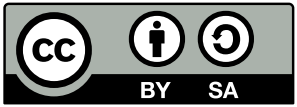
L01 Configuration Settings

Markus Raab

Institute of Information Systems Engineering, TU Wien



Elektra

Elektra [21]

- ELEKTRA is a framework implementing a modular ***configuration specification language*** for configuration settings
- ***configuration specification languages*** mitigate misconfigurations
- ELEKTRA enables ***no-futz computing*** [11], i.e., error-prone *"tinkering or fiddling experimentally" "should be allowed, but should never be required"*

Elektra as Virtual Filesystem

- configuration files are seen like “block devices”
- are mounted with respective filesystem drivers into the filesystem
- many tools and APIs evolved to work with files
- Idea of Elektra: establish a similar ecosystem for configuration

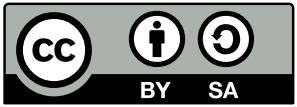
Why is Elektra not a Filesystem then?

- API semantics: key/value get/set
- namespaces: based on established semantics
- many features essential for misconfiguration hardening:
  - validation
  - visibility
  - defaults
  - ... (extensible specification)

L01 Configuration Settings

Markus Raab

Institute of Information Systems Engineering, TU Wien



Definitions

Learning Outcomes

Students will be able to

- remember definitions of configuration settings.

Basic Definitions

The ***execution environment*** is information outside the boundaries of each currently running process [6].
Controlling the execution environment is essential for configuration management [5, 12], testing [27, 31], and security [9, 14, 19, 25].

Configuration Setting

Definition

A ***configuration setting***, or ***setting*** in short, fulfills these properties:

1. It is provided by the execution environment.
2. It is *consumed* by an application.
3. It consists of a key, a configuration value, and potentially *metadata*. The ***configuration value***, or ***value*** in short, influences the application's behavior.
4. It can be *produced* by the maintainer, user, or system administrator of the software.

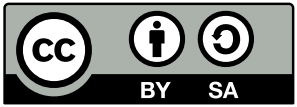
Synonyms for Configuration Settings

User preferences [13] and ***customization*** [1] stress that users make the change although that might not always be the case. ***Variability points*** [10, 16, 17, 28–30] aim at describing the capability of software to adapt its behavior. ***Derivation decision*** [7, 8] puts the decisions to make and not the result in focus. ***Configuration parameter*** [2, 34] is easily confused with other kinds of parameters. ***Configuration item*** [3] or ***configuration option*** [24, 35, 36] are sometimes not applicable, for example, “proxy option”, or “language item”. ***Configuration data*** [12] is often used in the context of programmable gate arrays and has a different meaning in that domain.

L01 Configuration Settings

Markus Raab

Institute of Information Systems Engineering, TU Wien



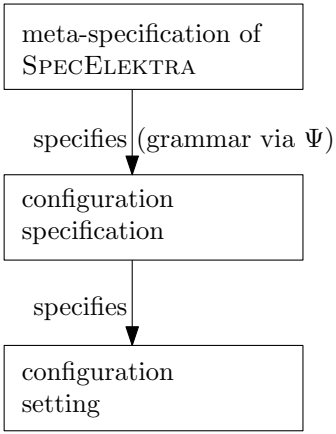
Metalevels

Learning Outcomes

Students will be able to

- describe three metalevels of configuration

Metalevels



We will now walk through metalevels bottom-up.

Configuration Settings

A configuration file may look like (properties format):

```
1    slapd/threads/listener=4
```

We apply these configuration settings imperatively using:

```
1    kdb set /slapd/threads/listener 4
```

Specifications

For specifications such as:

```
1     [slapd/threads/listener]
2       check/range:=1,2,4,8,16
3       default:=1
4       visibility:=advanced
5       description:=One thread is adequate\
6                for up to 16 CPU cores.
```

We apply the specifications imperatively using:

```
1     kdb meta-set /slapd/threads/listener\
2         check/range 1,2,4,8,16
3     kdb meta-set /slapd/threads/listener\
4         default 1
```

Meta-Specifications

For meta-specifications such as:

```
1    [visibility]
2    type:=enum critical important user\
3          advanced developer debug disabled
4    description:=Who should see this\
5         configuration setting?
```

