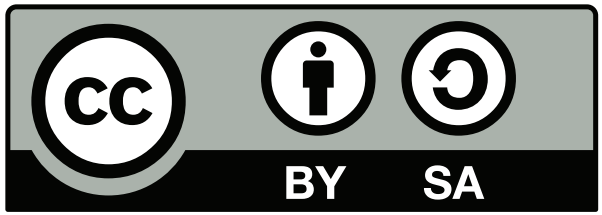
Configuration Management

Markus Raab

Institute of Information Systems Engineering, TU Wien

4.5.2018



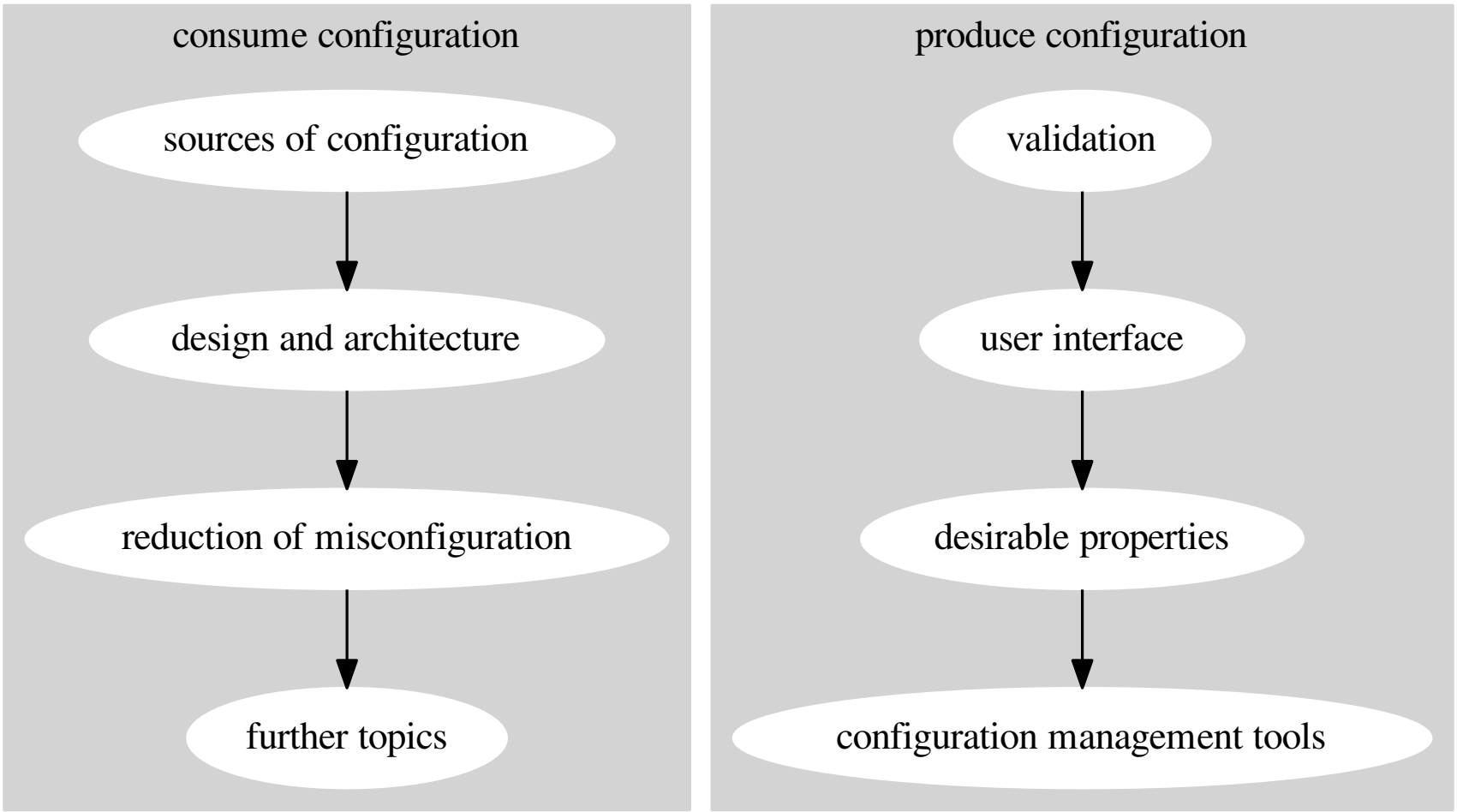
Organization

Next dates:

