L02 Configuration Specification Languages

Markus Raab

Institute of Information Systems Engineering, TU Wien

Theory

Rationale

- without specification you and others do not even know which settings are available
- needed for any further techniques we will discuss
- essential for ***no-futz computing*** Holland et al. [1]
- the foundation for any advanced tooling like configuration management tools
- needed as communication of producers and consumers of configuration

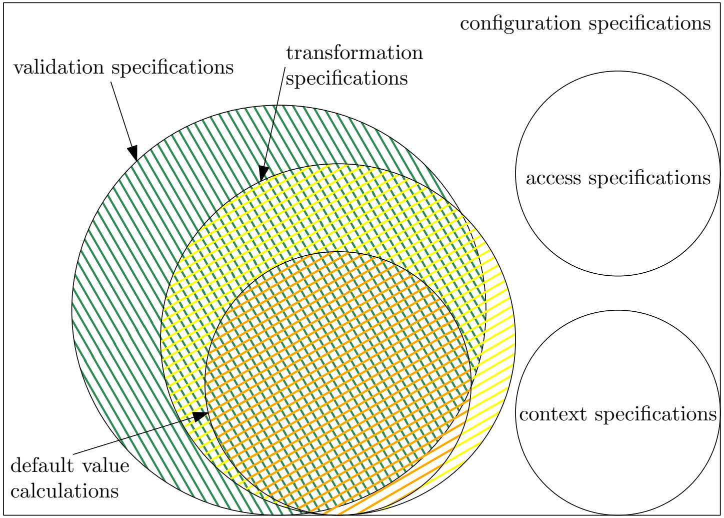
Q: "Configuration specification (e.g. XSD/JSON schemas) allows you to describe possible values and their meaning. Why do/would you specify configuration?"

58 %	for *"looking up what the value does"*,
51 %	it helps users to avoid common errors (*"so that users avoid common errors"*),
46 %	to simplify maintenance,
40 %	for rigorous validation,
39 %	for documentation generation (for example, man pages, user guide),
30 %	for external tools accessing configuration,
28 %	for generating user interfaces,
25 %	for code generation, and
24 %	for specification of links between configuration settings.

Limitations of Schemata designed for Data

- e.g. XSD/JSON schemas
- they are already very helpful but:
  - not key-value based
  - not easy to introspect
  - designed to validate data without semantics: file path vs. presence of file
  - not always possible to extend with plugins
  - tied to specific formats (e.g. XML/JSON)

Types of Specifications


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

Requirements

- formal/informal?
- complete?
- should be extensible
- should be external to application
- open for introspection
- should talk to users
- should allow generation of artefacts

Grammar

$\langle$*configuration specifications*$\rangle$ ::= { $\langle$*configuration specification*$\rangle$ }

$\langle$*configuration specification*$\rangle$ ::= '[' $\langle$*key*$\rangle$ ']' $\langle$*properties*$\rangle$

$\langle$*properties*$\rangle$ ::= { $\langle$*property*$\rangle$ }

$\langle$*property*$\rangle$ ::= $\langle$*property name*$\rangle$ ':=' [$\langle$*property value*$\rangle$]

Example:

```
[slapd/threads/listener]
default:=1
type:=long
```

L02 Configuration Specification Languages

Markus Raab

Institute of Information Systems Engineering, TU Wien

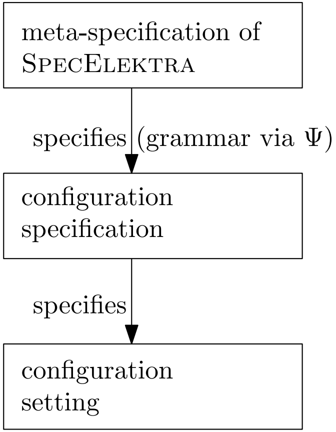
Practice

Learning Outcomes

Students will be able to

- use configuration specification languages.

Metalevels (Recapitulation)


meta-specification of SpecElektra
specifies (grammar via Ψ)
configuration specification
specifies
configuration setting

We will now walk through metalevels bottom-up.

Configuration Settings (Recapitulation)

A configuration file may look like:

```
a=5
b=10
c=15
```

We apply these configuration settings imperatively using:

```
kdb set /a 5
kdb set /b 10
kdb set /c 15
```

And we list them with `kdb ls /`.