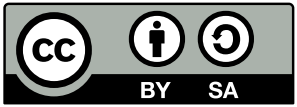
We apply the meta-specifications imperatively using:

```
1    kdb meta-set /elektra/meta/\
2        visibility type enum ...
3    kdb meta-set /elektra/meta/\
4        visibility description "Who ...
```

L01 Configuration Settings

Markus Raab

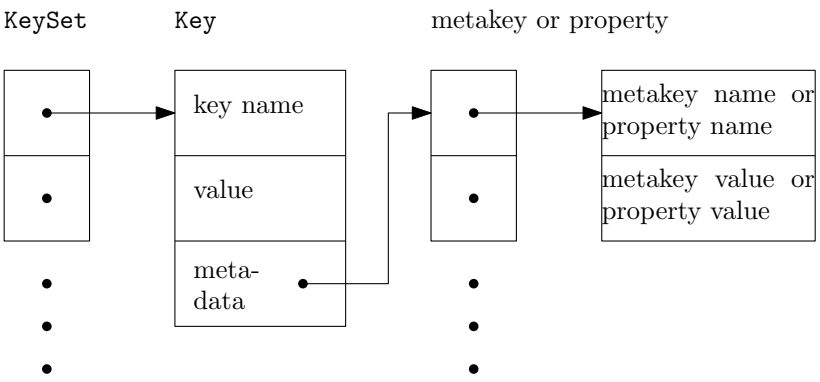
Institute of Information Systems Engineering, TU Wien



KeySet

KeySet

The common data structure between plugins and applications:



Grammar

Idea

Use configuration file format grammar to describe both configurations and (meta-)specifications

⟨*KeySet*⟩ ::= ‘ksNew’␣(’ { ⟨*Key*⟩ ‘, ↩’ } { ‘␣’ } ‘KS_END);’

⟨*Key*⟩ ::= ‘keyNew ␣(” ’ ⟨*key name*⟩ ‘” , ↩’ [⟨*Value*⟩] ⟨*properties*⟩ ‘KEY_END)’

⟨*Value*⟩ ::= { ‘␣’ } ‘KEY_VALUE, ␣” ’ ⟨*configuration value*⟩ ‘” , ↩’

⟨*properties*⟩ ::= { { ‘␣’ } ⟨*property*⟩ ‘, ↩’ }

⟨*property*⟩ ::= ‘KEY_META, ␣ ’ `<property name>` ‘ , ␣ ’ `<property value>` ‘ ’

Example

Example

Given the key `/slapd/threads/listener`, with the configuration value `4` and the property DEFAULT $\mapsto$ 1, ELEKTRA emits:

```
1 ksNew (keyNew ("/slapd/threads/listener",
2             KEY_VALUE, "4",
3             KEY_META, "default", "1",
4             KEY_END),
5         KS_END);
```

Finding

We have source code representing the settings. If we instantiate it, we get a data structure representing the settings. Plugins emitting such "configuration files" are code generators.

Usage in Applications

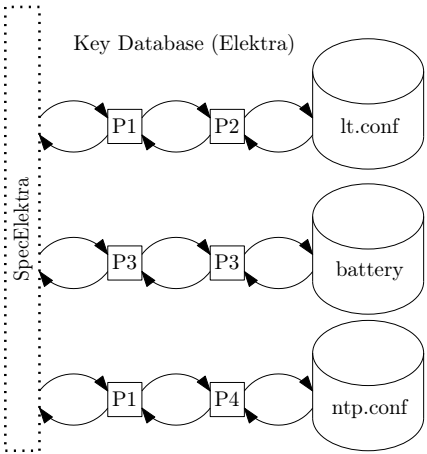
With the specification:

```
[slapd/threads/listener]
  check/range := 1,2,4,8,16
  default := 1
  visibility := advanced
  restrict/write := 1
```

GenElektra gives the user read-only access to the object `env.slapd.threads.listener`:

```
    std::cout << env.slapd.threads.listener;
    env.slapd.threads.listener = 3; // error
```

Implementation



Cylinders are configuration files, P? are plugins [22].

- syntax is defined via plugins reading/writing configuration files
- semantics are defined via
  - plugins interpreting properties
  - generated code used by applications

kdb.open(): The first step is to bootstrap into a situation where the necessary plugins can be loaded.

kdb.get(`KeySet`): The application (initially) fetches and (later) updates its configuration settings as a key set of type `KeySet` from the execution environment by one or many calls to `kdb.get`.