- 11.5.2018: ???
- 18.5.2018: guest lecture
- 25.5.2018: **team exercise submitted** (1h earlier?)
- 1.6.2018: lecture
- 8.6.2018: lecture
- 15.6.2018: last corrections of team exercise
- 22.6.2018: test

Popular Topics

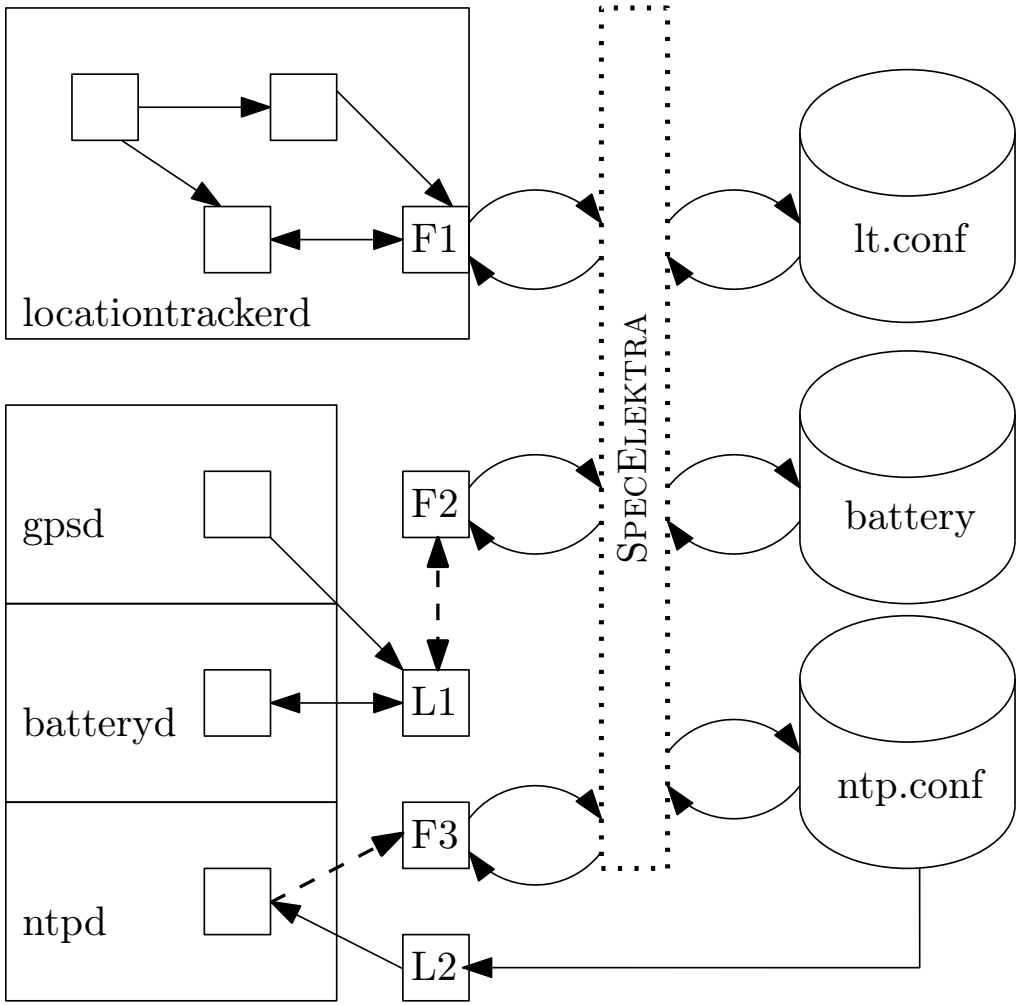
- 4 validation
- 4 user interface
- 3 tools (benefits?)
- 3 testability
- 3 complexity reduction (when conf. needed?)
- 3 architectural decisions
- 2 Puppet
- 2 modularity
- 2 environment variables
- 2 documentation
- 2 configuration specification
- 2 command-line args
- 2 code generation
- 1 variability
- 1 self-description
- 1 round-tripping
- 1 early detection
- 1 introspection
- 1 dependences
- 1 auto-detection
- 1 context-awareness
- 1 administrators


