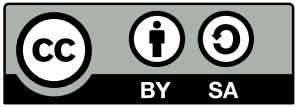
L02 Configuration Specification Languages

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

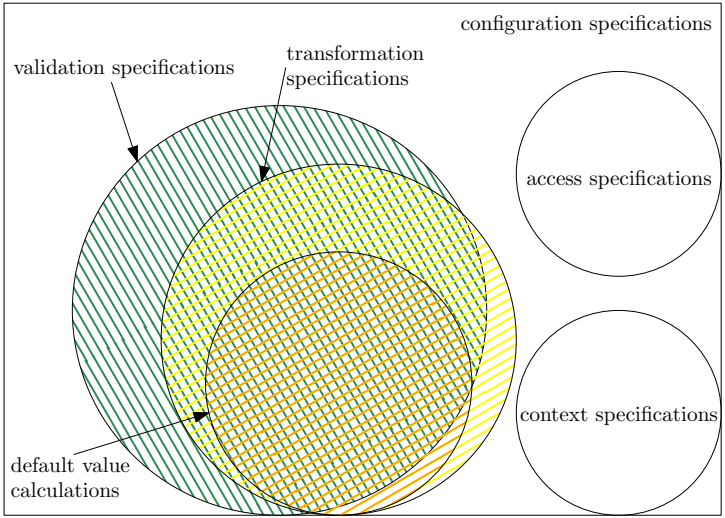
Q: "Configuration specification (e.g. XSD/JSON schemas) allows you to describe possible values and their meaning. Why do/would you specify configuration?"

- 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



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

⟨*configuration specifications*⟩ ::= { ⟨*configuration specification*⟩ }

⟨*configuration specification*⟩ ::= '[' ⟨*key*⟩ ']' ⟨*properties*⟩

⟨*properties*⟩ ::= { ⟨*property*⟩ }

⟨*property*⟩ ::= ⟨*property name*⟩ ':=' [⟨*property value*⟩]

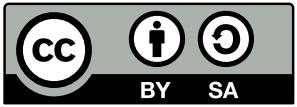
Example:

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

L02 Configuration Specification Languages

Markus Raab

Institute of Information Systems Engineering, TU Wien



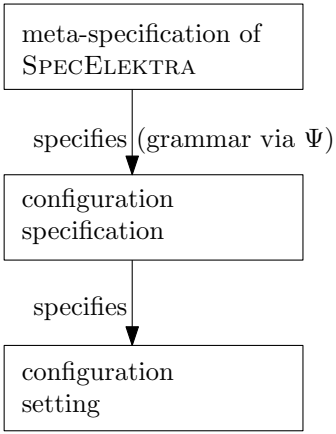
Practice

Learning Outcomes

Students will be able to

- use configuration specification languages.

Metalevels (Recapitulation)



We will now walk through metalevels bottom-up.

Configuration Settings (Recapitulation)

A configuration file may look like:

```
1  a=5
2  b=10
3  c=15
```

We apply these configuration settings imperatively using:

```
1  kdb set /a 5
2  kdb set /b 10
3  kdb set /c 15
```

And we list them with `kdb ls /`.

Specifications (Recapitulation)

For specifications such as:

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

We apply the specifications imperatively using:

```
1     kdb meta-set /slapd/threads/listener\
2         type short
3     kdb meta-set /slapd/threads/listener\
4         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:

```
1 []
2 mountpoint = vlc.ini
3 infos/plugins = ni
```

Hierarchy

Always prefer hierarchy separator (/) as only separator:

```
1 [server/ip]
```

Avoid other separators:

```
1 [server_ip]
2 [server-ip]
3 [server.ip]
```

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

Types

Presence alone indicates availability of a configuration setting:

```
1 [server/port]
```

Equivalent to `type:=any`.

Properties give restrictions:

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

Require vs. Default

Prefer default values:

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

Note that defaults must be sane and secure.

Avoid require:

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

Because this forces the user to take action.

Note

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

IP Addresses

```
1 [server/ip]
2 check/ipaddr:=ipv4
3 example:=0.0.0.0
4 default:=127.0.0.1
```

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

Arrays

```
1 [servers]
2 array:=
3
4 [servers/#/ip]
5 check/ipaddr:=ipv4
6
7 [servers/#/port]
8 type:=short
```

Command-line Options

Environment and command-line options can be considered with:

```
1 [recursive]
2   type:=boolean
3   opt:=r
4   opt/long:=recursive
5   env:=RECURSIVE
6   default:=0
```

Dates

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

Design Considerations

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

```
1 [image/width]
2 type:=long
3
4 [crop/width]
5 type:=long
6 check/range:=0-100
```

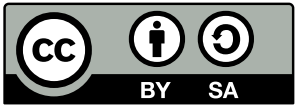
Artefacts

- 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

[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.