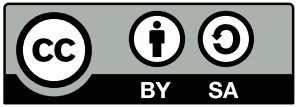
kdb.set(`KeySet`): When a user finishes editing configuration settings, `kdb.set` is in charge of writing all changes back to the key database.

kdb.close(): The last step is to close the connection to the key database.

L01 Configuration Settings

Markus Raab

Institute of Information Systems Engineering, TU Wien



Meeting

Language of the Talk?

Task

A English
B Slightly Prefer English
C Both are fine
D Slightly Prefer German
E German
F Don't care

Task

Which language do you prefer?
You can speak in either English or German in any case.

Video

I am keeping meetings short with many breaks.
You are allowed to:

- stretch
- move
- eat
- look somewhere else
- leave your place

Task

But please turn video on.

Interesting Topics

- CM-Tools: Ansible, Puppet IIIII
- configuration versioning II
- Validation II
- Spring Initializr (https://start.spring.io/) I
- Infrastructure CM I
- Specification and Integration I
- Integration (Common view on different configuration sources) I
- Kubernetes CM I
- Docker Compose YAML I

Question

Did other topics come up?

Learning Outcomes

Students will be able to

- remember definitions of configuration settings.

Task

Break.

Misconfiguration

Question

What are misconfigurations?

- *misconfigurations* [4, 26, 33, 34] are a major cause of system failures [18, 20, 32]
- much time is needed to fix misconfigurations [15, 18, 24, 34]

Configuration Setting

Question

Define Configuration Settings.

Definition

A ***configuration setting***, or ***setting*** in short, fulfills these properties:

1. It is provided by the execution environment.
2. It is *consumed* by an application.
3. It consists of a key, a configuration value, and potentially *metadata*. The ***configuration value***, or ***value*** in short, influences the application's behavior.
4. It can be *produced* by the maintainer, user, or system administrator of the software.

Learning Outcomes

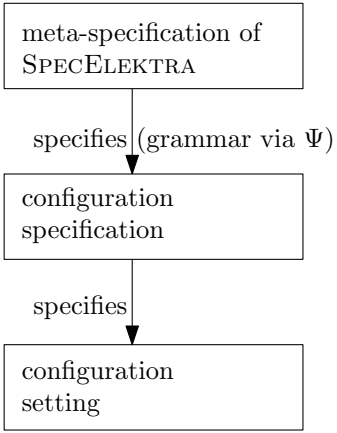
Students will be able to

- describe three metalevels of configuration

Metalevels

Question

Describe the three Metalevels in Elektra.



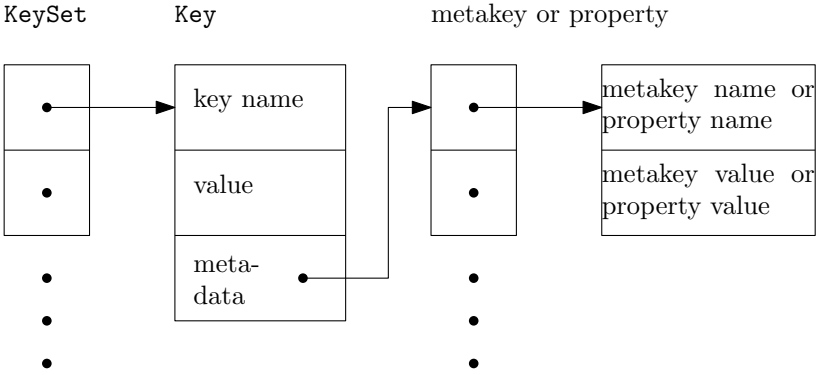
Task

Break.

KeySet

Question

Describe the common data structure in Elektra.



Task

Is meta-data separated from or included in the data structure KeySet?

Install Elektra

Task

Did you already install Elektra? How did you do it?

For example via Docker image:

- Debian Buster
- Alpine: `docker run -it elektra/elektra`

Task

Demo afterwards.

Pull Requests

Task

H0: public pull request

- build server and reviews take time
- make sure to modify `doc/news/_preparation_next_release.md` according to instructions
- we use automatic formatter of code (can also be done via Docker)

Task

Break.

First Steps with Elektra

Task

How have your first steps been?

Variant

Task

H1: choose variant

- Do you already have ideas for FLOSS?