consume configuration
sources of configuration
design and architecture
reduction of misconfiguration
further topics
produce configuration
validation
user interface
desirable properties
configuration management tools

Modularity (Recapitulation)

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 (Recapitulation)



Needed to keep applications independently.
Boxes are applications, cylinders are configuration files, F? are frontends or frontend adapters, L? are configuration libraries [11].

Plugins (Recapitulation)

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.

Introspection (Recapitulation)

Task

What is internal and external specification? What is introspection?

- internal: within applications' source code
- unified get/set access to (meta*)-key/values
- access via applications, CLI, GUI, web-UI, ...
- access via any programming language (similar to file systems)
- GUI, web-UI can semantically interpret metadata
- needed as communication of producers and consumers of configuration
- essential for ***no-futz computing*** Holland et al. [4]

Rationale (Recapitulation)

Task

How to ensure that configuration access points match with present configuration settings?

Configuration Specification:

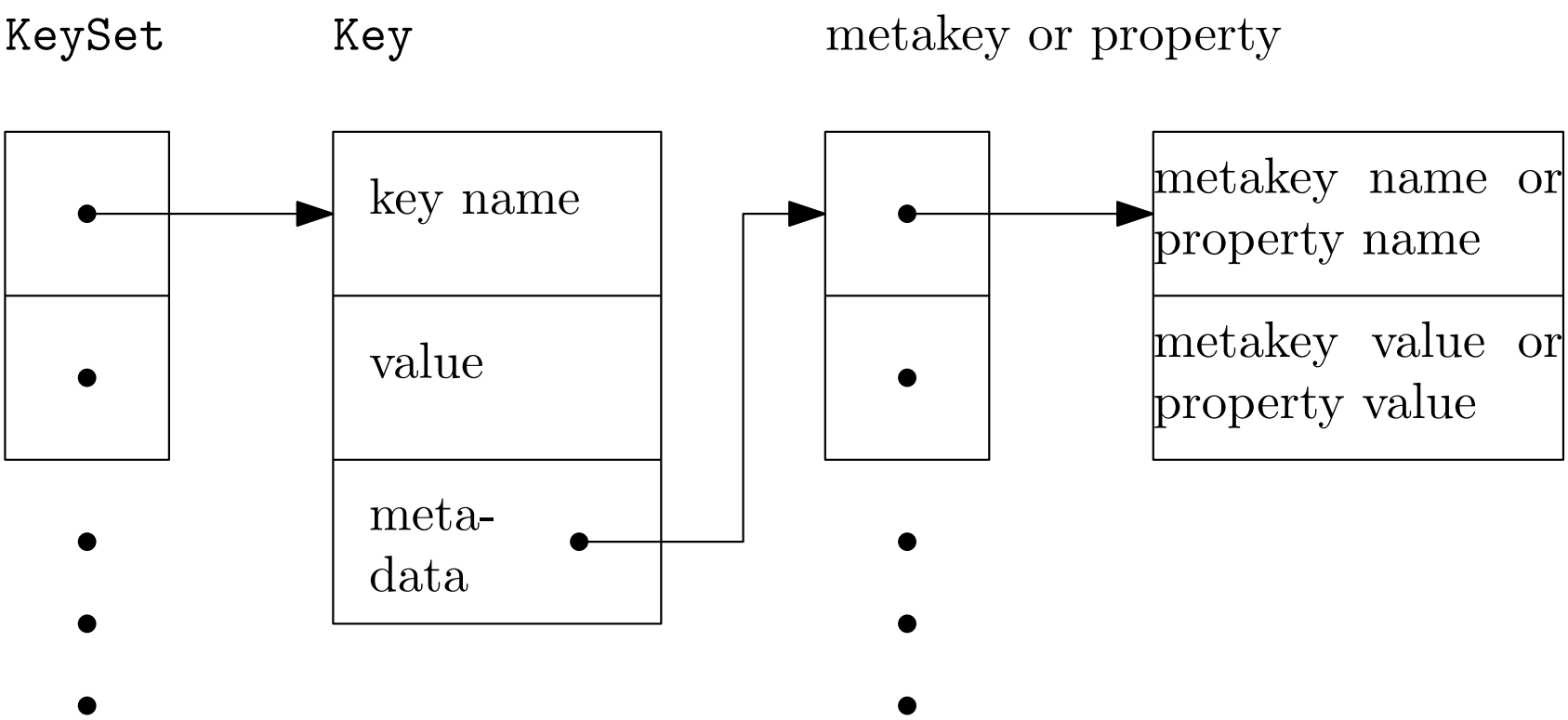
- without specification you and others do not even know which settings are available
- needed for any further techniques we will discuss:
  - code generation guarantees that configuration access points match with specification
  - validation guarantees that configuration settings match with specification

