being injected into the system.
- The *provider* abstractions in the dependences between the plugins abstract over concrete implementations of configuration access code.
- We have a modular implementation of SpecElektra.

Task

Which kind of modularity does *provider* improve?

Answer

3rd point of horizontal modularity on Slide 22

Examples

calculation with context:

```
[gps/status]
assign:=(battery > 'low') ? ('on') : ('off')
[battery]
plugins:=battery
```

Examples

resolve names of configuration files

```
[example]
  mountpoint:=/example.ini
```

depending on operating system, e.g. UNIX:

namespace	resolved path
spec	/example.ini
dir	${PWD}/example.ini
user	${HOME}/example.ini
system	/example.ini

Preview

next lecture after eastern:
code generation vs. introspection

[1] Markus Raab. Improving system integration using a modular configuration specification language. In *Companion Proceedings of the 15th International Conference on Modularity*, MODULARITY Companion 2016, pages 152–157, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4033-5. doi: 10.1145/2892664.2892691. URL `http://dx.doi.org/10.1145/2892664.2892691`.