Specifications (Recapitulation)

For specifications such as:

```
    [slapd/threads/listener]
      type:=short
      default:=1
```

We apply the specifications imperatively using:

```
    kdb meta-set /slapd/threads/listener\
        type short
    kdb meta-set /slapd/threads/listener\
        default 1
```

(automatically uses `spec:` namespace)

Meta-Specifications (Recapitulation)

For meta-specifications such as:

```
[type]
type:=enum short unsigned_short long \
    float double char boolean any string ...
description:=Defines the type of the value, \
     as specified in CORBA
```

We apply the meta-specifications imperatively using:

```
kdb meta-set system:/info/elektra/metadata/type/#0 \
    type "enum short ..."
kdb meta-set system:/info/elektra/metadata/type/#0 \
    description "Defines ..."
```

see `doc/METADATA.ini`

SpecElektra

- we use it to demonstrate configuration specification languages
- a modular ***specification language*** for configuration settings
- we use properties to specify configuration settings and configuration access
- SpecElektra specifies the behavior of Elektra

Mountpoint

The root of each configuration specification, e.g. in ni syntax:

```
[]
mountpoint = vlc.ini
infos/plugins = ni
```

Hierarchy

Always prefer hierarchy separator (/) as only separator:

```
[server/ip]
```

Avoid other separators:

```
[server_ip]
[server-ip]
[server.ip]
```

Because they limit extensibility as they do not create sections in configuration files.

Types

Presence alone indicates availability of a configuration setting:

```
[server/port]
```

Equivalent to `type:=any`.

Properties give restrictions:

```
[server/port]
type:=short
```

Require vs. Default

Prefer default values:

```
[server/ip]
default:=127.0.0.1
```

Note that defaults must be sane and secure.

Avoid require:

```
[server/ip]
require:=
```

Because this forces the user to take action.

Note

`require` and `default` do not make sense together.

IP Addresses

```
[server/ip]
check/ipaddr:=ipv4
example:=0.0.0.0
default:=127.0.0.1
```

Two plugins provide `check/ipaddr`: ipaddr and network
Will be automatically selected.

Arrays

```
[servers]
array:=

[servers/#/ip]
check/ipaddr:=ipv4

[servers/#/port]
type:=short
```

Command-line Options

Environment and command-line options can be considered with:

```
[recursive]
  type:=boolean
  opt:=r
  opt/long:=recursive
  env:=RECURSIVE
  default:=0
```

Dates

```
[mydate]
example:=2021-03-01
type:=string
check/date:=ISO8601
check/date/format:=calendardate complete extended
```

Design Considerations

Percentages
(e.g., configured image should be additionally cropped):

```
[image/width]
type:=long

[crop/width]
type:=long
check/range:=0-100
```

Artifacts

- plugins in configuration framework (e.g. validate settings)
- tooling (GUI, Web UI)
- generate examples/documentation
- auto-completion/syntax highlighting/IDE support

L02 Configuration Specification Languages

Markus Raab

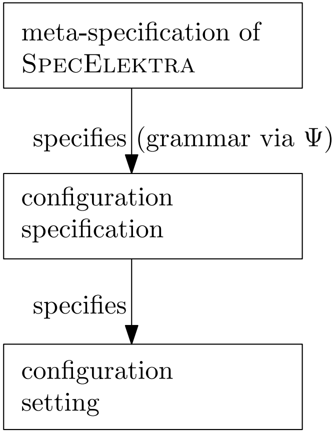
Institute of Information Systems Engineering, TU Wien

Meeting

Metalevels (Recapitulation)

Question

Draw and describe the three Metalevels in Elektra.


meta-specification of SpecElektra
specifies (grammar via Ψ)
configuration specification
specifies
configuration setting

Task

Why do we need a data structure?
(E.g. why not directly manipulating key/values?)

Task

What do we mean with a configuration specification?
Which requirements do we have for a configuration specification?

- should be extensible
- should be external to application
- open for introspection (for tooling)
- should talk to users
- should allow generation of artefacts

Task

Break.

Task

What can be part of a configuration specification? What can they be used for?

(METADATA.ini in shared notes)

Task

Break.

Task

Now, how do we implement such a specification? Which artifacts can we generate?

Artifacts

- plugins in configuration framework (e.g. validate settings)
- tooling (GUI, Web UI)
- generate examples/documentation
- auto-completion/syntax highlighting/IDE support

Teams

Task

All Teams formed?

Possible Tasks for P*

- Elektra support for some configuration management tool
- Integrate kdb tool in Latex or similar environments
- Improve error messages in validations plugin (also T2)
- Rewrite plugins to new comments style (also T2)

Working on H* together

- working on the same application allowed!
- simply use different configuration options

Task

Any applications already found?

Examples: gtimelog (Python), LanguageTool (Java), ...

Develop with Elektra

Task

Can you already compile software using Elektra?

Task

Can you reformat the code?

Task

Can you run all the tests?

Preview Next Week

- Configuration Libraries
- Lightweight to Strong Integration
- Sharing Configuration

[1] David A. Holland, William Josephson, Kostas Magoutis, Margo I. Seltzer, Christopher A. Stein, and Ada Lim. Research issues in no-futz computing. In *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*, pages 106–110. IEEE, May 2001. doi: 10.1109/HOTOS.2001.990069.

[2] Markus Raab and Gergö Barany. Introducing context awareness in unmodified, context-unaware software. In *Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering - Volume 1: ENASE,*, pages 218–225. INSTICC, ScitePress, 2017. ISBN 978-989-758-250-9. doi: 10.5220/0006326602180225.