Other Artefacts (Recapitulation):

- examples (e.g., defaults)
- documentation
- auto-completion/syntax highlighting/IDE support
- tooling (GUI, Web UI)
- validation code
- configuration management tool code

KeySet (Recapitulation)

The common data structure between plugins:



KeySet Generation (Recapitulation)

Question

Idea: What if the configuration file format grammar describes source code?

key `spec:/slapd/threads/listener`, with the configuration value `4` and the property default $\mapsto$ 1:

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

Finding

We get source code representing the settings.

Introspection vs. Code Generation? (Recapitulation)

- – more techniques for performance improvements with code generation
- \+ specification can be updated live on the system without recompilation
- \+ tooling has generic access to all specifications
- \+ new features the key database (e.g., better validation) are immediately available consistently

Implication

We generally prefer introspection, except for a very thin configuration access API.

Testing (Recapitulation)

Question

What do we want to test?

- That settings do what they should (devs and admins)
- That settings are properly validated (devs [13])
- Regression tests [10]

- Are all settings implemented?
- Are all settings used in tests?
- Are there unused settings in the code?

Testing by developers? (Recapitulation)

- ConfErr [6] uses models of key board layout, psychology and linguistics. Tool injects possible misconfiguration.
- Spex [13] analyzes the source code to find misconfigurations. As by-product it extracts internal specifications (including transformation bugs).
- External specification can be directly used to generate test cases.
- Find unused configuration settings.

When are settings used? (Recapitulation)

Implementation-time configuration accesses are hard-coded settings in the source code repository. For example, architectural decisions [3] lead to implementation-time settings.

Compile-time configuration accesses are configuration accesses resolved by the build system while compiling the code.

Deployment-time configuration accesses are configuration accesses while the software is installed.

Load-time configuration accesses are configuration accesses during the start of applications.

Run-time configuration accesses are configuration accesses during execution not limited to the startup procedure.

Latent Misconfiguration (Recapitulation)

Phases when we can detect misconfigurations:

- Compilation stage in configuration management tool
- Writing configuration settings on nodes
- Starting applications (load-time)
- When configuration setting is actually used (run-time)

Problem

More context vs. easier to detect and fix.

Documentation

Q: In detail, persons found it very important that (multiple choice, $n \geq 150$, *"You want to configure a FLOSS application. How important are the following ways for you?"*):

- 48 % documentation is shipped with the application
- 36 % configuration examples are shipped with the applications
- 17 % *"google, stackoverflow... (looking for my problem)"*
- 14 % looking at the website of the application
- 14 % use UIs that help them
- 14 % look into the source code
- 11 % *"wiki, tutorials... (looking for complete solutions)"*
- 5 % look into the configuration specification
- 2 % ask colleagues and friends

There are at least two forms of documentation necessary:

- Explanations
- Examples

Generation helps to avoid duplication:

Requirement

There must be a support for shipping correct documentation and examples generated from the configuration specifications.