Teamwork/Project

Task

T0/P0: Topics?

Feedback

- Materials?
- Accessibility?
- Any suggestions for improvements?

Preview Next Week

L02: Configuration Specification Languages

- to avoid misconfiguration
- to allow systematic introspection
- see TUWEL

D0: Install Elektra

[1] Eric Arnold Anderson. *Researching system administration*. PhD thesis, University of California at Berkeley, 2002.

[2] Paul Anderson. Towards a high-level machine configuration system. In *LISA*, volume 94, pages 19–26, 1994.

[3] Richard Anthony, DeJiu Chen, Mariusz Pelc, Magnus Persson, and Martin Törngren. Context-aware adaptation in DySCAS. *Electronic Communications of the EASST*, 19, 2009. URL `http://gala.gre.ac.uk/5533/`.

[4] Mona Attariyan and Jason Flinn. Automating configuration troubleshooting with dynamic information flow analysis. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–11, Berkeley, CA, USA, 2010. USENIX Association.

[5] Lionel Cons and Piotr Poznanski. Pan: A high-level configuration language. In *LISA*, volume 2, pages 83–98, 2002. URL `http://static.usenix.org/events/lisa02/tech/full_papers/cons/cons_html/`.

[6] Fernando J. Corbató, Fernando H. Saltzer, and C. T. Clingen. Multics: The first seven years. In *Proceedings of the May 16-18, 1972, Spring Joint Computer Conference*, AFIPS '72 (Spring), pages 571–583, New York, NY, USA, 1972. ACM. doi: 10.1145/1478873.1478950. URL http://dx.doi.org/10.1145/1478873.1478950.

[7] Software Productivity Consortium Services Corporation. *Reuse-driven Software Processes Guidebook: SPC-92019-CMC, Version 02.00. 03*. Software Productivity Consortium Services Corporation, 1993.

[8] Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid, and Andrzej Wąsowski. Cool features and tough decisions: A comparison of variability modeling approaches. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '12, pages 173–182, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1058-1. doi: 10.1145/2110147.2110167. URL http://dx.doi.org/10.1145/2110147.2110167.

[9] Ian Goldberg, David Wagner, Randi Thomas, and Eric A. Brewer. A secure environment for untrusted helper applications: Confining the wily hacker. In *Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography*, volume 6, pages 1–1, 1996.

[10] Sebastian Günther, Thomas Cleenewerck, and Viviane Jonckers. Software variability: the design space of configuration languages. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, pages 157–164. ACM, 2012. URL `http://dl.acm.org/citation.cfm?id=2110165`.

[11] 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.

[12] Peng Huang, William J. Bolosky, Abhishek Singh, and Yuanyuan Zhou. ConfValley: a systematic configuration validation framework for cloud services. In *EuroSys*, page 19, 2015.

[13] Dongpu Jin, Xiao Qu, Myra B. Cohen, and Brian Robinson. Configurations everywhere: Implications for testing and debugging in practice. In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, pages 215–224, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2768-8. doi: 10.1145/2591062.2591191. URL http://dx.doi.org/10.1145/2591062.2591191.

[14] Zhenkai Liang, V. N. Venkatakrishnan, and R. Sekar. Isolated program execution: an application transparent approach for executing untrusted programs. In *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, pages 182–191, December 2003. doi: 10.1109/CSAC.2003.1254323.

[15] Ratul Mahajan, David Wetherall, and Tom Anderson. Understanding BGP misconfiguration. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '02, pages 3–16, New York, NY, USA, 2002. ACM. ISBN 1-58113-570-X.

[16] Kim Mens, Rafael Capilla, Nicolas Cardozo, Bruno Dumas, et al. A taxonomy of context-aware software variability approaches. In *Workshop on Live Adaptation of Software Systems, collocated with Modularity 2016 conference*, 2016.

[17] Sarah Nadi, Thorsten Berger, Christian Kästner, and Krzysztof Czarnecki. Mining configuration constraints: static analyses and empirical results. In *ICSE*, pages 140–151, 2014.

[18] David Oppenheimer, Archana Ganapathi, and David A. Patterson. Why do Internet services fail, and what can be done about it? In *USENIX Symposium on Internet Technologies and Systems*, volume 67. Seattle, WA, 2003.