Question

How to avoid duplication between description text and other parts?

- Render type and defaults into the documentation
- Render requirements and rationale into the documentation
- Use visibility to only show relevant configuration settings

Example

```
1 [slapd/threads/listener]
2   check/range :=1,2,4,8,16
3   default :=1
4   description :=adjust to use more threads
5   rationale :=needed for many-core systems
6   requirement :=1234
7   visibility :=user
```

Reevaluate specifications (Recapitulation)

1. a requirement,
2. an architectural decision,
3. a technical need, and
4. an ad hoc decision.

Notification

Notification Goals

1. transient and persistent configuration settings always in sync
2. avoid polling of configuration settings
3. integrate in already existing mechanisms (main loops)[1]

[1] Is one of the main reasons why most framework already integrate configuration settings.

Conflicts

- If we write out configuration settings, we might conflict with settings recently written.
- Notification intensify this problem (emergent misbehavior)
- but many conflicts can be resolved with a 3-way merge

Upgrading Applications

The same problems happens when upgrading applications.

- System administrator changed the file.
- Package maintainer changed the file.

We can resolve many conflicts automatically, if we consider:

- the key/value structure (vs. line-based)
- the origin of both files
- the type of settings

Conflicts Example

Mine:

```
1 slapd/threads/listener=4
2
3 abc= \
4     def
```

Theirs:

```
1 abc = def
2 slapd/threads/listener=8
```

Origin:

```
1 slapd/threads/listener=8
2 abc=def
```

Context-Awareness

If you're a baker, making bread, you're a baker. If you make the best bread in the world, you're not an artist, but if you bake the bread in the gallery, you're an artist. So the context makes the difference.

— Marina Abramovic

As adapted from Chalmers [1]:

> ***Context*** *is the circumstances relevant to the configuration settings of the application.*

We extend the definition with:

> ***Context-aware configurations*** *are configuration settings that are consistent with its context.* ***Context-aware configuration access*** *is configuration access providing context-aware configuration.*

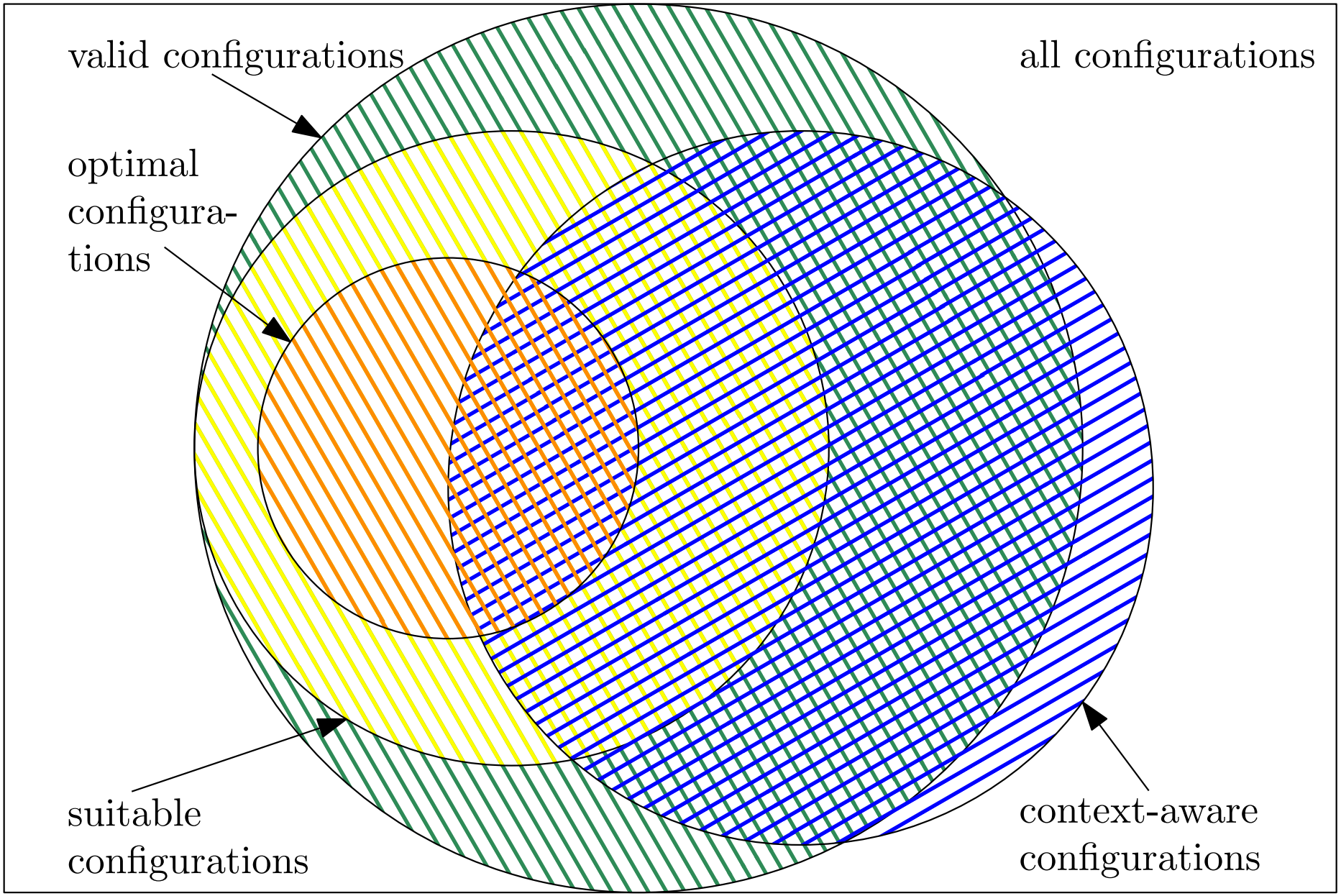
Types of Configuration

Valid configuration does not contradict the present validation specifications. With a valid configuration, applications can start but they may not do what the user wanted or may be inconsistent with context.

Suitable configuration is valid with respect to additional specifications from the user that describe the system the user requires [9].

Optimal configuration is optimal with respect to given optimization criteria. Optimization criteria are important if managing configuration of many computers but are rarely needed for configuration access discussed in this book.

Context-aware configuration is in accordance with its context. Unlike configuration settings, the context changes in ways outside of our control.


valid configurations
all configurations
optimal configura-tions
suitable configurations
context-aware configurations