[19] Jeff H. Perkins, Sunghun Kim, Sam Larsen, Saman Amarasinghe, Jonathan Bachrach, Michael Carbin, Carlos Pacheco, Frank Sherwood, Stelios Sidiroglou, Greg Sullivan, Weng-Fai Wong, Yoav Zibin, Michael D. Ernst, and Martin Rinard. Automatically patching errors in deployed software. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 87–102, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-752-3. doi: 10.1145/1629575.1629585. URL `http://dx.doi.org/10.1145/1629575.1629585`.

[20] Soila Pertet and Priya Narasimhan. Causes of failure in web applications (cmu-pdl-05-109). *Parallel Data Laboratory*, page 48, 2005.

[21] Markus Raab. Elektra: universal framework to access configuration parameters. *The Journal of Open Source Software*, 1(8):1–2, December 2016. doi: 10.21105/joss.00044. URL `http://dx.doi.org/10.21105/joss.00044`.

[22] 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`.

[23] 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.

[24] Ariel Rabkin and Randy Katz. Static extraction of program configuration options. In *Software Engineering (ICSE), 2011 33rd International Conference on*, pages 131–140. IEEE, 2011.

[25] Z. Cliffe Schreuders, Tanya Jane McGill, and Christian Payne. Towards usable application-oriented access controls: qualitative results from a usability study of SELinux, AppArmor and FBAC-LSM. *International Journal of Information Security and Privacy*, 6(1):57–76, 2012.

[26] Ya-Yunn Su, Mona Attariyan, and Jason Flinn. Autobash: Improving configuration management with operating system causality analysis. pages 237–250, 2007. doi: 10.1145/1294261.1294284. URL `http://dx.doi.org/10.1145/1294261.1294284`.

[27] Sander van der Burg and Eelco Dolstra. Automating system tests using declarative virtual machines. In *Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on*, pages 181–190. IEEE, 2010.

[28] Jilles Van Gurp, Jan Bosch, and Mikael Svahnberg. On the notion of variability in software product lines. In *Software Architecture, 2001. Proceedings. Working IEEE/IFIP Conference on*, pages 45–54. IEEE, 2001.

[29] Karina Villela, Adeline Silva, Tassio Vale, and Eduardo Santana de Almeida. A survey on software variability management approaches. In *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, pages 147–156, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2740-4. doi: 10.1145/2648511.2648527. URL `http://dx.doi.org/10.1145/2648511.2648527`.

[30] Alexander von Rhein, Thomas Thüm, Ina Schaefer, Jörg Liebig, and Sven Apel. Variability encoding: From compile-time to load-time variability. *Journal of Logical and Algebraic Methods in Programming*, 85(1, Part 2):125–145, 2016. ISSN 2352-2208. doi: http://dx.doi.org/10.1016/j.jlamp.2015.06.007. URL `http://www.sciencedirect.com/science/article/pii/S2352220815000577`. Formal Methods for Software Product Line Engineering.

[31] Huai Wang and Wing Kwong Chan. Weaving context sensitivity into test suite construction. In *Automated Software Engineering, 2009. ASE '09. 24th IEEE/ACM International Conference on*, pages 610–614, November 2009. doi: 10.1109/ASE.2009.79.

[32] Avishai Wool. A quantitative study of firewall configuration errors. *Computer*, 37 (6):62–67, 2004.

[33] Tianyin Xu and Yuanyuan Zhou. Systems approaches to tackling configuration errors: A survey. *ACM Comput. Surv.*, 47(4):70:1–70:41, July 2015. ISSN 0360-0300. doi: 10.1145/2791577. URL `http://dx.doi.org/10.1145/2791577`.

[34] Zuoning Yin, Xiao Ma, Jing Zheng, Yuanyuan Zhou, Lakshmi N. Bairavasundaram, and Shankar Pasupathy. An empirical study on configuration errors in commercial and open source systems. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 159–172, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0977-6. doi: 10.1145/2043556.2043572.

[35] Sai Zhang and Michael D. Ernst. Automated diagnosis of software configuration errors. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 312–321, Piscataway, NJ, USA, 2013. IEEE Press. ISBN 978-1-4673-3076-3.

[36] Sai Zhang and Michael D. Ernst. Which configuration option should I change? In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 152–163, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2756-5. doi: 10.1145/2568225.2568251. URL `http://dx.doi.org/10.1145/2568225.2568251`.