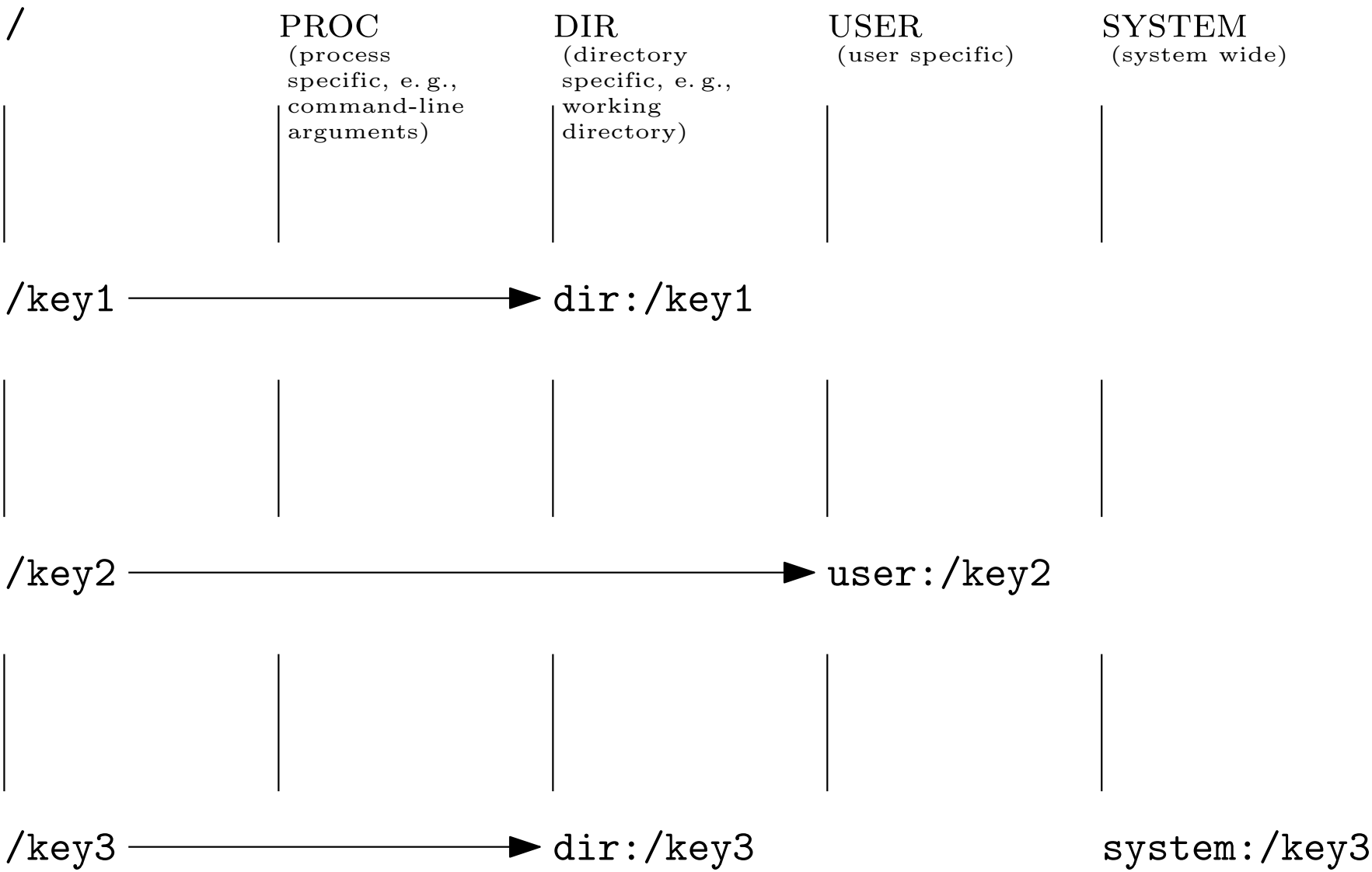
Viewpoints

Sensors: derive context from information sources of the system. Adding new context sensors increases the context available in a system.

Users: Context awareness can be subjective with respect to the needs of a user. For different users, we may need different context specifications. Configuration that is context aware for one user, can be context unaware regarding the wishes of another user. According to Khalil and Connelly [7, 8] personalization is essential.

Time: Because context varies in time, on changes, we need to renew the context awareness of configuration settings. In such situations we speak of ***context changes*** [2, 5].

Cascading (Recapitulation)



Context-aware Lookup

Determine threads from CPUs:

```
1 [cpu]
2   type:=long
3 [slapd/threads/listener]
4   context:=/slapd/threads/%cpu%/listener
```

Determine vibration from sensors:

```
1 [phone/call/vibration]
2   type:=boolean
3   context:=/phone/call/%inpocket%/vibration
```

Determine proxy settings from network:

```
1 [env/override/http_proxy]
2   context:=/http_proxy/%interface%/%network%
```

Team Work

Step 1 (15 min)

Familiarize yourself with one of the configuration systems:

1. environment variables (Factor 12)
2. QSettings
3. Apache Commons Configurations
4. any other system you want to...

Step 2 (45 min)

Come together in a group

1. decide who will create flip chart, moderate, and present
2. explain the system you looked into
3. explain what the system could do for the task
4. create together an architecture that fulfils the goals

Goals

1. early detection of misconfiguration
2. smooth upgrades
3. transient and persistent configuration consistent
4. documentation
5. context-aware
6. reuse software as much as possible but integrate them nicely

Requirements

Design a camera-system that provides:

1. should be able to take single pictures and streams
2. has modular camera modules which can be plugged into the system
3. should have camera profiles for different vendors
4. a Web-UI that shows all configuration settings
5. support remote configuration protocol (Web, SNMP, CMs, ...)
6. time synchronization via NTP
7. should be easily upgradeable with rescue if the upgrade fails

Tasks

1. design and architecture of configuration settings
2. design and architecture of configuration access
3. design decisions (which languages, which systems)
4. how to integrate the system
5. how to ensure smooth upgrades
6. design notifications
7. provide documentation for operators
8. ensure context-aware
9. reuse software as much as possible but integrate them nicely

[1] Daniel Chalmers. *Contextual mediation to support ubiquitous computing*. PhD thesis, University of London, 2002.

[2] Kurt Geihs, Paolo Barone, Frank Eliassen, Jacqueline Floch, Rolf Fricke, Eli Gjørven, Svein O. Hallsteinsen, Geir Horn, Mohammad Ullah Khan, Alessandro Mamelli, George A. Papadopoulos, Nearchos Paspallis, Roland Reichle, and Erlend Stav. A comprehensive solution for application-level adaptation. *Software: Practice and Experience*, 39(4): 385–422, 2009. ISSN 1097-024X. URL `http://dx.doi.org/10.1002/spe.900`.

[3] Neil B Harrison, Paris Avgeriou, and Uwe Zdun. Using patterns to capture architectural decisions. *Software, IEEE*, 24(4): 38–45, 2007. ISSN 0740-7459. doi: 10.1109/MS.2007.124.

[4] 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.

[5] Tetsuo Kamina, Tomoyuki Aotani, Hidehiko Masuhara, and Tetsuo Tamai. Context-oriented software engineering: A modularity vision. In *Proceedings of the 13th International Conference on Modularity*, MODULARITY '14, pages 85–98, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2772-5.

[6] Lorenzo Keller, Prasang Upadhyaya, and George Candea. Conferr: A tool for assessing resilience to human configuration errors. In *Dependable Systems and Networks With FTCS and DCC, 2008.*, pages 157–166. IEEE, 2008.

[7] Ashraf Khalil and Kay Connelly. *Context-Aware Configuration: A Study on Improving Cell Phone Awareness*, pages 197–209. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-31890-3. doi: 10.1007/11508373_15. URL http://dx.doi.org/10.1007/11508373_15.

[8] Ashraf Khalil and Kay Connelly. *Improving Cell Phone Awareness by Using Calendar Information*, pages 588–600. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-31722-7. doi: 10.1007/11555261_48. URL http://dx.doi.org/10.1007/11555261_48.

[9] Manuele Kirsch-Pinheiro, Raúl Mazo, Carine Souveyet, and Danillo Sprovieri. Requirements analysis for context-oriented systems. *Procedia Computer Science*, 83:253–261, 2016. ISSN 1877-0509. doi: http://dx.doi.org/10.1016/j.procs.2016.04.123. URL `http://www.sciencedirect.com/science/article/pii/S1877050916301466`. The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016) / The 6th International Conference on Sustainable Energy Information Technology (SEIT-2016) / Affiliated Workshops.

[10] Xiao Qu, Myra B. Cohen, and Gregg Rothermel. Configuration-aware regression testing: An empirical study of sampling and prioritization. In *Proceedings of the 2008 International Symposium on Software Testing and Analysis*, ISSTA '08, pages 75–86, New York, NY, USA, 2008. ACM.

ISBN 978-1-60558-050-0. doi: 10.1145/1390630.1390641. URL http://doi.acm.org/10.1145/1390630.1390641.

[11] 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.

[12] 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.

[13] Tianyin Xu, Jiaqi Zhang, Peng Huang, Jing Zheng, Tianwei Sheng, Ding Yuan, Yuanyuan Zhou, and Shankar Pasupathy. Do not blame users for misconfigurations. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 244–259. ACM, 2013.